Try it yourself

Induction - Exam Questions

/ 3 P

b) *Induction:* Consider the sequence $\{L_n\}_{n\geq 1}$ defined via $L_1 = L_2 = 1$ and the recurrence $L_{n+1} = L_n + 2L_{n-1}$ for $n \geq 2$.

Show that for all $n \in \mathbb{N} \setminus \{0\}$, the following equation holds:

$$\sum_{i=1}^{n} 2^{n-i} \cdot L_i^2 = L_n L_{n+1}.$$

b) **(This subtask is from August 2019 exam).** Let $T : \mathbb{N} \to \mathbb{R}$ be a function that satisfies the following two conditions:

$$T(n) \geq 4 \cdot T(\tfrac{n}{2}) + 3n \qquad \text{whenever } n \text{ is divisible by 2;}$$
$$T(1) = 4.$$

Prove by mathematical induction that

$$T(n) \geq 6n^2 - 2n$$

holds whenever n is a power of 2, i.e., $n = 2^k$ with $k \in \mathbb{N}_0$.

Asymptotic Notation - Exam Question

claim	true	false
$(2n + n^2 + 3)^2 = \Theta(n^4)$	☐	☐
$\frac{n}{\log n} \leq \mathcal{O}(\sqrt{n})$	☐	☐
$\log(n!) = \Theta(n \log n)$	☐	☐
$\sum_{i=1}^{\log_5 n} 5^i \geq \Omega(n \log n)$	☐	☐

Mini-exam (Induction + Asymptotic Notation)

/ 5 P

a) *Asymptotic notation quiz:* For each of the following claims, state whether it is true or false. You get 1P for a correct answer, -1P for a wrong answer, 0P for a missing answer. You get at least 0 points in total. FS23

Assume $n \geq 4$.

Claim	true	false
$n^3 + n^4 = \Theta(n^4)$	☐	☐
$n^{10} \leq O(\log(n)^{100})$	☐	☐
$1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leq O(2^n)$	☐	☐
Suppose $a_1 = 1$ and $a_{i+1} = 3a_i + 1$ for all $i \geq 2$. Then $a_n \leq O(4^n)$.	☐	☐
$2^{10\log(n)} = \Theta(2^{20\log(n)})$	☐	☐

/ 4 P

b) *Induction:* Consider the Fibonacci numbers $(F_n)_{n\in\mathbb{N}}$, which are given by $F_0 = 0$, $F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$. Show by mathematical induction that for any integer $n \geq 0$, HS21

$$F_{n+1}^2 \geq \sum_{k=0}^{n} F_k^2.$$

Hint: *Use the facts that $F_{n+1} \geq F_n$ and $F_n \geq 0$ for all $n \in \mathbb{N}$ (you don't need to justify that).*

Loop Counting – Exam Questions

FS 23 **Theory Task T2.** / 15 P

In this part, you should **justify your answers briefly**.

/ 4 P a) *Counting iterations:* For the following code snippets, derive an asymptotic bound for the number of times f is called. Simplify the expression as much as possible and state it in Θ-notation as concisely as possible.

i) Snippet 1:

Algorithm 1

```
for i = 1, ..., n do
    for j = 1, ..., i^2 do
        f()
    f()
```

ii) Snippet 2:

Algorithm 2

```
for i = 1, ..., n do
    k ← 1
    while k ≤ i^2 do
        f()
        k ← 2k
    f()
```

Searchs / Sorts I

Exam Questions

/ 2 P

e) *Sorting algorithms:*

Below you see four sequences of snapshots, each obtained during the execution of one of the following five algorithms: `InsertionSort`, `SelectionSort`, ~~`QuickSort`~~, `MergeSort`, and `BubbleSort`. For each sequence, write down the corresponding algorithm.

8	6	4	2	5	1	3	7
6	4	2	5	1	3	7	8
4	2	5	1	3	6	7	8

Algorithm:

8	6	4	2	5	1	3	7
1	6	4	2	5	8	3	7
1	2	4	6	5	8	3	7

Algorithm:

8	6	4	2	5	1	3	7
6	8	2	4	1	5	3	7
2	4	6	8	1	3	5	7

Algorithm:

8	6	4	2	5	1	3	7
6	8	4	2	5	1	3	7
4	6	8	2	5	1	3	7

Algorithm:

/ 4 P

g) *Sorting algorithms quiz:* For each of the following claims, state whether it is true or false. You get 1P for a correct answer, -1P for a wrong answer, 0P for a missing answer. You get at least 0 points in total.

Claim	**true**	**false**
There exist arrays of length n which can be sorted with `BubbleSort` after $\Theta(n)$ swaps.	☐	☐
There exist arrays of length n for which the runtime of `InsertionSort` is $\Theta(n)$.	☐	☐
Consider a sequence of n numbers $\{x_1, ..., x_n\}$, where $0 \leq x_i \leq 1000$, $\forall i = 1, ..., n$, is given as input. There exists an algorithm with runtime $O(n)$ which sorts any such sequence.	☐	☐
There exist a comparison-based sorting algorithm that can sort any array of length n in runtime $O(n)$.	☐	☐

/ 3 P g) *Sorting algorithms:*

i) Consider the sequence $6, 5, 4, 1, 2, 3$. How many swaps does `Bubble Sort` perform to sort this sequence? *Give the exact number of swaps required.*

ii) Consider the sequence $6, 5, 4, 1, 2, 3$. How many swaps does `Selection Sort` perform to sort this sequence? *Give the exact number of swaps required.*

iii) Let $n \in \mathbb{N}$ be an even number and consider the sequence with the following structure:

$$2, 1, 4, 3, 6, 5, \ldots, n, n-1.$$

How many swaps does `Insertion Sort` perform to sort this sequence? *Give the exact number, not just the asymptotics.*

Formal Correctness Proof Example

Bubble Sort

Algorithm 1 Bubble Sort (input: array $A[1 \ldots n]$).

```
for j = 1, ..., n do
    for i = 1, ..., n − 1 do
        if A[i] > A[i + 1] then
            Swap A[i] and A[i + 1]
```

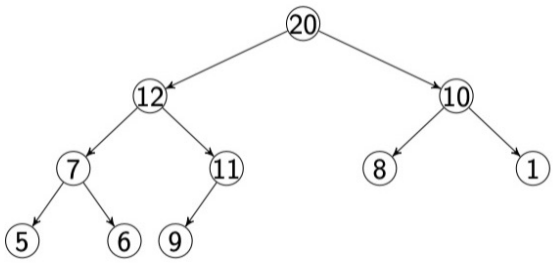
Prove correctness of this algorithm by mathematical induction.

Hint: *Use the invariant $I(j)$ that was introduced in the lecture: "After j iterations the j largest elements are at the correct place."*

Heap Tree - Exam Questions

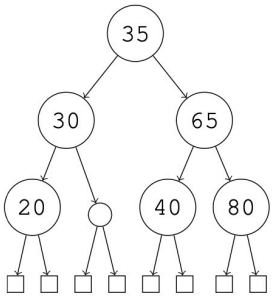
/ 1 P a) *Min-Heap*: Draw the Min-Heap that is obtained when inserting into an empty heap the keys `8, 3, 2, 7, 4, 1` in this order.

/ 1 P b) *Max-Heap*: Draw the resulting Max-Heap obtained from the following Max-Heap by performing the operation `DELETE-MAX` **twice**.



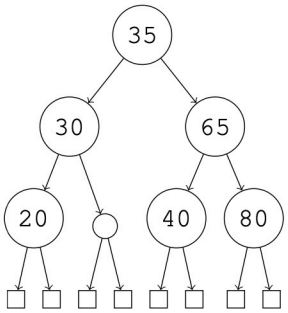
BST + AVL - exam questions

i) Draw the **binary search tree** obtained from the following tree by performing the two operations `INSERT`(45) and `DELETE`(30), in that order.



ii) Draw the **AVL tree** that is obtained when inserting the keys $3, 2, 7, 6, 8, 9$ in this order into an empty tree (it suffices to draw only the final tree).

iii) Draw the **AVL tree** that is obtained by deleting key 65 from the tree below.



DP - exam question

Theory Task T3. **/ 9 P**

You are given an array of n natural numbers $a_1, \ldots, a_n \in \mathbb{N}$, and two natural numbers $A, B \in \mathbb{N}$. You want to determine whether there is a subset $I \subseteq \{1, \ldots, n\}$ satisfying

$$\sum_{i \in I} a_i = A \quad \text{and} \quad \sum_{i \in I} a_i^2 = B.$$

For example,

- The answer for the input $(a_i)_{i \leq n} = [2, 4, 8, 1, 4, 5, 3]$, $A = 8$ and $B = 30$ is *yes* because the set of indices $I = \{1, 4, 6\}$, which corresponds to $(a_i)_{i \leq I} = [2, 1, 5]$, yields the *sum* $2 + 1 + 5 = 8$ and the *sum-of-squares* $2^2 + 1^2 + 5^2 = 30$.
- The answer for the input $(a_i)_{i \leq n} = [2, 4, 8, 1]$, $A = 6$ and $B = 15$ is *no*.

Provide a *dynamic programming* algorithm that determines whether such a subset I exists. In order to get full points, your algorithm should have an $O(n \cdot A \cdot B)$ runtime. Address the following aspects in your solution:

1) *Definition of the DP table*: What are the dimensions of the table $DP[\ldots]$? What is the meaning of each entry ?
2) *Computation of an entry*: How can an entry be computed from the values of other entries ? Specify the base cases, i.e., the entries that do not depend on others.
3) *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps ?
4) *Extracting the solution*: How can the final solution be extracted once the table has been filled ?
5) *Running time*: What is the running time of your algorithm ? Provide it in Θ-notation in terms of n, A and B, and justify your answer.

Size of the DP table / Number of entries: _____

Meaning of a table entry:

Scheme continues on the next page.

Computation of an entry (initialization and recursion):

Order of computation:

Extracting the result:

Running time:

DP - mock Exam

Theory Task T3. / 9 P

In this problem, you are given an array $A = [a_1, \ldots, a_n]$ of n pairwise distinct positive integers, i.e. $a_i \in \mathbb{Z}_{\geq 1}$ for $i \in \{1, \ldots, n\}$ and $a_i \neq a_j$ for $i \neq j \in \{1, \ldots, n\}$.

/ 7 P a) Provide a *dynamic programming* algorithm that, given an array A of n pairwise distinct positive integers as above, returns `True` if there are two different (non-empty) subsets I_1 and I_2 of $\{1, \ldots, n\}$ (i.e. $\emptyset \neq I_1, I_2 \subseteq \{1, \ldots, n\}$ and $I_1 \neq I_2$) such that $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$ and `False` otherwise. In particular, the algorithm does *not* have to return the sets I_1 and I_2.

For example,

- For the array $[2, 3, 4, 5, 7]$ the output should be `True` since for $I_1 = \{1, 2, 4\}$ and $I_2 = \{2, 5\}$ we have $\sum_{i \in I_1} a_i = 2 + 3 + 5 = 10 = 3 + 7 = \sum_{i \in I_2} a_i$.
- For the array $[2, 3, 4, 10, 20]$ the output should be `False` since there are no two different non-empty subsets I_1 and I_2 of $\{1, 2, 3, 4, 5\}$ that satisfy $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$.

In order to obtain full points, your algorithm should run in time $O(n \cdot S)$, where $S = \sum_{i=1}^{n} a_i$. In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Hint: It might be helpful to think about the following more general problem: For any integer $1 \leq m \leq S$, determine how many different subsets $I \subseteq \{1, \ldots, n\}$ there are such that $m = \sum_{i \in I} a_i$.

Dimensions of the DP table:

Scheme continues on the next page.

Subproblems:

Recursion:

Calculation order:

Scheme continues on the next page.

Extracting the solution:

Running time:

/ 2 P b) Show that for any array A as above, the following two conditions are equivalent.

i) There are non-empty sets $I_1, I_2 \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$ and $I_1 \neq I_2$.

ii) There are non-empty sets $I_1, I_2 \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$ and $I_1 \cap I_2 = \emptyset$.

DP - Practice

Theory Task T3. / 9 P

You are given an array of n natural numbers $a_1, \ldots, a_n \in \mathbb{N}$, and two natural numbers $A, B \in \mathbb{N}$. You want to determine whether there is a subset $I \subseteq \{1, \ldots, n\}$ satisfying

$$\sum_{i \in I} a_i = A \quad \text{and} \quad \sum_{i \in I} a_i^2 = B.$$

For example,

- The answer for the input $(a_i)_{i \leq n} = [2, 4, 8, 1, 4, 5, 3]$, $A = 8$ and $B = 30$ is *yes* because the set of indices $I = \{1, 4, 6\}$, which corresponds to $(a_i)_{i \leq I} = [2, 1, 5]$, yields the *sum* $2 + 1 + 5 = 8$ and the *sum-of-squares* $2^2 + 1^2 + 5^2 = 30$.
- The answer for the input $(a_i)_{i \leq n} = [2, 4, 8, 1]$, $A = 6$ and $B = 15$ is *no*.

Provide a *dynamic programming* algorithm that determines whether such a subset I exists. In order to get full points, your algorithm should have an $O(n \cdot A \cdot B)$ runtime. Address the following aspects in your solution:

Theory Task T3. / 9 P

Let $A = [a_1, a_2, \ldots, a_n]$ be an array of non-negative integers. Given A and a non-negative integers $S \geq 0$, we want to determine whether S can be written as a (non-repeating) sum of elements of A, where we are allowed to take the square of elements. Formally, we want to determine if there exist $I, J \subseteq \{1, 2, \ldots, n\}$ with $I \cap J = \emptyset$, $I \cup J = \{1, 2, \ldots, n\}$ such that:

$$S = \sum_{i \in I} a_i + \sum_{j \in J} a_j^2.$$

Provide a *dynamic programming* algorithm that outputs `True` if this is possible, and `False` otherwise. For example, 2-

- The inputs $A = [2, 4, 4]$, $S = 34$ should result in `True`, since $34 = 2 + 4^2 + 4^2$.
- The inputs $A = [2, 4, 4]$, $S = 35$ should result in `False`.
- The inputs $A = [2, 4, 3, 22]$, $S = 21$ should result in `False`.

In order to obtain full points, your algorithm should run in time $O(n^2 \cdot S)$. Address the following aspects of your solution:

Theory Task T3. / 9 P

You are given an array of n natural numbers $a_1, \ldots, a_n \in \mathbb{N}$ summing to $A := \sum_{i=1}^{n} a_i$, which is a multiple of 3. You want to determine whether it is possible to partition $\{1, \ldots, n\}$ into three disjoint subsets I, J, K such that the corresponding elements of the array yield the same sum, i.e.

$$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{k \in K} a_k = \frac{A}{3}.$$

Note that I, J, K form a partition of $\{1, \ldots, n\}$ if and only if $I \cap J = I \cap K = J \cap K = \emptyset$ and $I \cup J \cup K = \{1, \ldots, n\}$.

For example, the answer for the input $[2, 4, 8, 1, 4, 5, 3]$ is *yes*, because there is the partition $\{3, 4\}$, $\{2, 6\}$, $\{1, 5, 7\}$ (corresponding to the subarrays $[8, 1]$, $[4, 5]$, $[2, 4, 3]$, which are all summing to 9). On the other hand, the answer for the input $[3, 2, 5, 2]$ is *no*.

Provide a *dynamic programming* algorithm that determines whether such a partition exists. Your algorithm should have an $\mathcal{O}(nA^2)$ runtime to get full points. Address the following aspects in your solution:

DP - Practice

1)

$a_1, \ldots, a_n \in \mathbb{N}$ $A, B \in \mathbb{N}$. $I \subseteq \{1, \ldots, n\}$

$$\sum_{i \in I} a_i = A \quad \text{and} \quad \sum_{i \in I} a_i^2 = B.$$

your algorithm should have an $O(n \cdot A \cdot B)$ runtime.

Size of DP table :

Meaning of a table entry :

Initialization, Recursion:

Order, Extracting solution, Running time :

---

2) $A = [a_1, a_2, \ldots, a_n]$ $S \geq 0$,

$I, J \subseteq \{1, 2, \ldots, n\}$ with $I \cap J = \emptyset$, $I \cup J = \{1, 2, \ldots, n\}$

$$S = \sum_{i \in I} a_i + \sum_{j \in J} a_j^2.$$

your algorithm should run in time $O(n^2 \cdot S)$.

Size of DP table :

Meaning of a table entry :

Initialization, Recursion:

Order, Extracting solution, Running time :

---

3) $a_1, \ldots, a_n \in \mathbb{N}$ summing to $A := \sum_{i=1}^{n} a_i$, multiple of 3.

$I \cap J = I \cap K = J \cap K = \emptyset$ $I \cup J \cup K = \{1, \ldots, n\}$.

$$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{k \in K} a_k = \frac{A}{3}.$$

algorithm should have an $\mathcal{O}(nA^2)$ runtime

Size of DP table :

Meaning of a table entry :

Initialization, Recursion:

Order, Extracting solution, Running time :

Graph Definitions – Graph Quiz (exam question)

/ 5 P

c) *Graph quiz:* For each of the following claims, state whether it is true or false. You get 1P for a correct answer, -1P for a wrong answer, 0P for a missing answer. You get at least 0 points in total.

As a reminder, here are a few definitions for a (directed) graph $G = (V, E)$:

For $k \geq 2$, a (directed) *walk* is a sequence of vertices $v_1, \ldots, v_k$ such that for every two consecutive vertices v_i, v_{i+1}, we have $\{v_i, v_{i+1}\} \in E$ (resp. $(v_i, v_{i+1}) \in E$ for a directed walk).

A (directed) *closed walk* is a (directed) walk with $v_1 = v_k$.

A (directed) *cycle* is a (directed) closed walk where $k \geq 3$ and all vertices (except v_1 and v_k) are distinct.

A (directed) *closed Eulerian walk* is a (directed) closed walk which traverses every edge in E exactly once.

For a vertex v in a directed graph $G = (V, E)$, the *in-degree* of v is the number of edges in E that end in v (i.e., of the form (w, v)), and the *out-degree* of v is the number of edges in E that start in v (i.e., of the form (v, w)).

Claim	**true**	**false**
A connected graph must contain a cycle.	☐	☐
A graph $G = (V, E)$ with $\lvert E\rvert \leq \lvert V\rvert - 1$ is a tree.	☐	☐
Let $G = (V, E)$ be a graph with $\lvert E\rvert \geq 4$, which contains a closed Eulerian walk. If we remove one edge from E, the resulting graph does not contain a closed Eulerian walk, no matter which edge we remove (the vertex set does not change).	☐	☐
Let $G = (V, E)$ be a *directed* graph. If the in-degree and out-degree of every vertex $v \in V$ is even, then G contains a *directed* closed Eulerian walk.	☐	☐

Topological Sorting - Exam Question

/ 3 P

d) *Directed Acyclic Tournament*

A *tournament* is a directed graph $G = (V, E)$ such that:

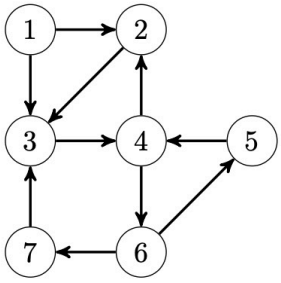
- G has no self loops, i.e., $(v, v) \notin E$, for all $v \in V$. (Note that the graphs that we usually consider have no self loops.)
- For every two distinct vertices $u, v \in V$, either $(u, v) \in E$ or $(v, u) \in E$ but not both.

Let G be a directed acyclic graph that is also a tournament. Show that G has a unique topological sorting.

DFS - Exam Questions

/ 2 P

d) *Depth-first search:* Consider the following directed graph:

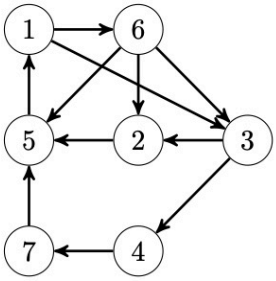


i) Draw the depth-first tree resulting from a depth-first search starting from vertex 1. Process the neighbors of a vertex in increasing order.

ii) Write out two edges e_1, e_2 such that the directed graph above has a topological ordering after removing e_1 and e_2 (the vertex set does not change).

Remark: There could be multiple valid solutions. In this case, you only need to write down one of them.

/ 2 P d) *Depth-first search:* Consider the following directed graph:



i) Draw the depth-first tree resulting from a depth-first search starting from vertex 1. Process the neighbors of a vertex in increasing order.

ii) Write out all the cross edges and all the back edges (specify which ones are cross edges, and which ones are back edges).

BFS - Exam Question

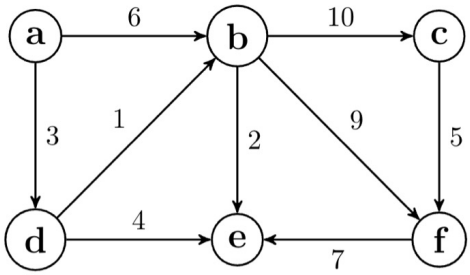
/ 4 P

c) *Finding a shortest cycle*

Describe an algorithm which, given an unweighted directed graph $G = (V, E)$ and a vertex $v \in V$, finds a shortest cycle containing v. If there is no such cycle, the algorithm should report that v is not a vertex of any cycle. Faster algorithms are worth more points. To get full points, aim for $O(|V| + |E|)$ runtime.

Dijkstra's Algo - Exam Questions

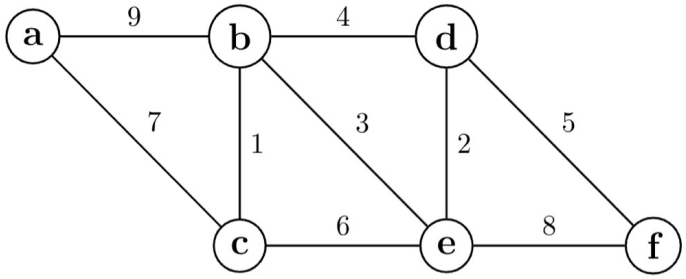
/ 2 P f) *Shortest Path Tree:* Consider the following graph:



i) Highlight the edges that are part of the shortest-path tree rooted at vertex a (i.e., the output of Dijkstra's algorithm if we were to start from vertex a).

ii) Does the above graph have a topological ordering? If yes, write down one topological ordering. If no, give an argument.

/ 2 P f) *Shortest Path Tree:* Consider the following graph:



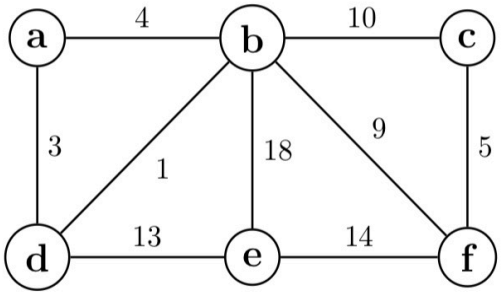
i) Highlight the edges that are part of the shortest-path tree rooted at vertex a (i.e., the output of Dijkstra's algorithm if we were to start from vertex a).

ii) Write out all positive integers x such that we could replace the weight 8 of the edge $\{e, f\}$ in the above graph by x, such that the edge would be in at least one shortest-path tree rooted at a of the resulting graph.

MST – Exam Question

/ 2 P

e) *Minimum Spanning Tree:* Consider the following graph:



i) Highlight the edges that are part of the minimum spanning tree. (Either in the picture above, or you can recreate the graph below).

ii) Write out all positive integers x such that if we replace the weight 1 of edge $\{b, d\}$ in the above graph with x, then edge $\{b, d\}$ would be in at least one minimum spanning tree of the resulting graph.

Graph Modelling

/ 7 P

a) Alice lives in Zurich and wants to visit her grandma in Stockholm. Since she has $n \in \mathbb{N}$ full weeks of vacation time from September 1st, she would like to use some of this time travel by train from Zurich to Stockholm and discover nice cities along the way. However, as a student, Alice may have a lot of time, but not a lot of money. Hence, while she does not mind spending long hours in the train, she would like to minimize her costs. You are tasked with helping her find the cost of the cheapest route from Zurich to Stockholm.

You are given a timetable composed of N rows $(r_1, \ldots, r_N)$, each representing one possible (direct) train journey. Let $\mathcal{C}$ be the set of cities. Each row r_i is of the form $r_i = (dt_i, at_i, dc_i, ac_i, p_i)$ where $dt_i \in \mathbb{N}$ is the departure time of the train, $at_i \in \mathbb{N}$ its arrival time, $cd_i \in \mathcal{C}$ its departure city, $ac_i \neq dc_i \in \mathcal{C}$ its arrival city, and $p_i \in \mathbb{N}$ the price of the journey in CHF. All departure and arrival times are expressed in minutes from September 1st, midnight. There are $D = 1440$ minutes in a day. You are guaranteed that the timetable allows Alice to travel from Zurich to Stockholm in less than n weeks, or $7 \cdot D \cdot n$ minutes. Note that as a Youth Hostel Premium Member, Alice does not need to pay anything to stay in the various cities she travels through.

i) Model the problem as a graph problem, so that you can solve it using an algorithm from the lecture. Describe the set of vertices, the set of edges and the weights. How many vertices and edges does your graph have? What is the corresponding graph problem?

ii) Which algorithm from the lecture can you use to solve this graph problem? Justify why you can use this algorithm, and state its asymptotic running time (with Θ) in terms of the number of vertices $|V|$ and edges $|E|$ in the graph.

/ 7 P

b) Alice will likely use trains of a famous railway company on her way, and these trains are notoriously unreliable. In fact, each train can be cancelled arbitrarily. Being aware of this situation, Alice wants to plan for the worst and know how much more expensive the journey can become if trains are cancelled.

Alice has now selected her preferred journey. She will visit cities $c_1 = \text{Zurich}, c_2, \ldots, c_{T+1} = \text{Stockholm}$ in this order, with an expected total cost t. The timetable is the same as before, but now, even if trains are cancelled, Alice will always follow the same routes $c_1 \to c_2$, ..., $c_T \to c_{T+1}$.

For each of the T journeys, Alice buys the ticket at the departure station shortly before the train starts. When buying a ticket, Alice can be sure that her next train will work, but she never knows whether further connections will behave as expected. The only thing that the railway company **does** guarantee to Alice is that she will be able to reach Stockholm from Zurich within n weeks eventually.

i) Model the problem as a longest path problem in a directed acyclic graph. Describe the set of vertices, the set of edges and the weights. Prove that your directed graph is acyclic and explain why the longest path problem you propose provides the right solution to the problem.

ii) Design an efficient algorithm that determines how much more expensive Alice's journey can become. In order to get full points, your algorithm should have $O(|V||E|)$ runtime.

Hint: What happens if you replace every edge with cost w by an edge with cost $-w$? Recall that, as proven in Task 2, a longest path always exists.

Solutions

Induction - Exam Questions

FS 23

/ 3 P b) *Induction:* Consider the sequence $\{L_n\}_{n\geq 1}$ defined via $L_1 = L_2 = 1$ and the recurrence $L_{n+1} = L_n + 2L_{n-1}$ for $n \geq 2$.

Show that for all $n \in \mathbb{N} \setminus \{0\}$, the following equation holds:

$$\sum_{i=1}^{n} 2^{n-i} \cdot L_i^2 = L_n L_{n+1}.$$

Base Case: $(n=1)$ $\sum_{i=1}^{1} 2^{1-i} \cdot L_i^2 = 2^0 \cdot L_1^2 = 1 \cdot L_1^2 = 1 = 1 \cdot 1 = L_1 \cdot L_2$

* Expand the sum if you get stuck ∇

I.H.: $k \in \mathbb{N} \setminus \{0\}$. Assume that $\sum_{i=1}^{k} 2^{k-i} \cdot L_i^2 = 2^{k-1} \cdot L_1^2 + 2^{k-2} \cdot L_2^2 + \ldots + L_k^2 = L_k L_{k+1}$

I.S.: $2^k L_1^2 + 2^{k-1} L_2^2 + \ldots + 2L_k^2 + L_{k+1}^2$

$= 2(2^{k-1} L_1^2 + 2^{k-2} L_2^2 + \ldots + L_k^2) + L_{k+1}^2$

$\overset{\text{I.H.}}{=} 2(L_k L_{k+1}) + L_{k+1}^2$

$= L_{k+1}(2L_k + L_{k+1})$

$\overset{\text{def}}{=} L_{k+1} L_{k+2}$

b) **(This subtask is from August 2019 exam).** Let $T : \mathbb{N} \to \mathbb{R}$ be a function that satisfies the following two conditions:

$T(n) \geq 4 \cdot T(\frac{n}{2}) + 3n$ whenever n is divisible by 2; definition (can be applied everywhere)
$T(1) = 4.$

Prove by mathematical induction that

$\star$ $T(n) \geq 6n^2 - 2n$ (property to prove)

holds whenever n is a power of 2, i.e., $n = 2^k$ with $k \in \mathbb{N}_0$. (ind. over which variable?)

Base Case: $(k=0)$ $n = 2^0 = 1$ $\quad T(2^0) = T(1) \overset{\text{def}}{=} 4 \geq 6 \cdot 1^2 - 2 \cdot 1 = 4$

I.H.: Property holds for $b = 2^a$ for some $a \in \mathbb{N}_0$

$$T(2^a) \geq 6 \cdot (2^a)^2 - 2 \cdot (2^a)$$

I.S.: $(2^{a+1} = 2 \cdot b)$

$$\begin{aligned}
T(2^{a+1}) &\overset{\text{def}}{\geq} 4 \cdot T(2^a) + 3 \cdot 2^{a+1} \\
&\overset{\text{I.H.}}{\geq} 4 \cdot (6 \cdot (2^a)^2 - 2 \cdot 2^a) + 3 \cdot 2^{a+1} \\
&= 24 \cdot 2^{2a} - 4 \cdot 2^{a+1} + 3 \cdot 2^{a+1} \\
&= 24 \cdot 2^{2a} - 2^{a+1} \\
&= 6 \cdot 4 \cdot 2^{2a} - 2^{a+1} \\
&= 6 \cdot 2^{2a+2} - 2^{a+1} \\
&= 6 \cdot (2^{a+1})^2 - 2^{a+1} \\
&\geq 6 \cdot (2^{a+1})^2 - 2 \cdot (2^{a+1})
\end{aligned}$$

$4 \cdot (-2 \cdot 2^a) = -8 \cdot 2^a = -4 \cdot 2 \cdot 2^a = -4 \cdot 2^{a+1}$

$4 \cdot 2^{2a} = 2^2 \cdot 2^{2a} = 2^{2a+2}$

$2^{2a+2} = 2^{2(a+1)} = (2^{a+1})^2$

last step explanation: If we know y is ≥ 0, then this implies $x - y \geq x - 2y$

Asymptotic Notation - Exam Question

FS 20

claim	true	false
$(2n + n^2 + 3)^2 = \Theta(n^4)$	☒	☐
$\frac{n}{\log n} \leq \mathcal{O}(\sqrt{n})$	☐	☒
$\log(n!) = \Theta(n \log n)$	☒	☐
$\sum_{i=1}^{\log_5 n} 5^i \geq \Omega(n \log n)$	☐	☒

Asymptotic Notation
Mini cheat-sheet

$\lim_{n\to\infty}$: $1 < \log(\log(n)) < \log(n) < \sqrt{n} < n < n\cdot\log(n) < n\cdot\sqrt{n} < n^2 < 2^n < n! < n^n$

Sums

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Geometric series : $\sum_{k=0}^{n} q^k = \frac{q^{n+1}-1}{q-1}$

$$\sum_{k=0}^{3} 3^k = 3^0 + 3^1 + 3^2 + 3^3 = \frac{3^4-1}{3-1} = 40$$

Factorial

$$\frac{n}{2}^{\frac{n}{2}} \leq n! \leq n^n$$

Θ : $n\cdot\ln n = \Theta(\ln(n!))$

$\Rightarrow n\cdot\ln n \leq O(n!) \wedge n\cdot\ln n \geq \Omega(n!)$

$(n^2+2n+3)^2 = n^4 + \ldots$ $\quad n^4$

$\frac{\frac{n}{\log n}}{\sqrt{n}} = \frac{\sqrt{n}}{\log n} \to \infty$

$\sum_{i=1}^{\log_5 n} 5^i = \frac{5^{\log_5 n + 1} - 1}{5-1} - 5^0$

$= \frac{5n-1}{4} - 1 \rightsquigarrow n$

$n \geq \Omega(n \log n)$

$\log(\frac{n}{2}^{\frac{n}{2}}) \leq \log(n!) \leq \log(n^n)$

$\log(n^n) = n\log(n)$

$= \frac{n}{2}\log(\frac{n}{2})$

$= \frac{n}{2}(\log n - \log 2)$

$= \frac{n\log n}{2} - \frac{n\log 2}{2}$ $\quad \frac{n\log 2}{2} \to 0$

$= n\log n\,(\frac{1}{2} - \frac{\log 2}{2\log n})$

$n\log(n) \leq \log n! \leq n\log n$

Mini-exam (Induction + Asymptotic Notation)

/ 5 P a) *Asymptotic notation quiz:* For each of the following claims, state whether it is true or false. You get 1P for a correct answer, -1P for a wrong answer, 0P for a missing answer. You get at least 0 points in total. FS23

Assume $n \geq 4$.

Claim	true	false
$n^3 + n^4 = \Theta(n^4)$	☒	☐
$n^{10} \leq O(\log(n)^{100})$	☐	☒
$1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leq O(2^n)$	☐	☒
Suppose $a_1 = 1$ and $a_{i+1} = 3a_i + 1$ for all $i \geq 2$. Then $a_n \leq O(4^n)$.	☒	☐
$2^{10\log(n)} = \Theta(2^{20\log(n)})$	☐	☒

/ 4 P b) *Induction:* Consider the Fibonacci numbers $(F_n)_{n\in\mathbb{N}}$, which are given by $F_0 = 0$, $F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$. Show by mathematical induction that for any integer $n \geq 0$, HS21

$$F_{n+1}^2 \geq \sum_{k=0}^{n} F_k^2.$$

Hint: *Use the facts that $F_{n+1} \geq F_n$ and $F_n \geq 0$ for all $n \in \mathbb{N}$ (you don't need to justify that).*

Base Case: $n=0$

$$F_1^2 = 1 \geq \sum_{k=0}^{0} F_k^2 = F_0^2 = 0$$

$n=1$

$$F_2^2 \overset{\text{def}}{=} 1 \geq \sum_{k=0}^{1} F_k^2 = F_0^2 + F_1^2 = 0 + 1 = 1$$

I.H.: For some $m \geq 1$

$$F_{m+1}^2 \geq \sum_{k=0}^{m} F_k^2$$

I.S.: $m \to m+1$:

$$\begin{aligned} F_{m+1+1}^2 &\overset{\text{def}}{=} (F_{m+1} + F_m)^2 \\ &= F_{m+1}^2 + 2F_{m+1}F_m + F_m^2 \\ &\geq \sum_{k=0}^{m} F_k^2 + 2F_{m+1}F_m + F_m^2 \\ &\geq \sum_{k=0}^{m} F_k^2 + 2F_{m+1}F_m \qquad | F_m \geq 0,\ F_m^2 \geq 0 \\ &= \sum_{k=0}^{m} F_k^2 + F_{m+1}(F_m + F_m) \\ &\geq \sum_{k=0}^{m} F_k^2 + \underbrace{F_{m+1}(F_m + F_{m-1})}_{= F_{m+1}^2}. \qquad | F_m \geq F_{m-1} \\ &\geq \sum_{k=0}^{m+1} F_k^2 \qquad | \text{ By the principle ...} \end{aligned}$$

Loop Counting – Exam Questions

FS 23 Theory Task T2.

/ 15 P

In this part, you should **justify your answers briefly**.

/ 4 P

a) *Counting iterations:* For the following code snippets, derive an asymptotic bound for the number of times f is called. Simplify the expression as much as possible and state it in Θ-notation as concisely as possible.

i) Snippet 1:

Algorithm 1

```
for i = 1, ..., n do
    for j = 1, ..., i^2 do
        f()
    f()
```

$$\sum_{i=1}^{n}\left(\left(\sum_{j=1}^{i^2} 1\right)+1\right) = \sum_{i=1}^{n} i^2+1 = \sum_{i=1}^{n} i^2 + n = \frac{n(n+1)(2n+1)}{6} + n = \Theta(n^3)$$

$\sum_{j=1}^{i^2} 1$: $(i^2-1+1)\cdot 1 = i^2$

$\sum_{i=1}^{n} 1 = (n-1+1)\cdot 1 = n$

for the mini cheat sheet:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^2(n+1)^2}{4}$$

ii) Snippet 2:

Algorithm 2

```
for i = 1, ..., n do
    k ← 1
    while k ≤ i^2 do
        f()
        k ← 2k
    f()
```

k: $1, 2, 4, \ldots, i^2$

$2^0, 2^1, 2^2, \ldots, 2^{\log_2 i^2}$

$$\sum_{i=1}^{n}\left(\left(\sum_{j=0}^{\lfloor \log_2 i^2 \rfloor} 1\right)+1\right) = n + \sum_{i=1}^{n}\left(\sum_{j=0}^{\lfloor \log_2 (i^2) \rfloor} 1\right) = n + \sum_{i=1}^{n}\left(\log_2(i^2)\right)$$

(+1 ignored for rest)

$$= n + \sum_{i=1}^{n} 2\log(i)$$

$$= 2\log(1) + 2\log(2) + \ldots + 2\log(n) + n$$

$$= 2\log(n!) + n = \Theta(n \log n)$$

$\left(\frac{n}{2}\right)^{n/2} \leq n! \leq n^n$

Searchs / Sorts I

Exam Questions

/ 2 P

e) *Sorting algorithms:*

Below you see four sequences of snapshots, each obtained during the execution of one of the following five algorithms: `InsertionSort`, `SelectionSort`, ~~`QuickSort`~~, `MergeSort`, and `BubbleSort`. For each sequence, write down the corresponding algorithm.

8	6	4	2	5	1	3	7
6	4	2	5	1	3	7	8
4	2	5	1	3	6	7	8

Algorithm:

8	6	4	2	5	1	3	7
1	6	4	2	5	8	3	7
1	2	4	6	5	8	3	7

Algorithm:

8	6	4	2	5	1	3	7
6	4	2	5	1	3	7	8
4	2	5	1	3	6	7	8

not swapped

Algorithm: Bubble Sort

8	6	4	2	5	1	3	7
1	6	4	2	5	8	3	7
1	2	4	6	5	8	3	7

Algorithm: Selection Sort (variant with min val)

8	6	4	2	5	1	3	7
6	8	2	4	1	5	3	7
2	4	6	8	1	3	5	7

Algorithm:

8	6	4	2	5	1	3	7
6	8	4	2	5	1	3	7
4	6	8	2	5	1	3	7

Algorithm:

8	6	4	2	5	1	3	7
6	8	2	4	1	5	3	7
2	4	6	8	1	3	5	7

Algorithm: Merge Sort

8	6	4	2	5	1	3	7
6	8	4	2	5	1	3	7
4	6	8	2	5	1	3	7

Algorithm: Insertion Sort

/ 4 P g) *Sorting algorithms quiz:* For each of the following claims, state whether it is true or false. You get 1P for a correct answer, -1P for a wrong answer, 0P for a missing answer. You get at least 0 points in total.

Claim	true	false
There exist arrays of length n which can be sorted with `BubbleSort` after $\Theta(n)$ swaps.	☒	☐

[5, 1, 2, 3, 4] n-1 swaps $= \Theta(n)$

[1, 2, 3] no swaps ✓

There exist arrays of length n for which the runtime of `InsertionSort` is $\Theta(n)$.	☐	☒

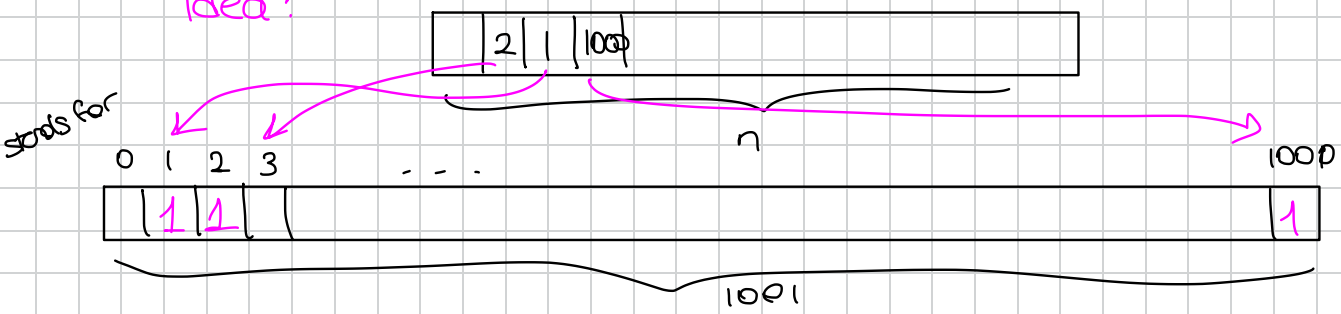
* Insertion Sort uses Binary Search to find the correct positions

Even if the array is already sorted, it's $\Theta(n \log n)$

Consider a sequence of n numbers $\{x_1, ..., x_n\}$, where $0 \leq x_i \leq 1000$, $\forall i = 1, ..., n$, is given as input. There exists an algorithm with runtime $O(n)$ which sorts any such sequence.	☒	☐

Key Realization: 1000 → a constant

Idea:



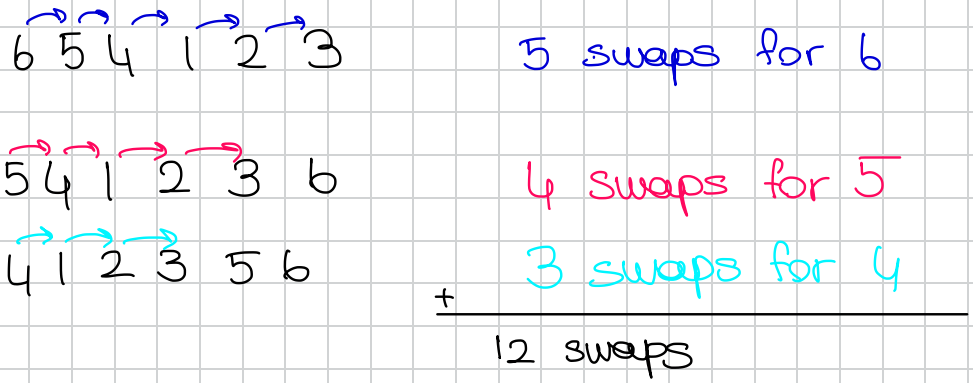
$n + 1001 \leq O(n)$ ✓

There exist a comparison-based sorting algorithm that can sort any array of length n in runtime $O(n)$.	☐	☒

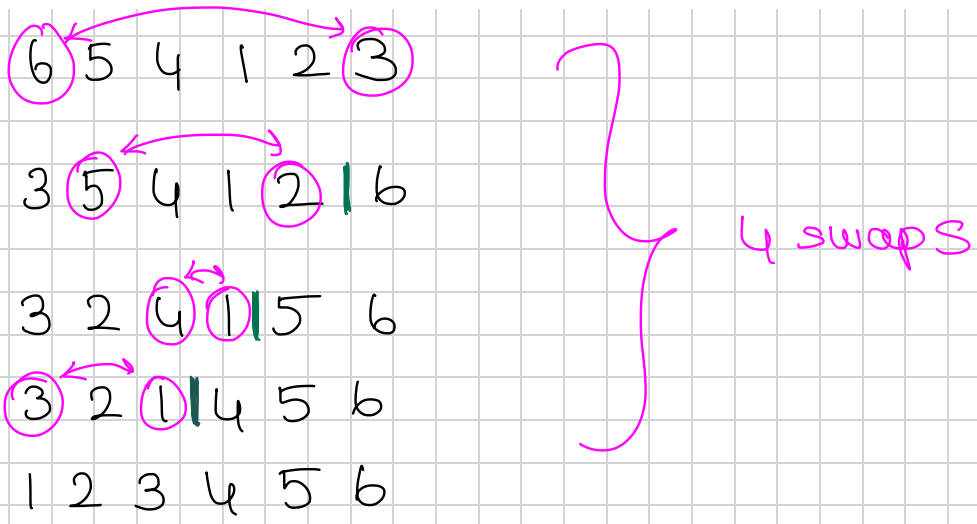
Not possible!

/ 3 P g) *Sorting algorithms:*

i) Consider the sequence $6, 5, 4, 1, 2, 3$. How many swaps does `Bubble Sort` perform to sort this sequence? *Give the exact number of swaps required.*



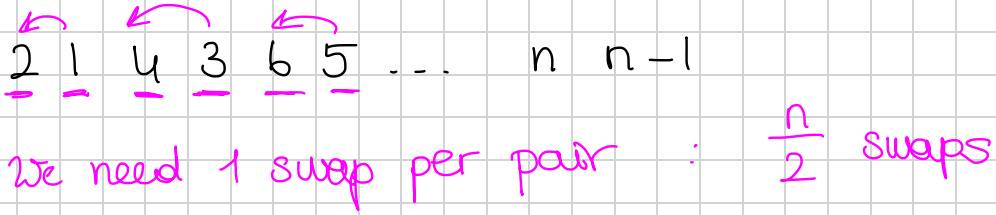
ii) Consider the sequence $6, 5, 4, 1, 2, 3$. How many swaps does `Selection Sort` perform to sort this sequence? *Give the exact number of swaps required.*



iii) Let $n \in \mathbb{N}$ be an even number and consider the sequence with the following structure:

$$2, 1, 4, 3, 6, 5, \ldots, n, n-1.$$

How many swaps does `Insertion Sort` perform to sort this sequence? *Give the exact number, not just the asymptotics.*



Formal Correctness Proof Example

Bubble Sort

Algorithm 1 Bubble Sort (input: array $A[1 \dots n]$).

```
for j = 1, ..., n do
    for i = 1, ..., n − 1 do
        if A[i] > A[i + 1] then
            Swap A[i] and A[i + 1]
```

Prove correctness of this algorithm by mathematical induction.

Hint: *Use the invariant $I(j)$ that was introduced in the lecture: "After j iterations the j largest elements are at the correct place."* → to prove

Solution:

We prove the invariant in the hint by mathematical induction on j.

- **Base Case.**
  We prove the statement for $j = 1$. Assume that the largest element of A is at position l in the beginning. After the first $l - 1$ iterations of the second for-loop, it is still at position l. For all further steps with $i \geq l$, $A[i]$ contains the largest element and thus the largest element is swapped to position $i + 1$. Hence, in the end the largest element is at position n, which shows $I(1)$.

- **Induction Hypothesis.**
  We assume that the invariant is true for $j = k$ for some $k \in \mathbb{N}$, $k < n$, i.e. after k iterations the k largest elements are at the correct position.

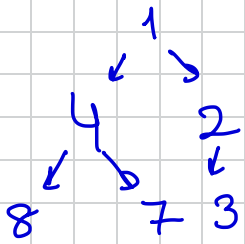
- **Inductive Step.**
  We must show that the invariant also holds for $j = k + 1$. By the induction hypothesis the k largest elements are at the correct position after k steps, i.e. at the positions $A[n - k + 1 \dots n]$. We now consider step $k + 1$. Note that in this iteration the positions of the k largest elements are not changed since for $i \geq n - k$, we will never have $A[i] > A[i + 1]$. Thus, in order to show $I(k + 1)$ it is enough to show that after step $k + 1$ also the $(k + 1)$st largest element is at the correct position. The $(k + 1)$st largest element is the largest element of $A[1 \dots n - k]$ (all elements that are larger than it come later by $I(k)$). Thus, by the argumentation in the base case, after $i = n - k - 1$ iterations in the second for-loop, it is at position $A[n - k]$. But for the other k iterations of the second for-loop, nothing changes as was already argued before (the largest elements do not change their position). Thus, after step $k + 1$, the $k + 1$ largest elements are at the correct position, which shows $I(k + 1)$.

By the principle of mathematical induction, $I(j)$ is true for all $j \in \mathbb{N}$, $j \leq n$. In particular, $I(n)$ holds, which means that after the first n iterations the n largest elements are at the correct position. This shows that after n steps the array is sorted, which shows correctness of the Bubble Sort algorithm.

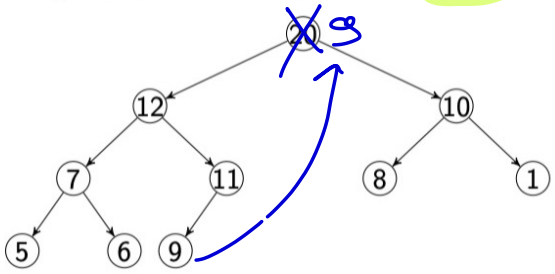
☆ Make sure you have all the key ideas !
↳ try to do it in detail

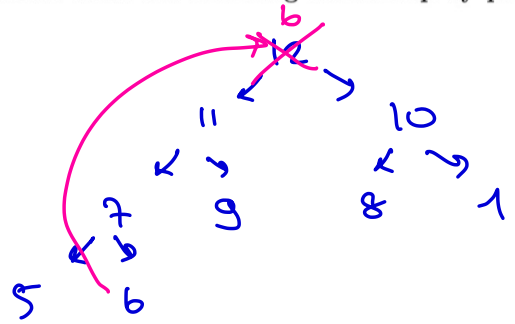
Heap Tree - Exam Questions

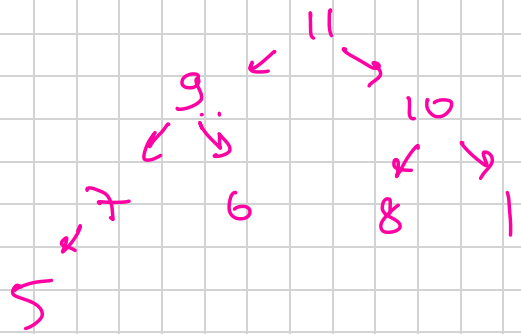
/ 1 P a) *Min-Heap*: Draw the Min-Heap that is obtained when inserting into an empty heap the keys 8, 3, 2, 7, 4, 1 in this order.



/ 1 P b) *Max-Heap*: Draw the resulting Max-Heap obtained from the following Max-Heap by performing the operation DELETE-MAX **twice**.

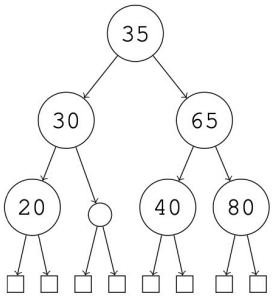




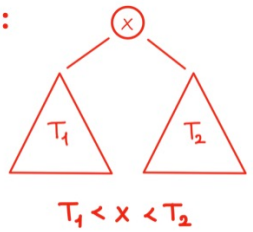


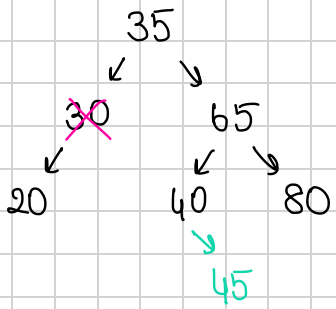
BST + AVL - exam questions

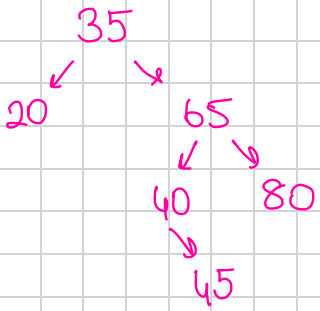
i) Draw the **binary search tree** obtained from the following tree by performing the two operations INSERT(45) and DELETE(30), in that order.



BST Condition :

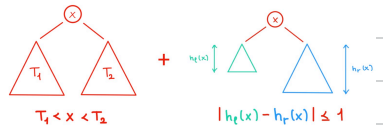


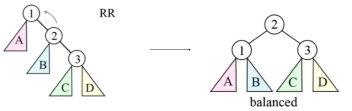


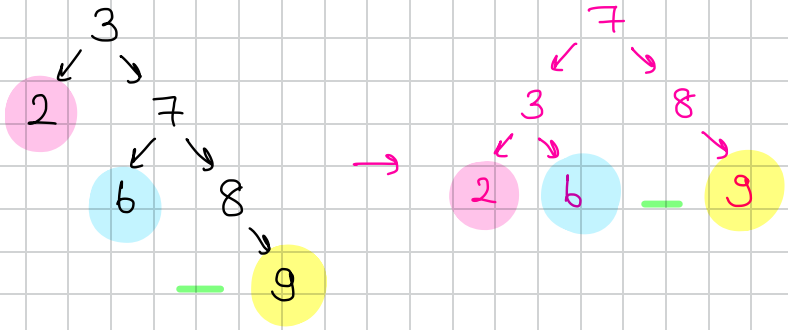


ii) Draw the **AVL tree** that is obtained when inserting the keys $3, 2, 7, 6, 8, 9$ in this order into an empty tree (it suffices to draw only the final tree).

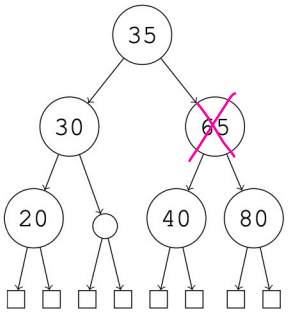
AVL-Tree Condition :



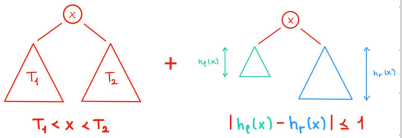




iii) Draw the **AVL tree** that is obtained by deleting key 65 from the tree below.



AVL-Tree Condition :



DP - exam question

Theory Task T3. / 9 P

You are given an array of n natural numbers $a_1, \ldots, a_n \in \mathbb{N}$, and two natural numbers $A, B \in \mathbb{N}$. You want to determine whether there is a subset $I \subseteq \{1, \ldots, n\}$ satisfying

$$\sum_{i \in I} a_i = A \quad \text{and} \quad \sum_{i \in I} a_i^2 = B.$$

Subset Sum

For example,

- The answer for the input $(a_i)_{i \le n} = [2, 4, 8, 1, 4, 5, 3]$, $A = 8$ and $B = 30$ is *yes* because the set of indices $I = \{1, 4, 6\}$, which corresponds to $(a_i)_{i \le I} = [2, 1, 5]$, yields the *sum* $2 + 1 + 5 = 8$ and the *sum-of-squares* $2^2 + 1^2 + 5^2 = 30$.
- The answer for the input $(a_i)_{i \le n} = [2, 4, 8, 1]$, $A = 6$ and $B = 15$ is *no*.

Provide a *dynamic programming* algorithm that determines whether such a subset I exists. In order to get full points, your algorithm should have an $O(n \cdot A \cdot B)$ runtime. Address the following aspects in your solution:

1) *Definition of the DP table*: What are the dimensions of the table $DP[\ldots]$? What is the meaning of each entry ?
2) *Computation of an entry*: How can an entry be computed from the values of other entries ? Specify the base cases, i.e., the entries that do not depend on others.
3) *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps ?
4) *Extracting the solution*: How can the final solution be extracted once the table has been filled ?
5) *Running time*: What is the running time of your algorithm ? Provide it in Θ-notation in terms of n, A and B, and justify your answer.

Size of the DP table / Number of entries: $[0 \ldots n] \times [0 \ldots A] \times [0 \ldots B]$

Meaning of a table entry:

DP[i][a][b] := true if there is $I \subseteq \{1 \ldots i\}$ s.t. $\sum_{i \in I} a_i = a$ $\sum_{i \in I} a_i^2 = b$

false otherwise

Scheme continues on the next page.

Computation of an entry (initialization and recursion):

DP[0][0][0] = true DP[0][x][y] = false for $x, y \geq 1$

DP[i][a][b] = DP[i-1][a][b] ∨ DP[i-1][a-a_i][b-a_i^2]

Consider every entry out-of-bounds to be false

Order of computation:

We compute with increasing order of i, a, b.

Extracting the result:

If DP[n][A][B] == true then the answer is yes

DP[n][A][B] == false then the answer is no

Running time:

We need to fill n+1 × A+1 × B+1 entries

Computation time for each of them is Θ(1)

Therefore running time is Θ(n·A·B)

DP - mock Exam

Theory Task T3. / 9 P

In this problem, you are given an array $A = [a_1, \ldots, a_n]$ of n pairwise distinct positive integers, i.e. $a_i \in \mathbb{Z}_{\geq 1}$ for $i \in \{1, \ldots, n\}$ and $a_i \neq a_j$ for $i \neq j \in \{1, \ldots, n\}$.

/ 7 P a) Provide a *dynamic programming* algorithm that, given an array A of n pairwise distinct positive integers as above, returns `True` if there are two different (non-empty) subsets I_1 and I_2 of $\{1, \ldots, n\}$ (i.e. $\emptyset \neq I_1, I_2 \subseteq \{1, \ldots, n\}$ and $I_1 \neq I_2$) such that $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$ and `False` otherwise. In particular, the algorithm does *not* have to return the sets I_1 and I_2.

For example,

- For the array $[2, 3, 4, 5, 7]$ the output should be `True` since for $I_1 = \{1, 2, 4\}$ and $I_2 = \{2, 5\}$ we have $\sum_{i \in I_1} a_i = 2 + 3 + 5 = 10 = 3 + 7 = \sum_{i \in I_2} a_i$.
- For the array $[2, 3, 4, 10, 20]$ the output should be `False` since there are no two different non-empty subsets I_1 and I_2 of $\{1, 2, 3, 4, 5\}$ that satisfy $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$.

In order to obtain full points, your algorithm should run in time $O(n \cdot S)$, where $S = \sum_{i=1}^{n} a_i$. In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Hint: It might be helpful to think about the following more general problem: For any integer $1 \leq m \leq S$, determine how many different subsets $I \subseteq \{1, \ldots, n\}$ there are such that $m = \sum_{i \in I} a_i$.

Dimensions of the DP table: DP[1...n][1...S] n×S

Scheme continues on the next page.

Subproblems:

$DP[i][s]$:= "the number of different (non-empty) subsets $I \subseteq \{1 \dots i\}$ s.t. $s = \sum_{i \in I} a_i$ "

Recursion:

Init:

$DP[1][A[1]] = 1$

$DP[1][s] = 0$ (if $s \neq A[1]$)

Recursion:

$$DP[i][s] = \begin{cases} 1 + DP[i-1][s] & s - a_i = 0 \\ DP[i-1][s-a_i] + DP[i-1][s] & s - a_i \geq 1 \\ DP[i-1][s] & s - a_i < 0 \end{cases}$$

($a_i = s$) we don't use a_i in $DP[i-1][s]$ then $\{a_i\}$

$DP[i-1][s-a_i]$: we use a_i

$DP[i-1][s]$: what we had without using a_i

(otherwise) we can't use a_i ($a_i > s$)

It works because elements are pairwise distinct

Calculation order:

Calculate $DP[i][s]$ by increasing i, increasing s

Scheme continues on the next page.

Extracting the solution:

Return true if for any $s \in \{1 \dots S\}$ we have $DP[n][s] \geq 2$

Return false otherwise

Running time:

$n \cdot S$ entries

each entry computed in $O(1)$

extracting the solution in $O(S)$

running time to construct the table $= O(n \cdot S)$

/ 2 P b) Show that for any array A as above, the following two conditions are equivalent.

i) There are non-empty sets $I_1, I_2 \subseteq \{1, \dots, n\}$ such that $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$ and $I_1 \neq I_2$.

ii) There are non-empty sets $I_1, I_2 \subseteq \{1, \dots, n\}$ such that $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$ and $I_1 \cap I_2 = \emptyset$.

ii $\Rightarrow$ i) $I_1 \cap I_2 = \emptyset \wedge \sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$

$\Rightarrow I_1 \neq I_2$ (they are non-empty)

$\wedge \sum_{i \in I} a_i = \sum_{i \in I_2} a_i$

i $\Rightarrow$ ii) $I_1, I_2 \subseteq \{1 \dots n\}$ $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$

and $I_1 \neq I_2$

$\overline{I_1} := I_1 \setminus (I_1 \cap I_2)$ $\overline{I_2} := I_2 \setminus (I_1 \cap I_2)$

$I_1 \neq I_2 \Rightarrow \overline{I_1} \neq \overline{I_2}$

$\overline{I_1} \cap \overline{I_2} = \emptyset$ by def

(we remove common elements)

$\sum_{i \in \overline{I_1}} a_i = \sum_{i \in I_1} a_i - \sum_{i \in I_1 \cap I_2} a_i = \sum_{i \in I_2} a_i - \sum_{i \in I_1 \cap I_2} a_i = \sum_{i \in \overline{I_2}} a_i$ $a_i > 0$ and $\overline{I_1} \neq \overline{I_2}$

$\overline{I_1} \neq \overline{I_2} = \emptyset$ and $\sum_{i \in \overline{I_1}} a_i = \sum_{i \in \overline{I_2}} a_i$ $\overline{I_1} \cap \overline{I_2} = \emptyset$ non empty

DP Mini Exam - Proof at the end

In this problem, you are given an array $A = [a_1, \ldots, a_n]$ of n pairwise distinct positive integers, i.e. $a_i \in \mathbb{Z}_{\geq 1}$ for $i \in \{1, \ldots, n\}$ and $a_i \neq a_j$ for $i \neq j \in \{1, \ldots, n\}$.

i ⇔ ii

/ 2 P b) Show that for any array A as above, the following two conditions are equivalent.

i) There are non-empty sets $I_1, I_2 \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$ and $I_1 \neq I_2$.

ii) There are non-empty sets $I_1, I_2 \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$ and $I_1 \cap I_2 = \emptyset$.

ii ⇒ i)

Assume there are non-empty sets $I_1, I_2 \subseteq \{1 \ldots n\}$ s.t. $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$ and $I_1 \cap I_2 = \emptyset$

$\Rightarrow I_1 \neq I_2$

I_1 and I_2 are non-empty + $I_1 \cap I_2 = \emptyset$
I_1 has at least one element that's not in I_2.

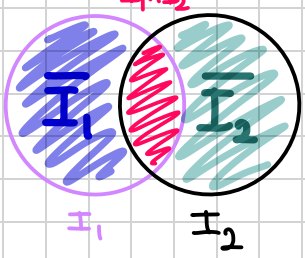
I_1 and I_2 also satisfies this

i ⇒ ii)

Assume there are non-empty sets $I_1, I_2 \subseteq \{1 \ldots n\}$ s.t. $\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$ and $I_1 \neq I_2$

We construct $\overline{I_1}$ and $\overline{I_2}$ s.t. $\overline{I_1}, \overline{I_2} \subseteq \{1 \ldots n\}$ are non-empty, $\sum_{i \in \overline{I_1}} a_i = \sum_{i \in \overline{I_2}} a_i$, $\overline{I_1} \cap \overline{I_2} = \emptyset$

$\overline{I_1} := I_1 \setminus (I_1 \cap I_2)$



$\overline{I_2} := I_2 \setminus (I_1 \cap I_2)$

$I_1 \neq I_2 \Rightarrow \overline{I_1} \neq \overline{I_2}$

we just removed the common elements, they have to have some uncommon elements s.t. $I_1 \neq I_2$ can hold

$\overline{I_1} \cap \overline{I_2} = \emptyset$: By def we've removed all common elements from both The intersection is definitely $\emptyset$

$\sum_{i \in \overline{I_1}} a_i = \sum_{i \in \overline{I_2}} a_i$:

$$\sum_{i \in \overline{I_1}} a_i = \sum_{i \in I_1} a_i - \sum_{i \in I_1 \cap I_2} a_i$$

$$= \sum_{i \in I_2} a_i - \sum_{i \in I_1 \cap I_2} a_i \qquad | \sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$$

$$= \sum_{i \in \overline{I_2}} a_i$$

$\overline{I_1}$ and $\overline{I_2}$ are non-empty : All $a_i > 0$ and $\overline{I_1} \neq \overline{I_2}$ $\Rightarrow$ $\overline{I_1}$ and $\overline{I_2}$ are non-empty

DP - Practice

Theory Task T3. / 9 P

You are given an array of n natural numbers $a_1, \ldots, a_n \in \mathbb{N}$, and two natural numbers $A, B \in \mathbb{N}$. You want to determine whether there is a subset $I \subseteq \{1, \ldots, n\}$ satisfying

$$\sum_{i \in I} a_i = A \quad \text{and} \quad \sum_{i \in I} a_i^2 = B.$$

For example,

- The answer for the input $(a_i)_{i \le n} = [2, 4, 8, 1, 4, 5, 3]$, $A = 8$ and $B = 30$ is *yes* because the set of indices $I = \{1, 4, 6\}$, which corresponds to $(a_i)_{i \le I} = [2, 1, 5]$, yields the *sum* $2 + 1 + 5 = 8$ and the *sum-of-squares* $2^2 + 1^2 + 5^2 = 30$.
- The answer for the input $(a_i)_{i \le n} = [2, 4, 8, 1]$, $A = 6$ and $B = 15$ is *no*.

Provide a *dynamic programming* algorithm that determines whether such a subset I exists. In order to get full points, your algorithm should have an $O(n \cdot A \cdot B)$ runtime. Address the following aspects in your solution:

Theory Task T3. / 9 P

Let $A = [a_1, a_2, \ldots, a_n]$ be an array of non-negative integers. Given A and a non-negative integers $S \ge 0$, we want to determine whether S can be written as a (non-repeating) sum of elements of A, where we are allowed to take the square of elements. Formally, we want to determine if there exist $I, J \subseteq \{1, 2, \ldots, n\}$ with $I \cap J = \emptyset$, $I \cup J = \{1, 2, \ldots, n\}$ such that:

$$S = \sum_{i \in I} a_i + \sum_{j \in J} a_j^2.$$

Provide a *dynamic programming* algorithm that outputs `True` if this is possible, and `False` otherwise. For example,

- The inputs $A = [2, 4, 4]$, $S = 34$ should result in `True`, since $34 = 2 + 4^2 + 4^2$.
- The inputs $A = [2, 4, 4]$, $S = 35$ should result in `False`.
- The inputs $A = [2, 4, 3, 22]$, $S = 21$ should result in `False`.

In order to obtain full points, your algorithm should run in time $O(n^2 \cdot S)$. Address the following aspects of your solution:

Theory Task T3. / 9 P

You are given an array of n natural numbers $a_1, \ldots, a_n \in \mathbb{N}$ summing to $A := \sum_{i=1}^{n} a_i$, which is a multiple of 3. You want to determine whether it is possible to partition $\{1, \ldots, n\}$ into three disjoint subsets I, J, K such that the corresponding elements of the array yield the same sum, i.e.

$$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{k \in K} a_k = \frac{A}{3}.$$

Note that I, J, K form a partition of $\{1, \ldots, n\}$ if and only if $I \cap J = I \cap K = J \cap K = \emptyset$ and $I \cup J \cup K = \{1, \ldots, n\}$.

For example, the answer for the input $[2, 4, 8, 1, 4, 5, 3]$ is *yes*, because there is the partition $\{3, 4\}$, $\{2, 6\}$, $\{1, 5, 7\}$ (corresponding to the subarrays $[8, 1]$, $[4, 5]$, $[2, 4, 3]$, which are all summing to 9). On the other hand, the answer for the input $[3, 2, 5, 2]$ is *no*.

Provide a *dynamic programming* algorithm that determines whether such a partition exists. Your algorithm should have an $\mathcal{O}(nA^2)$ runtime to get full points. Address the following aspects in your solution:

DP - Practice

1) $1 \dots n$ / $1 \dots i$

$a_1, \dots, a_n \in \mathbb{N}$. $A, B \in \mathbb{N}$. $I \subseteq \{1, \dots, n\}$

$$\sum_{i \in I} a_i = A \quad \text{and} \quad \sum_{i \in I} a_i^2 = B.$$

your algorithm should have an $O(n \cdot A \cdot B)$ runtime.

Size of DP table: DP[0...n][0...A][0...B]

Meaning of a table entry:

DP[i][a][b] := There exists $I \subseteq \{1 \dots i\}$ s.t. $\sum_{i \in I} a_i = a$ and $\sum_{i \in I} a_i^2 = b$

Initialization, Recursion:

DP[0][0][0] = 1 DP[0][k][l] = 0 (k,l ≠ 0)

DP[i][a][b] = max { DP[i-1][a][b], DP[i-1][a-a_i][b-a_i^2] }

(Out of bounds is 0) (Justification ...)

Order, Extracting solution, Running time: for i = 1...n, increasing i, DP[n][A][B]

O(n·A·B) (justification)

2) $A = [a_1, a_2, \dots, a_n]$ $S \geq 0$,

$I, J \subseteq \{1, 2, \dots, n\}$ with $I \cap J = \emptyset$, $I \cup J = \{1, 2, \dots, n\}$

$$S = \sum_{i \in I} a_i + \sum_{j \in J} a_j^2.$$

your algorithm should run in time $O(n^2 \cdot S)$.

Size of DP table: DP[0...n][0...S]

Meaning of a table entry:

DP[i][s] := There exists $I, J \subseteq \{1 \dots i\}$ $I \cap J = \emptyset$, $I \cup J = \{1 \dots i\}$ s.t. $s = \sum_{i \in I} a_i + \sum_{j \in J} a_j^2$

Initialization, Recursion:

DP[0][0] = true DP[0][k] (k ≠ 0) = false

DP[i][s] = DP[i-1][s-a_i] ∨ DP[i-1][s-a_i^2]

(Out of bounds is false) (Justification ...)

Order, Extracting solution, Running time: for i = 1...n, for s = 1...S, increasing i, DP[n][s], O(n·S) (justification)

3) $a_1, \dots, a_n \in \mathbb{N}$ summing to $A := \sum_{i=1}^{n} a_i$, multiple of 3.

$I \cap J = I \cap K = J \cap K = \emptyset$ $I \cup J \cup K = \{1, \dots, n\}$.

$$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{k \in K} a_k = \frac{A}{3}.$$

algorithm should have an $\mathcal{O}(nA^2)$ runtime

Size of DP table: DP[0...n][0...$\frac{A}{3}$][0...$\frac{A}{3}$]

Meaning of a table entry:

DP[i][b][c] := There exists 2 disjoint sets $I, J \subseteq \{1 \dots i\}$ s.t. $\sum_{i \in I} a_i = b$, $\sum_{j \in J} a_j = c$

Initialization, Recursion:

DP[0][0][0] = 1 DP[0][k][l] = 0 (k,l ≠ 0)

DP[i][b][c] = max { DP[i-1][b][c], DP[i-1][b-a_i][c], DP[i-1][b][c-a_i] }

(Out of bounds is 0) (Justification ...)

Order, Extracting solution, Running time: for i = 1...n, increasing i, DP[n][$\frac{A}{3}$][$\frac{A}{3}$]

O(n·A^2) (justification)

Graph Definitions – Graph Quiz (exam question)

/ 5 P

c) *Graph quiz:* For each of the following claims, state whether it is true or false. You get 1P for a correct answer, -1P for a wrong answer, 0P for a missing answer. You get at least 0 points in total.

As a reminder, here are a few definitions for a (directed) graph $G = (V, E)$:

For $k \geq 2$, a (directed) *walk* is a sequence of vertices $v_1, \ldots, v_k$ such that for every two consecutive vertices v_i, v_{i+1}, we have $\{v_i, v_{i+1}\} \in E$ (resp. $(v_i, v_{i+1}) \in E$ for a directed walk).

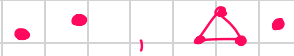
A (directed) *closed walk* is a (directed) walk with $v_1 = v_k$.

A (directed) *cycle* is a (directed) closed walk where $k \geq 3$ and all vertices (except v_1 and v_k) are distinct.

A (directed) *closed Eulerian walk* is a (directed) closed walk which traverses every edge in E exactly once.

For a vertex v in a directed graph $G = (V, E)$, the *in-degree* of v is the number of edges in E that end in v (i.e., of the form (w, v)), and the *out-degree* of v is the number of edges in E that start in v (i.e., of the form (v, w)).

Claim	**true**	**false**
A connected graph must contain a cycle.	☐	☐
A graph $G = (V, E)$ with $\|E\| \leq \|V\| - 1$ is a tree.	☐	☐
Let $G = (V, E)$ be a graph with $\|E\| \geq 4$, which contains a closed Eulerian walk. If we remove one edge from E, the resulting graph does not contain a closed Eulerian walk, no matter which edge we remove (the vertex set does not change).	☐	☐
Let $G = (V, E)$ be a *directed* graph. If the in-degree and out-degree of every vertex $v \in V$ is even, then G contains a *directed* closed Eulerian walk.	☐	☐

	T/F	Justification
A connected graph must contain a cycle.	False	Counterex.: Tree,
A graph $G = (V, E)$ with $\|E\| \leq \|V\| - 1$ is a tree.	False	Counterex.: 
Let $G = (V, E)$ be a graph with $\|E\| \geq 4$, which contains a closed Eulerian walk. If we remove one edge from E, the resulting graph does not contain a closed Eulerian walk, no matter which edge we remove (the vertex set does not change).	True	Remove any edge $\{u, v\}$ deg(u) and deg(v) will decrease by one Initially G contained a closed eulerian w. ⇒ Every vertex in G had even degree deg(u) and deg(v) are odd All vertices don't have even deg ⇒ G can't contain a closed eulerian walk
Let $G = (V, E)$ be a *directed* graph. If the in-degree and out-degree of every vertex $v \in V$ is even, then G contains a *directed* closed Eulerian walk.	False	Counterex.: 

Topological Sorting - Exam Question

/ 3 P

d) *Directed Acyclic Tournament*

A *tournament* is a directed graph $G = (V, E)$ such that:

- G has no self loops, i.e., $(v, v) \notin E$, for all $v \in V$. (Note that the graphs that we usually consider have no self loops.)
- For every two distinct vertices $u, v \in V$, either $(u, v) \in E$ or $(v, u) \in E$ but not both.

Let G be a directed acyclic graph that is also a tournament. Show that G has a unique topological sorting.

Since G is a DAG, it has at least one top. sorting $(v_1, \dots, v_n)$

G is also a tournament $\Rightarrow$ For all i $1 \leq i < n$ either $(v_i, v_{i+1}) \in E$ or (v_{i+1}, v_i) (but not both)

Since $(v_1 \dots v_n)$ is a top. sorting, the edge between v_i and v_{i+1} must be (v_i, v_{i+1}). (It cannot be (v_{i+1}, v_i))

$(v_1, \dots, v_n)$ is a path in G

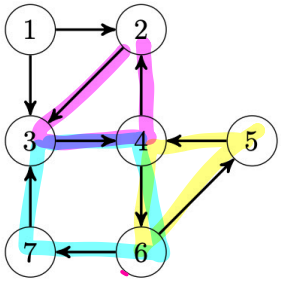
v_i is before v_{i+1} in any top. sorting

$(v_1, \dots, v_n)$ is the unique top. sorting !

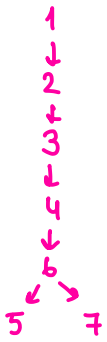
DFS - Exam Questions

/ 2 P

d) *Depth-first search:* Consider the following directed graph:



i) Draw the depth-first tree resulting from a depth-first search starting from vertex 1. Process the neighbors of a vertex in increasing order.

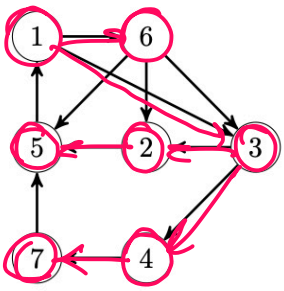


ii) Write out two edges e_1, e_2 such that the directed graph above has a topological ordering after removing e_1 and e_2 (the vertex set does not change).

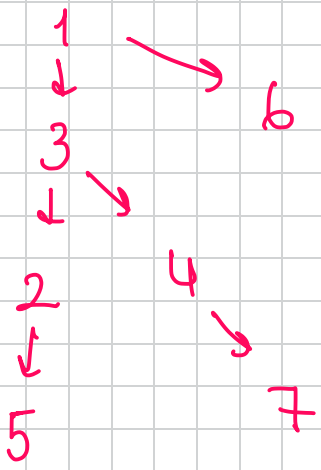
Remark: There could be multiple valid solutions. In this case, you only need to write down one of them.

DAG (4,2) and (4,6)

/ 2 P d) *Depth-first search:* Consider the following directed graph:



i) Draw the depth-first tree resulting from a depth-first search starting from vertex 1. Process the neighbors of a vertex in increasing order.



ii) Write out all the cross edges and all the back edges (specify which ones are cross edges, and which ones are back edges).

(5,1) back edge

(7,5) cross edge

(6,5) cross

(6,2) cross

(6,3) cross

BFS - Exam Question

: input

: output

/ 4 P

c) *Finding a shortest cycle*

Describe an algorithm which, given an unweighted directed graph $G = (V, E)$ and a vertex $v \in V$, finds a shortest cycle containing v. If there is no such cycle, the algorithm should report that v is not a vertex of any cycle. Faster algorithms are worth more points. To get full points, aim for $O(|V| + |E|)$ runtime.

- DFS or BFS

Given G and $v \in V$

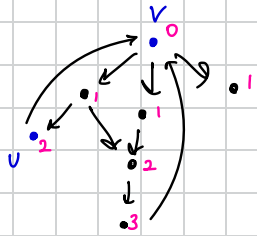
We start the BFS from v

If we don't encounter v for a second time, output "v is not a vertex of any cycle"

If we encounter v for a second time,

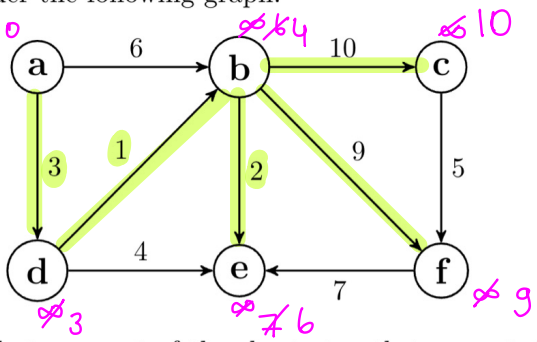
Let u be the vertex from which we've reached v
let P be the shortest path from v to u (already found by BFS)

Adding edge (u,v) to P will give us a shortest cycle containing v.



Dijkstra's Algo - Exam Questions

/ 2 P f) *Shortest Path Tree:* Consider the following graph:

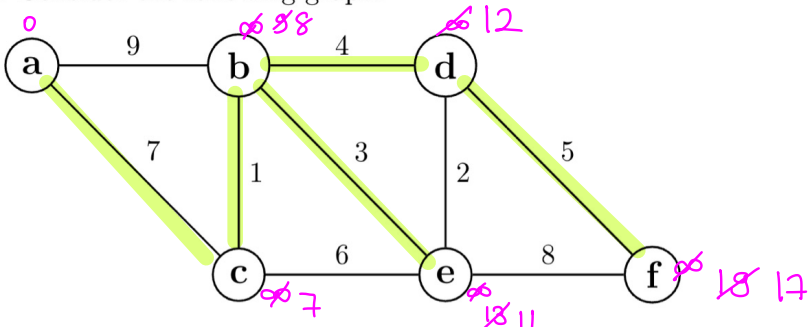


i) Highlight the edges that are part of the shortest-path tree rooted at vertex a (i.e., the output of Dijkstra's algorithm if we were to start from vertex a).

ii) Does the above graph have a topological ordering? If yes, write down one topological ordering. If no, give an argument.

Yes, a, d, b, c, f, e

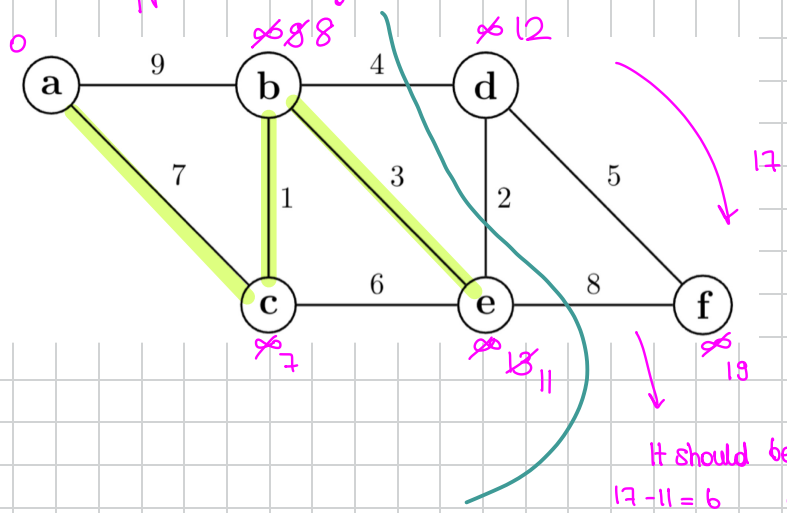
/ 2 P f) *Shortest Path Tree:* Consider the following graph:



i) Highlight the edges that are part of the shortest-path tree rooted at vertex a (i.e., the output of Dijkstra's algorithm if we were to start from vertex a).

ii) Write out ~~all positive integers~~ x such that we could replace the weight 8 of the edge $\{e, f\}$ in the above graph by x, such that the edge would be in at least one shortest-path tree rooted at a of the resulting graph.

what happened there?



It should be ≤ 17 !

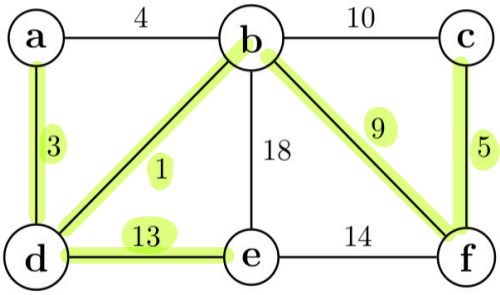
17 - 11 = 6 c(e,f) ≤ 6 !

1, 2, 3, 4, 5, 6

MST - Exam Question

/ 2 P

e) *Minimum Spanning Tree:* Consider the following graph:



Fast Kruskal

Consider edges in increasing order

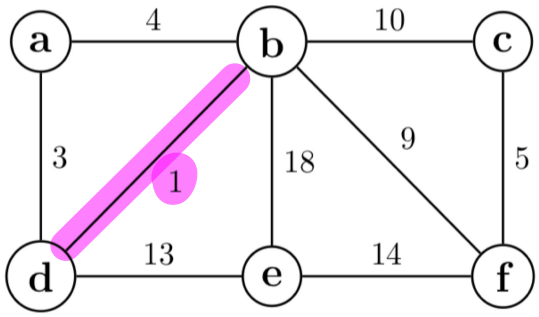
Add what you can and should!

↓ connects additional node/-s

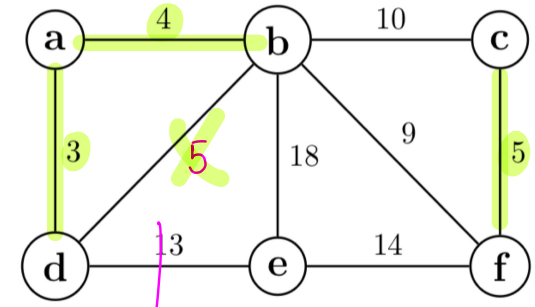
no circles!

i) Highlight the edges that are part of the minimum spanning tree. (Either in the picture above, or you can recreate the graph below).

ii) Write out all positive integers x such that if we replace the weight 1 of edge $\{b, d\}$ in the above graph with x, then edge $\{b, d\}$ would be in at least one minimum spanning tree of the resulting graph.



$x \in \{1, 2, 3, 4\}$



5 or bigger

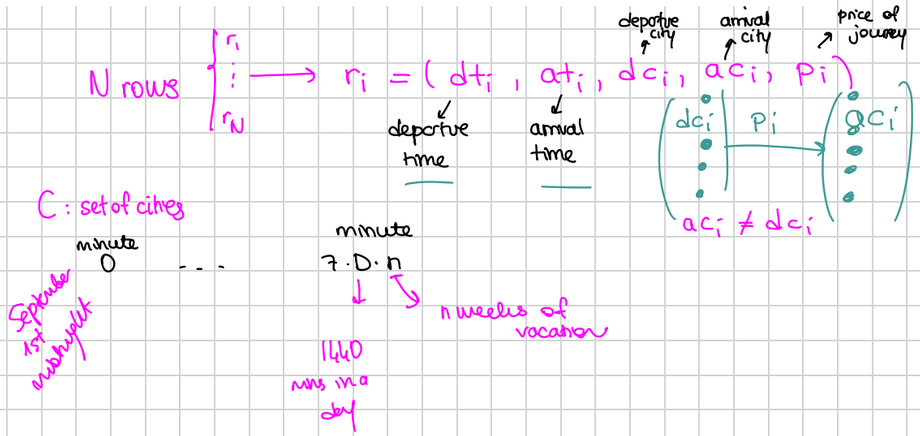
makes a circle

Graph Modelling

/ 7 P

a) Alice lives in Zurich and wants to visit her grandma in Stockholm. Since she has $n \in \mathbb{N}$ full weeks of vacation time from September 1st, she would like to use some of this time travel by train from Zurich to Stockholm and discover nice cities along the way. However, as a student, Alice may have a lot of time, but not a lot of money. Hence, while she does not mind spending long hours in the train, she would like to minimize her costs. You are tasked with helping her find the cost of the cheapest route from Zurich to Stockholm.

You are given a timetable composed of N rows $(r_1, \ldots, r_N)$, each representing one possible (direct) train journey. Let $\mathcal{C}$ be the set of cities. Each row r_i is of the form $r_i = (dt_i, at_i, dc_i, ac_i, p_i)$ where $dt_i \in \mathbb{N}$ is the departure time of the train, $at_i \in \mathbb{N}$ its arrival time, $cd_i \in \mathcal{C}$ its departure city, $ac_i \neq dc_i \in \mathcal{C}$ its arrival city, and $p_i \in \mathbb{N}$ the price of the journey in CHF. All departure and arrival times are expressed in minutes from September 1st, midnight. There are $D = 1440$ minutes in a day. You are guaranteed that the timetable allows Alice to travel from Zurich to Stockholm in less than n weeks, or $7 \cdot D \cdot n$ minutes. Note that as a Youth Hostel Premium Member, Alice does not need to pay anything to stay in the various cities she travels through.

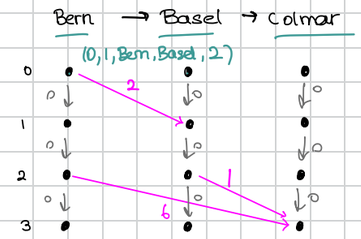


/ 7 P

b) Alice will likely use trains of a famous railway company on her way, and these trains are notoriously unreliable. In fact, each train can be cancelled arbitrarily. Being aware of this situation, Alice wants to plan for the worst and know how much more expensive the journey can become if trains are cancelled.

Alice has now selected her preferred journey. She will visit cities $c_1 = \text{Zurich}, c_2, \ldots, c_{T+1} = \text{Stockholm}$ in this order, with an expected total cost t. The timetable is the same as before, but now, even if trains are cancelled, Alice will always follow the same routes $c_1 \to c_2$, ..., $c_T \to c_{T+1}$.

For each of the T journeys, Alice buys the ticket at the departure station shortly before the train starts. When buying a ticket, Alice can be sure that her next train will work, but she never knows whether further connections will behave as expected. The only thing that the railway company **does** guarantee to Alice is that she will be able to reach Stockholm from Zurich within n weeks eventually.



i) Model the problem as a graph problem, so that you can solve it using an algorithm from the lecture. Describe the set of vertices, the set of edges and the weights. How many vertices and edges does your graph have? What is the corresponding graph problem?

$G = (V, E, w)$ is a weighted, directed Graph.

$V = \{(c,t) \mid c \in C,\ 0 \le t \le 7 \cdot D \cdot n\}$ $\quad |V| = |C| \cdot (7Dn+1)$

$E = \{((dc_i, dt_i), (ac_i, at_i)) \mid 1 \le i \le N\}$
$\cup$
$\{((c,t),(c,t')) \mid c \in C,\ 0 \le t < t' \le 7 \cdot D \cdot n\}$ $\quad |E| = N + (7 \cdot D \cdot n) \cdot |C|$ edges

$w(((dc_i, dt_i), (ac_i, at_i))) = p_i \quad \forall 1 \le i \le N$ $\quad w(((c,t),(c,t+1))) = 0 \quad \forall c \in C,\ 0 \le t < 7 \cdot D \cdot n$

Graph Problem: Cost of the shortest path from $(\text{Zürich}, 0)$ to $(\text{Stockholm}, 7 \cdot D \cdot n)$

ii) Which algorithm from the lecture can you use to solve this graph problem? Justify why you can use this algorithm, and state its asymptotic running time (with Θ) in terms of the number of vertices $|V|$ and edges $|E|$ in the graph.

Dijkstra (weighted, nonnegative edge costs)
in $\Theta((|V| + |E|) \cdot \log |V|)$

i) Model the problem as a longest path problem in a directed acyclic graph. Describe the set of vertices, the set of edges and the weights. Prove that your directed graph is acyclic and explain why the longest path problem you propose provides the right solution to the problem.

$C' = (c_1, c_2, \ldots, c_{T+1})$ $\quad G' = (V', E', w)$ is a weighted, directed graph.

$V' = \{(c,t) \mid c \in C',\ 0 \le t \le 7 \cdot D \cdot n\}$ $\quad c'_j \to c'_{j+1}$

$E' = \{((dc_i, dt_i), (ac_i, at_i)) \mid 1 \le i \le N,\ \exists 1 \le j < T.\ dc_i = c_j \wedge ac_i = c_{j+1}\}$
$\cup$
$\{((c,t),(c,t')) \mid c \in C,\ 0 \le t < t' \le 7 \cdot D \cdot n\}$

w just like in a

Graph Problem: Longest path (cost of the) from $(\text{Zurich}, 0)$ to $(\text{Stockholm}, 7 \cdot D \cdot n)$ in G'

Worst scenario would be that only the most expensive journey remaining uncancelled!

ii) Design an efficient algorithm that determines how much more expensive Alice's journey can become. In order to get full points, your algorithm should have $O(|V||E|)$ runtime.

Hint: What happens if you replace every edge with cost w by an edge with cost $-w$? Recall that, as proven in Task 2, a longest path always exists.

- Compute $G'' = (V, E, w')$ where $w'(e) = -w(e)$ for all $e \in E$
- Bellman-Ford starting from $(\text{Zurich}, 0)$ (to obtain $d_{G''}((\text{Zurich},0), (\text{Stockholm}, 7 \cdot D \cdot n))$
- Return $-d_{G''}((\text{Zurich},0), (\text{Stockholm}, 7 \cdot D \cdot n))$ $\quad -d_{G''} - t$

Bellman-Ford runtime: $O(|V| \cdot |E|)$

(No Dijkstra : G'' has negative weights, but contains no cycles
Bellman Ford without cycle detection is enough)