Instructions for the exam Algorithmen & Datenstrukturen

Conduct

- **Duration and weight:** The exam takes 4 hours. During this time, you should solve both the theoretical and the programming tasks. Both parts are designed for 2 hours each and are weighted equally. Nevertheless you can spend more or less time per part, if you like. We highly recommend you to start with the programming part and to continue with the theoretical part after no more than 2 hours.
- **Early hand in:** Due to the structure of this exam, it is not possible to hand in early.

Exam conditions

- **Disturbance:** Report any circumstances that disturb you during the exam **immediately**.
- **Disciplinary Code:** Dishonest behaviour can entail disciplinary measures according to the Disciplinary Code of ETH Zurich.
- **Clarifications:** During the exam, any questions regarding the content of the exam are only answered through the Judge (see point 4 of the technical guide). This also applies to the theoretical tasks.

Logistics

- **Allowed material:** Besides pens, dictionaries and a simple watch no extra material is allowed (no devices that are capable of communication, programmable or used to save data). Before the start of the exam, all forbidden items have to be put away. All electronic devices, especially mobile phones, smart watches and password flash drives, have to go to your luggage.
- **Luggage and jackets** have to be stored on the side of the room and can not go to your desk. We cannot guard your luggage, so you store it there at your own risk. In order to reduce that risk we advise you to not bring any valuables to the exam. Put a nametag onto your bag to prevent a mix-up.
- **Legi-check:** Put your legi (ETH-card) clearly visible on your desk. At the check after the start of the exam, we will also ask you to open the submissions page on the judge. Therefore please perform the steps of the technical guide shortly after the beginning of the exam.
- **Food and drinks** at your desk in a reasonable fashion are allowed. So you may bring a small amount of food and drinks, provided that (1) they do not disturb others (smell/sound), (2) dry food only (nothing liquid, greasy, etc.), (3) beverages only in closed and resealable containers (that you close whenever you don't drink), and (4) you are very careful and clean things up afterwards.
- **Earplugs:** Only Ohropax (or similar earplugs) are allowed, no earmuffs, no headphones.
- **Restroom:** If you would like to visit a restroom during the exam, please raise your hand. An assistant will accompany you. In the Schleusenraum it is **not** possible to use the restroom. We recommend to go to the toilet ahead of the Schleuse if you are assigned to the second run.

Programming part (computer exam)

- **nethz password:** To log in to the judge, you need your nethz account. Please ensure that you can remember your login credentials.
- **Best submission counts:** Just like with the programming tasks during the semester, you can submit to each programming task as often as you like. The best submission counts per task. Only what you submit to the judge counts.
- **Do not turn it off!** You may not turn off your computer for any reason, also not near the end of the exam. You risk loosing all of your files!
- **System issues:** If any system error occurs immediately raise your hand and do not click on any error messages.
- **Java libraries:** Formally, you can only use the objects, functions, etc. from `java.lang.*` and not from other packages. The package `java.lang` contains all the objects available by default, for example Math, Integer, String, System, Double, Boolean, etc., and you can use these freely (e.g. `Math.max()` or `Integer.MAX_VALUE` are fine).

  On the other hand, importing or using other packages is forbidden, e.g. you may not use `java.util.*`, `java.io.*` etc. Note this also forbids the Java collections. The module `java.util.Scanner` used to read the input in the provided templates is the only allowed exception.

  Also note that this restriction is not checked by the judge on submission, but will be checked after the exam and penalized.

  This is not to make your task any harder - on the contrary (since it means that you do not need to know all of the forbidden things). It is just to limit the solutions to only the things that you saw and used throughout the course.

Theoretical part (paper exam)

- Please write your student number on **every** sheet.
- Use a new sheet for every problem or write your solutions directly onto the task sheet (especially for task T1).
- You are not allowed to use your own paper. We will provide enough paper for you.
- Please write **legibly** with **blue or black**, non-erasable ink. We will only grade what we can read. Pencils are not allowed.
- You may only give one solution for each problem. Invalid attempts need to be clearly crossed out. Formulate your solutions comprehensibly.
- If you use algorithms and notation other than that of the lecture, you need to **briefly** explain them in such a way that the results can be understood and checked.
- At the end of the exam, put all sheets except for the one with your nametag into the envelope and seal it.

Programming Task P1. / 20 P

Enrolment Key: `blumenwiese`

Submission: see Section 3 of the Technical Guide

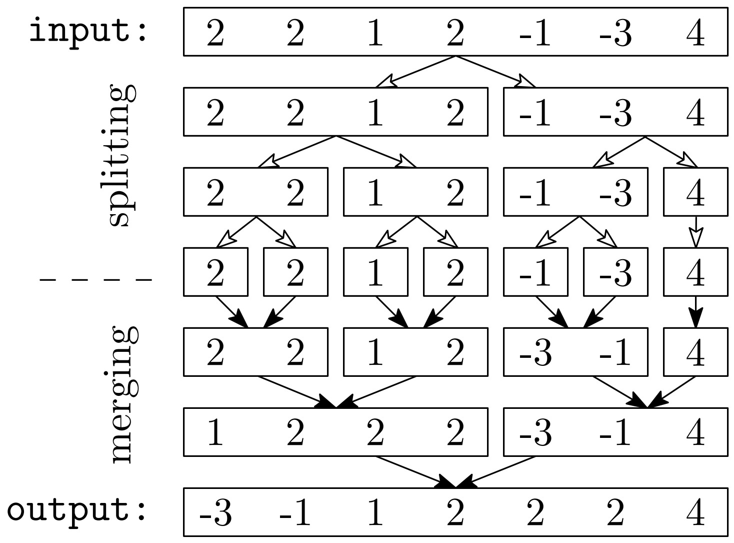
Implementing Mergesort

Your task is to implement the merging of two sorted subsequences in the mergesort algorithm. Most of the implementation of the mergesort algorithm is already provided by the template (reading the input, recursive calls, output). Your task is to complete the code of the function `mergeSort()`: The recursive calls to `mergeSort()` return two sorted arrays L and R, and your task is to merge these two arrays L and R into a sorted array B which is then returned. See the template for details.

To solve the task it is sufficient to replace the `TODO` comment in the template with your Java code. We strongly recommend not to change the rest of the template (even though it is not forbidden). Note that the `mergeSort()` function outputs some information that lets the judge verify that mergesort is running as intended and this output needs to stay exactly the same.

The numbers being sorted are integers between $-1\,000\,000$ and $1\,000\,000$. Numbers may occur more than once, and you need to preserve all the occurrences. The array to be sorted may contain up to $50\,000$ elements.

Example The following diagram illustrates mergesort on the input `2 2 1 2 -1 -3 4`.



Grading You may get up to 100 judge points. The program should implement every merge operation of k elements in time $\mathcal{O}(k)$. Submit only your `Main.java`.

Instructions For this exercise, we provide a program template as an Eclipse project in your workspace, and the template already implements most of the functionality. Your task is to complete the code of the function `mergeSort()`.

As usual, the project also contains data for your local testing and a `Judge.java` program that runs your `Main.java` on all the local tests – just open and run `Judge.java` in the project. The local test data are different than the data that are used in the online judge.

The input and output are handled by the template – you should not need the rest of this text.

Input The first line of the input contains only the number of test cases.

Each test consists of two lines. The first line contains n, the number of integers to sort, and the second line contains n integers to be sorted.

Output Upon every return from `mergeSort()` of at least 2 elements, the length of the array, the smallest and largest value are printed on a separate line, separated by spaces (calls to sort 1 element do not print anything.) After sorting the input of a test case, the sorted array is printed on a single line with numbers separated by spaces.

Example input (for the example above):

```
1
7
2 2 1 2 -1 -3 4
```

For the example output, see the file `testdata/example.out` in the project.

Space for your notes. These will not be graded. Only what was submitted to the judge counts for this exercise.

/ 20 P

Programming Task P2.

Enrolment Key: `blumenwiese`

Submission: see Section 3 of the Technical Guide

Cell Towers

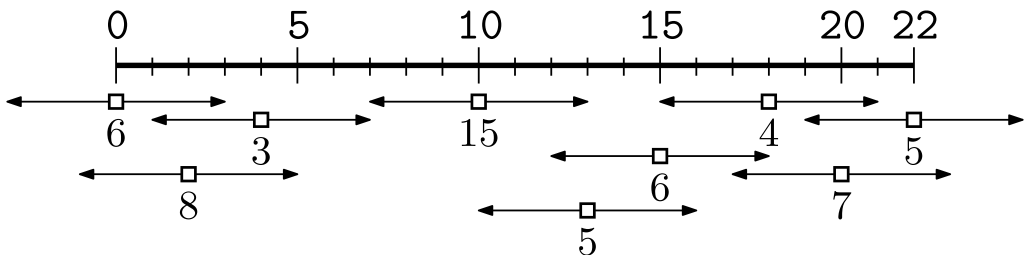
A brand-new highway was built across the country, and the drivers already complain about poor cell phone signal on the road. Your task is to build new cell towers to provide full coverage of the road, and to do that at the smallest cost.

The road is perfectly straight, L kilometers long, starting at km 0 and ending at km L, and initially has no cell towers. Every built cell tower covers R kilometers of the road in each of the two directions. A list of n possible tower locations and their costs is provided. The i-th possible tower location is d_i kilometers from the start of the road and costs c_i to build, $i = 0, \ldots, n-1$. The possible tower locations are ordered such that $0 \le d_0 < d_1 < d_2 < \cdots < d_{n-1} \le L$. All these locations d_i are different.

The inputs satisfy $1 \le L \le 1\,000\,000$, $1 \le R \le 1000$, $1 \le n \le 200\,000$, $1 \le c_i \le 1000$. The sum of all the costs does fit into a variable of type `int`. We guarantee that covering the entire road with a signal is possible.

Note that approximate (almost-optimal) solutions and "greedy algorithms" will likely *not* get any points – only the optimal solution is accepted. We recommend to design a 1-dimensional dynamic program.

Example The following picture shows an example with $L = 22$, $R = 3$ and $n = 9$, the squares are possible tower locations on the road, the numbers below them are their costs and the arrows indicate their covering range. An optimal solution has cost 37 (using towers with costs $6 + 3 + 15 + 6 + 7$). The input data may be found below.



Grading You may get up to 100 judge points. To get full points, your program should run in time $\mathcal{O}(nR)$ but slower solutions may get partial points. Submit only your `Main.java`.

Instructions For this exercise, we provide a program template as an Eclipse project in your workspace that helps you reading the inputs and writing the output.

As usual, the project also contains data for your local testing and a `Judge.java` program that runs your `Main.java` on all the local tests – just open and run `Judge.java` in the project. The local test data are different and generally smaller than the data that are used in the online judge.

The input and output are handled by the template – you should not need the rest of this text.

Input The first line of the input contains only the number of test cases.

The first line of each test case contains integers L, R and n, separated by spaces. The second line of each test case contains the possible locations of the towers as n integers separated by spaces. The third line of each test case contains the cost of the n locations as integers separated by spaces.

Output Output the cost of an optimal tower placement as an integer for each test case on a separate line.

Example input (for the example above):

```
1
22 3 9
0 2 4 10 13 15 18 20 22
6 8 3 15 5 6 4 7 5
```

Example output:

```
37
```

Space for your notes. These will not be graded. Only what was submitted to the judge counts for this exercise.

/ 16 P

Theory Task T1.

Notes:

1) In this problem, you have to provide **solutions only**. You should write them directly on this sheet.
2) We assume letters to be ordered alphabetically and numbers to be ordered ascendingly, according to their values.

/ 1 P

a) Perform two iterations of *Insertion Sort* on the following array. The array has already been sorted by previous iterations of this algorithm up to the double bar.

3	15	20	32	19	5	25	18	21	17	16	45
1	2	3	4	5	6	7	8	9	10	11	12

1	2	3	4	5	6	7	8	9	10	11	12

1	2	3	4	5	6	7	8	9	10	11	12

/ 1 P

b) Provide the number of necessary key comparisons when the elements T, S, I and L are accessed (in this order) and the list is reorganized using the Move-to-Front rule:

$$S \to I \to L \to E \to N \to T$$

Number of key comparisons: ____________________

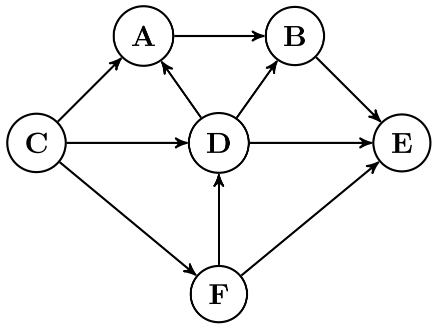
/ 1 P

c) Insert the keys $17, 19, 4, 26$ in this order into the hash table below. Use double hashing with the hash function $h(k) = k \bmod 11$, and use $h'(k) = 1 + (k \bmod 9)$ for probing (*to the left*).

				15		6	18			21
0	1	2	3	4	5	6	7	8	9	10

/ 1 P

d) Provide a topological sort of the graph below.



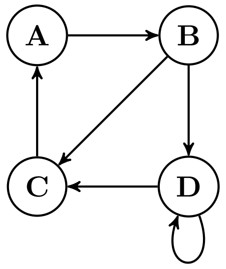
Topological sort:

_____, _____, _____, _____, _____, _____.

/ 1 P

e) Provide for the following graph G its adjacency matrix. Compute its reflexive, transitive closure and provide the corresponding adjacency matrix.

Graph G:



Adjacency matrix of G:

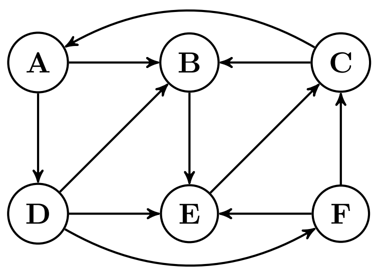
	A	B	C	D
A				
B				
C				
D				

Adjacency matrix of the reflexive and transitive closure of G:

	A	B	C	D
A				
B				
C				
D				

/ 1 P

f) It can happen that a depth-first search visits the vertices of a graph in the same order as a breadth-first search. In the following graph, mark all starting vertices for which this is the case, under the assumption that both traversals visit the neighboring vertices in alphabetically increasing order.



/ 1 P

g) Draw the binary search tree with the postorder traversal $1, 4, 3, 10, 7, 6$.

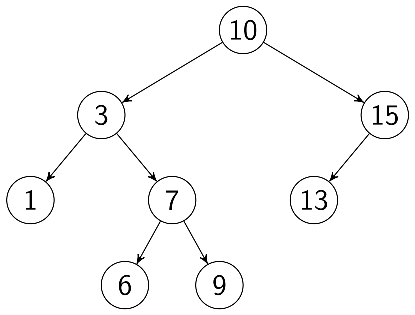
/ 2 P

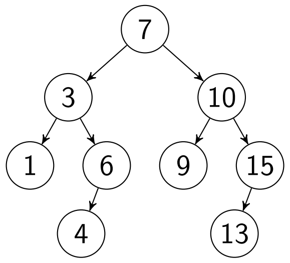
h) For each of the following statements, mark with a cross whether it is true or false. Every correct answer gives 0.5 points, for every wrong answer 0.5 points are removed. A missing answer gives 0 points. Overall the exercise gives at least 0 points. You don't have to justify your answer.

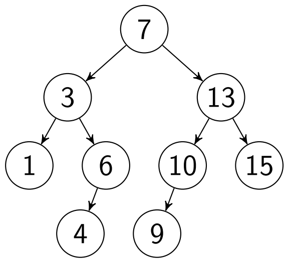
Heapsort is in-situ and stable.	☐ True	☐ False
In a Min-Heap the largest element is stored in a node without successors.	☐ True	☐ False
The height of a natural binary search tree is in $\mathcal{O}(\log n)$ where n denotes the number of keys stored in the tree.	☐ True	☐ False
A directed graph is acyclic if and only if it can be sorted topologogically.	☐ True	☐ False

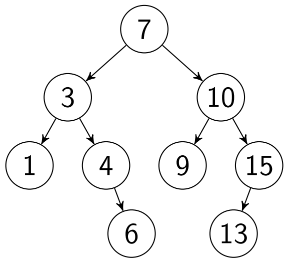
/ 1 P

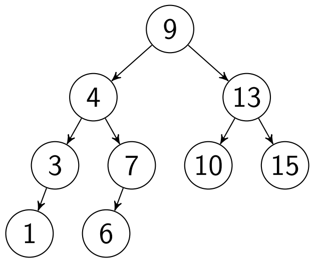
Which AVL tree results from inserting the key 4 into the following AVL tree and reestablishing the AVL condition afterwards?

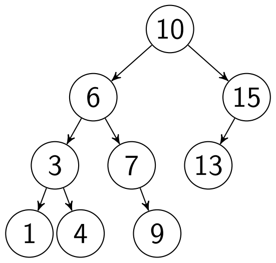


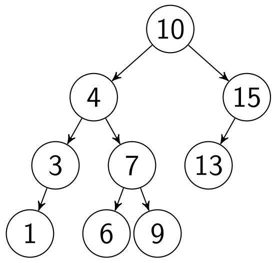
☐


☐


☐


☐


☐


☐


/ 3 P

i) Consider the following recursive formula:

$$T(n) := \begin{cases} 5 \cdot T(n/7) + 8 & n > 1 \\ 3 & n = 1 \end{cases}$$

Specify a closed (non-recursive) form for $T(n)$ that is *as simple as possible*, and prove its correctness using mathematical induction. You may assume that n is a power of 7. Hence, use $n = 7^k$ or $k = \log_7(n)$.

Notice: For $q \neq 1$, we have $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

Derivation (if required):

Closed and simplified form:

$T(n) = T(7^k) =$

Proof by induction:

Proof by induction (continuation):

/ 1 P

j) Specify for the following code fragment in Θ notation (as concisely as possible) how often the function f is asymptotically invoked in dependency of $n \in \mathbb{N}$. The function f does not invoke itself. You do not need to justify your answer.

```
for(int i = 1; i <= 2*n; i = i+5 ) {
        for(int j = 1; j*j <= n; j = j+1 )
                f();
        for(int k = 1; k*k < 1000; k = k+1 )
                f();
}
```

Number of invocations as concisely as possible in Θ notation:

/ 1 P

k) Specify for the following code fragment in Θ notation (as concisely as possible) how often the function f is asymptotically invoked in dependency of $n \in \mathbb{N}$. The function f does not invoke itself. You do not need to justify your answer.

```
for ( int i = 1; i < n; i = 2*i ) {
        for ( int j = n; j > 1; j = j/2 )
                f();
}
```

Number of invocations as concisely as possible in Θ notation:

/ 1 P

l) Specify an **order** for the functions below such that the following holds: If the function f is left of the function g, then $f \in \mathcal{O}(g)$.

Example: The three functions n^3, n^7, n^9 are already in a correct order, since $n^3 \in \mathcal{O}(n^7)$ and $n^7 \in \mathcal{O}(n^9)$.

$$\binom{n}{3},\ n!,\ 10^8,\ n^{\frac{3}{2}},\ \sqrt{n}\log n,\ \frac{n}{\log^3(n)},\ 2^n,\ 3^{\frac{n}{2}},\ \log(n^6)$$

Solution: ______, ______, ______, ______, ______, ______, ______, ______, ______.

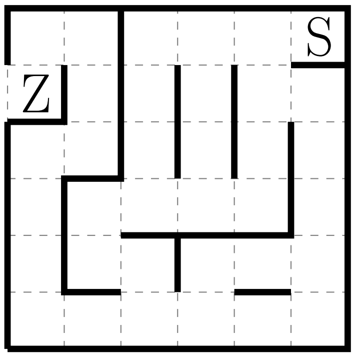
Theory Task T2. / 12 P

King Minos instructs Daedalus to construct a maze to imprison the Minotaur. Daedalus presents his maze with n fields and a given starting field as a drawing on gridded paper. In the following figure you can see an example maze with $n = 36$ fields. The starting field is indicated by an S, and the target field at the exit with a Z. We want to determine how fast the Minotaur can escape from the given maze.

/ 4 P

a) Model this problem as a shortest path problem:

- Describe how the maze can be represented as a graph such that the following is true: The number of vertices on a shortest path between two vertices representing the starting and target field corresponds exactly to the smallest number of fields that have to be visited for reaching the target field Z.
- Indicate how many vertices and edges your graph has in dependency of n.
- Name an algorithm of the lecture that solves the shortest path problem for this graph as efficiently as possible. Also, provide the running time as concisely as possible in Θ notation.



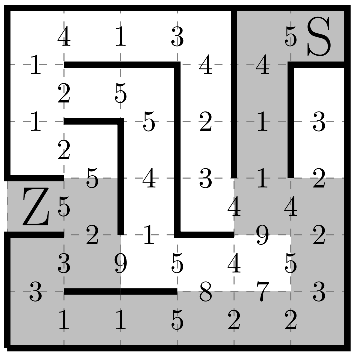
Example: In the example on the right the Minotaur has to visit at least 21 fields (including the starting and the target field) to escape.

/ 4 P

b) Various obstacles exist to complicate the escape. The time required to move from one field to another changes from obstacle to obstacle. For two adjacent fields that are not separated by a wall you are given the time that it takes to move from one field to another.

How can the modeling from task a) be adapted to compute, under consideration of the given times, a fastest route to escape from the maze?

- Describe how the maze can be represented as a graph such that the following is true: The length of a shortest path between two vertices representing the starting and target field corresponds exactly to the minimum time necessary for reaching the target field Z.
- Indicate how many vertices and edges your graph has in dependency of n.
- Name an algorithm of the lecture that solves the shortest path problem for this graph as efficiently as possible. Also, provide the running time as concisely as possible in Θ notation.

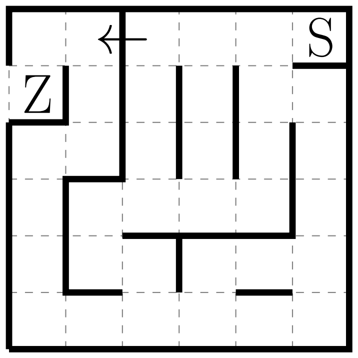


Example: In the example on the right one needs at least 44 time units to escape. The fastest route is indicated in gray.

/ 4 P

c) The Minotaur has the force to destroy exactly one inner wall of the maze (i.e. a wall for which both adjacent fields are inside the maze).

Compute how many fields the Minotaur has to visit on his escape at least, by modeling the problem as a shortest path problem. Name an algorithm that solves this problem as efficiently as possible. Also, provide the running time as concisely as possible in Θ notation.
Example: In the example on the right the Minotaur can destroy the wall that is marked with an arrow, and has to visit only 7 fields to escape (and not 21 as in task a)).



Subtask a)

- Definition of the graph (if possible, in words and not formal):

- Number of vertices and edges (as concisely as possible in Θ notation):

- Shortest path algorithm, as efficient as possible:

- Running time (as concisely as possible in Θ notation):

Subtask b)

- Definition of the graph (if possible, in words and not formal):

- Number of vertices and edges (as concisely as possible in Θ notation):

- Shortest path algorithm, as efficient as possible:

- Running time (as concisely as possible in Θ notation):

Subtask c)

- Modeling as a shortest path problem:

- Shortest path algorithm, as efficient as possible:

- Running time (as concisely as possible in Θ notation):

/ 12 P

Theory Task T3.

The longest baguette of the world has a length of l centimeters. We want to sell it as expensive as possible. For this purpose it may be cut into pieces. Each piece of length $l_i \in \mathbb{N}$ can be sold to a price $p_i \in \mathbb{N}$, for $i \in \{1, \ldots, n\}$. We want to determine, how the baguette can be cut into pieces such that the overall selling price is maximized. Of course, the sum of the lengths of all pieces may not exceed the length of the baguette.

Example: You are given a baguette of length $l = 121$ cm, and we allow $n = 3$ different piece lengths, namely $l_1 = 50$ cm (with price $p_1 = 25$), $l_2 = 26$ cm (with price $p_2 = 14$) and $l_3 = 20$ cm (with price $p_3 = 10$). Then, ideally we sell three pieces of length 26 cm and two with length 20 cm, for an overall price of $3 \cdot 14 + 2 \cdot 10 = 62$.

/ 7 P

a) Give a dynamic programming algorithm that obtains as input l and n as well as l_i and p_i for every $i \in \{1, \ldots, n\}$, and that computes the maximum overall selling price. Hence, for the example above the number 62 has to be calculated. Explain the following aspects in your solution.

   1) What is the meaning of a table entry, and which size does the DP table have?
   2) How can the table be initialized, and how can an entry be computed from the values of other entries?
   3) In which order can the entries be computed?
   4) How can the final solution be extracted once the table has been filled?

/ 3 P

b) Describe how you can determine which partition leads to the maximum overall selling price. For the example above a best partition consists of three pieces of length 26 cm with price 14 each and of another two pieces of length 20 cm with price 10 each.

/ 2 P

c) Provide the asymptotic running times of your algorithms for the subtasks a) and b), and justify your answer.

Subtask a)

Size of the DP table / Number of entries: ____________________

Meaning of a table entry:

$DP[\_\_\_\_\_]$: ____

____

Computation of an entry:

Computation order:

Computation of the maximum overall selling price:

Subtask b)

Determining the optimal partition:

Subtask c)

Running times (as concisely as possible in Θ notation) with justification:

Extra space. Please indicate clearly to which task your notes belong. Please cross out all notes that you don't want to be graded.