assignment 1

▼ Task 1

▼ Question 1A

```
#define MAX_DISTANCE 50

function find_d_max(D)
  nxt_throw ← MAX_DISTANCE - D + 1;
  while nxt_throw <= DISTANCE do
    if Throw(nxt_throw) then
      nxt_throw ← nxt_throw + 1
    else
      return nxt_throw - 1

  return MAX_DISTANCE
```

▼ Question 1B

```
#define MAX_DISTANCE 50

function find_d_max_optimized(D)
  nxt_throw ← MAX_DISTANCE - D + 1;
  interval ← sqrt(MAX_DISTANCE)
  d_max ← nxt_throw

  for d_max to MAX_DISTANCE step interval do
    if not Throw(d_max) break

  for i ← d_max - interval + 1 to MAX_DISTANCE do
    if not Throw(i) return i - 1;

  return MAX_DISTANCE
```

- To get at most 10 throws for D=50, use the triangular numbers backwards. First rock: [10, 19, 27, 34, 40, 45, 49], then use the second rock to step.

▼ Question 1C

Throwing the first rock using larger intervals (sqrt of `MAX_DISTANCE`) to discover a rough estimate of where the ice starts to break.

Once this is discovered, the second rock is carefully thrown meter by meter within the interval to find the target.

Completing both parts will require `2 * sqrt(MAX_DISTANCE)` operations in the worst case scenario, leading to a time complexity of **`O(sqrt(n))`**.

▼ Question 1D

- Yes as the optimization is based on divide and conquer. More rocks would allow more 'dividing', and hence improving the time efficiency.
- As this is a binary search, the throw complexity is **`O(logD)`** which is clearly faster than the algorithm in Part B.

```
function find_dmax_c(lower, upper)
  if lower + 1 == upper then
    if Throw(upper) then
      return upper
    else
      return lower

  mid ← (lower + upper) / 2

  if Throw(mid) then
    return find_dmax_c(mid, upper)
  else
    return find_dmax_c(lower, mid)
```

▼ Task 2

▼ Part B

Time complexity analysis by operation:

1. Looping over the entire map will require a complexity of `O(S + L + T)`.
2. DFS Algorithm: `O(L + T)`, since the recursion will stop once ocean is reached (the amount of ocean nodes explored by the DFS algorithm does not scale with input size, but with the structure of the island.).

Since `max(L + T) < height * width` , operation 1 and 2 will be the fastest growing times as input size scales. Answer: $O(S + L + T)$.

▼ Part C

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

Since treasure can only be on land, there is no need to iterate over the entire ocean, but only the edge. On a bigger scale, the amount of ocean nodes visited grows slower than land nodes, therefore `S` should not affect the time complexity.

The DFS can be initiated at the first treasure location on each island. All proceeding treasure locations on the same island can be skipped or removed from the array once the DFS reaches it.

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

In the best case scenario, there is only 1 island filled only with 1 treasure nodes and no empty land. Only treasure nodes need to be explored, the time complexity will be $O(T)$

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

In the worse case scenario, there is only 1 empty island with one treasure node. All nodes would need to be explored. The time complexity will be $O(L+T)$.

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

The time complexity of this algorithm is $O(L+T)$. On average, the complexity will be $O(N_t + I_s)$ since N_t is the average of treasure nodes and I_s is the average of land nodes.

(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.

```
Map: 1 0 -1
```

- Old Algorithim: **Treasure** → **Land** → **Ocean**. Total Nodes visited 3.
    - Once ocean is reached, the DFS finishes, and the algorithim will proceed to the of the Land and Ocean node, then terminate. These are **NOT** visited because are marked as visited in a seperate list.

- New Algorithim: **Treasure → Land → Ocean**. The algorithim will terminate here. Total Nodes visited: 3.

▼ **Part D**

If *A* is the number of airports in the map, find the worst case time complexity in terms of S, L, T, A.

- Let `N` represents the total of `S + L + T` nodes.

Time complexity analysis by operation:

1. Inserting all the elements into the pirority queue: $O(N + logN)$
2. Pulling out points, then adding airports by iterating the entire map. $O(NlogN)$.
3. Updating the pirority queue: $O(logN)$

Total: $O(N + NlogN)$

▼ **Part E**

What is the worst case time complexity of this algorithm?

Ocean with a depth of -50m is essentially just land since the time it takes to move from it is 5 minutes.

This means that the map will only contain land, and airports.

This means the amount of possible paths will be based off the amount of airports, and not the input size.

The map does not need to be iterated and the shortest path can be calculated. Let `D` be the distance between the start and the end nodes. In the worse case, all possible combinations of the airports paths will need to be calculated. $O(1 + A^A)$