Task 1: Algorithmic Design

River loves fishing but unfortunately cannot swim. She lives in Norway and her favourite fishing spot is a fjord whose shore forms a perfect circle. Due the cold temperatures the fjord for fishing is mostly frozen with ice. There is an opening however in the center of the fjord that has the shape of a disk. The main goal is for River to use this opening for fishing.

River estimates the distance between the shore and the perimeter of the opening to be $D = 50$ meters. This is too far for her to throw the fishing line into the opening. She has to get as close as possible to the opening to determine if she can throw the fishing line into the opening.

For the sake of simplicity we assume that River can only do 1 meter steps moving from the shore towards the opening. However, if River steps on thin ice it will break. To determine how far she can go towards the opening, River will use a rock to determine if it is safe to step on the ice in front of her.

We make the following assumptions:

- River can throw a rock as far as necessary (although there is no benefit throwing the rock into the opening or even further).
- River can throw the rock in 1 meter increments, for example, 17 meters or 38 meters, but not at fractional values in-between such as 6.83 meters.
- If River throws a rock and the ice does not break, the ice in-between River's current position and the position where the rock landed does not break either (the ice gets thinner towards the opening).
- If the rock breaks the ice, it falls into the fjord and cannot be reused. If the rock does not break the ice, River can walk towards the location where the rock landed and can throw the same rock again.
- For simplicity, you may assume that the fjord has the shape of a disk, its shore is a circle, and River only throws rocks (and walks) radially towards the fjord's center.

For all subsequent tasks you may assume that there is a function called that takes an integer value (a distance) and returns if the ice breaks if a stone is thrown at this distance and otherwise.

Part A.

Assume that River finds a single rock nearby and uses that rock to determine how close she can move to the opening. Assume that the initial distance between River's current location and the opening is D meters.

Write an algorithm using pseudocode (as we have done in the lectures) that returns the largest

distance $d_{max} \leq D$ River can walk safely towards the opening using the smallest number of throws. Your algorithm has to take D as an argument and returns d_{max}.

Part B.

After a bit more searching, River now finds two rocks nearby and will throw the rocks to determine how close she can get to the opening.

Write an algorithm using pseudocode that returns the largest distance $d_{max} \leq D$ River can walk safely towards the opening with the smallest number of throws using two rocks (River gets quickly exhausted as the stones are heavy). Your algorithm has to take the initial value D has an input argument.

Let $T(2, D)$ denote the (minimum) number of throws necessary to determine d_{max} given 2 rocks and the initial distance $D \in \mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers. When computing the largest distance d_{max} your algorithm must use a total number of throws that grows asymptotically slower than $O(D)$ in the worst case. More formally: $\lim_{D \to \infty} \frac{T(2,D)}{D} = 0$. We call T *sublinear* in D.

Hint: since the initial distance between River and the opening is $D = 50$ meters, it is possible to compute the largest distance River can walk towards the opening with at most 10 throws but any solution that is sublinear in D is eligible for full marks.

Part C.

In this part you need to analyze the efficiency of your algorithm developed in Part B. Given a distance D determine the number of throws $T(2, D)$ your algorithm requires using Big O notation.

Part D.

Assume there is a large pile of rocks and River can throw as many rocks as needed to determine the largest distance River can walk towards the opening. Is it possible to design an algorithm that requires less throws than your algorithm developed in Part B to compute d_{max} given D? Either design such an algorithm using pseudocode or show why it is not possible.

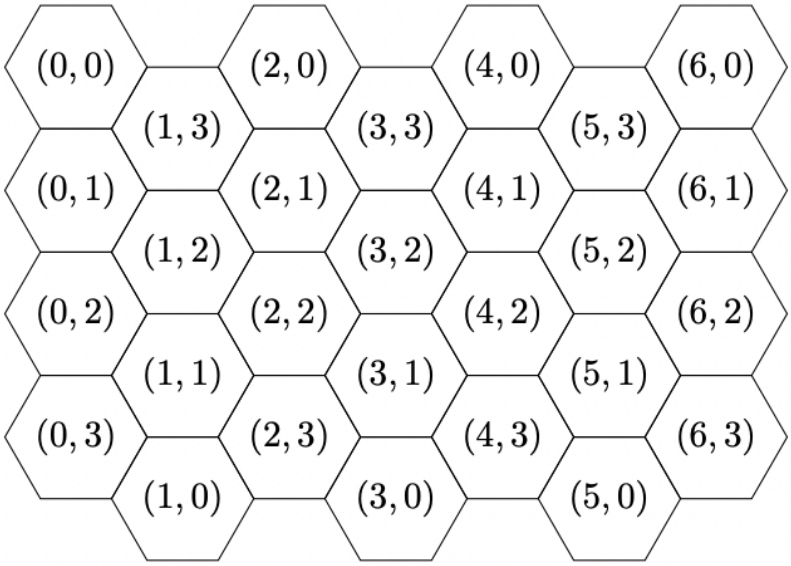
Part E.

Only attempt this part if you have solved all other tasks of Assignment 1. If River had 3 rocks, what is the minimum number of throws $T(3, D)$ necessary to determine the closest distance towards the opening given? Explain your answer.

Note: You can get full marks if you solve Part A to D. You will get an additional bonus mark if you can solve Part E.

Task 2: C Problem

Olivia's grandfather sent her a letter describing the great lost treasures he discovered decades ago while exploring the Mystic Archipelago. To obscure his findings, he stored the details of his treasure map in a hexagonal structure like the one below.



The points in a 7×4 array correspond to the points shown above. The same behaviour of odd and even columns in a different order generalises to an $m \times n$ map.

Part A.

Write a function in C,

```
struct point *getAdjacentPoints(struct map *m, struct point *p)
```

that returns all the adjacent points around an inputted point which would lie on the map.

- These should be sorted first by the first coordinate, and then the second coordinate.
- If the inputted point is not on the map, the point is considered to have no adjacent points.

Part B.

After decades of secrecy, Olivia's grandfather's letter revealed what each number corresponds to in his map.

- If the value of the point is negative, it represents the depth of the ocean floor in metres.
- If the value of the point is 0, it represents land without treasure.
- If the value of the point is 100, it represents an airport.
- If the value of the point is positive but not 100, it represents the value of that specific treasure.

Each bit of treasure corresponds to an artefact. Artefacts that belong to the same island are related, and so if you find multiple artefacts on the same island, they are worth much more than the sum of their parts. The total value of artefacts on a specific island is given by this formula

$$f(x_1, x_2, \ldots, x_n) = n \times x_1 \times x_2 \times \cdots \times x_n$$

This means that if there is a treasure worth 20 on island 1, and island 2 has treasures worth 2, 3, 4, then the total value that can be found is

$$20 + \left[3 \times 2 \times 3 \times 4\right] = 92$$

This is because artefacts from different islands are unrelated and so you can only add the sum of the value of islands.

Write a function

```
int mapValue(struct map *m)
```

That finds the total value of all the treasure found in a map. The map has S, L, T points of sea, land and treasure respectively, with an airport counting as land.

Describe the time complexity of the algorithm in terms of S, L, T.

Part C.

Olivia further finds out that her grandfather stored the locations of each treasure in a separate array. Olivia then uses this information to make a better algorithm to find the total value of a map. For islands with treasure, N_T represents the expected value of the number of treasure points on the island. The expected value of the size of islands with treasure on the map is I_S. The values of N_T and I_S are constants that represent the expected values taken over every single map. Specific maps do not necessarily follow these constants.

(i) Briefly describe how the new algorithm improves upon the above one. You may but do not have to use pseudocode.

(ii) Find the best case time complexity of the new algorithm.

(iii) Find the worst case time complexity of the new algorithm.

(iv) Explain a reasonable definition to use for the average case of the new algorithm and find its time complexity.

(v) Give an example of a map with at least 1 treasure point, 1 land point and 1 sea point where your algorithm in this question explores as many points as the algorithm you created in the previous question. Briefly explain your answer.

Give each of the time complexities in terms of S, L, T, N_T, I_S when appropriate. When calculating the time complexities, you are seeing how the time complexity grows with respect to values S, L, T.

Each answer only requires 1-2 sentences of brief justification.

Part D.

Olivia discovers that her grandfather was looking for the location of a mysterious treasure called The Star of Alhambra. This is far more valuable than any of the previous treasures he mapped out and he narrowed down the number of locations it could be. Olivia wants to be able to go from one place to another as quickly as possible to check as many of these locations as she can. The amount of time it takes to move to an adjacent point is determined by the current point you are on.

- It takes 5 minutes to move from a land point to an adjacent point.
- It takes $2 + \text{ceil}(\frac{\text{depth}^2}{1000})$ time to move from a point in the sea.
- It takes $\max(15, |x_1 - x_2|^2 - 85)$ minutes to fly from one airport to any other airport. where x_1, x_2 are the x coordinates of the airports.
- It takes 5 minutes to move from an airport or treasure to an adjacent point (treat like land).

Write a function

```
int minTime(struct map *m, struct point *start, struct point *end)
```

that finds the minimum amount of time from the start point to the end point.

If A is the number of airports in the map, find the worst case time complexity in terms of S, L, T, A. Briefly explain your algorithm and data structures used to justify your time complexity.

Part E.

Olivia finds some peculiar maps where any ocean has a depth of exactly $-50m$. She strongly suspects that these are the ones to contain The Star of Alhambra. Out of all these maps, she noticed that there were never more than 5 airports. All of these have a known location. Write a function:

```
int minTimeDry(struct map *m, struct point *start, struct point *end, struct point *airports, int n
```

which finds the minimum distance from one point to another faster than the previous algorithm.

What is the worst case time complexity of this algorithm? To get full credit you must find the algorithm with the best worst case time complexity.

> TIP: If you think you've worked out the most efficient algorithm, but can't work out how to actually write it, check out the `printMap` function.