CSE321 – Introduction to Algorithm Design
HW-4

Yusuf Arslan
200104004112

1. In this question, I used recursive solution to find longest path. In each step, either 'i' or 'j' incremented by 1 and each visited node added to a list. If we reach the end of the list (n, m) then, sum of the path compared with the previous path, if the current path is larger than the previous path, it replaced with previous path. By this method, I checked all possible paths and find the longest path.

   The time complexity of this function is `O(2^(n+m))`, where n and m are the dimensions of the map.

   This is because at each step, the function calls itself twice, once to move to the right and once to move down, creating two branches of the recursive tree. This means that the function will make 2^(n+m) recursive calls, when the path goes through every possible cell in the map.

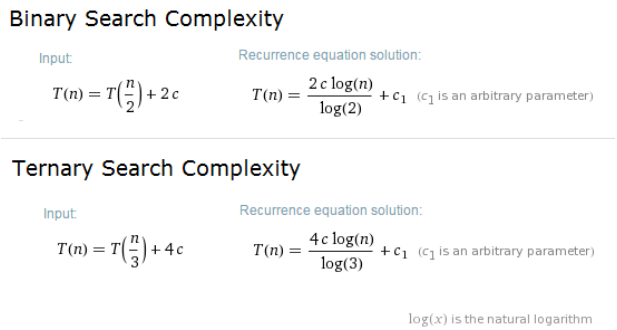
2. Quick select algorithm is used to find the kth smallest element in an array.
   The algorithm works by selecting a pivot element from the array and partitioning the array into two subarrays: a left subarray containing the elements that are smaller than the pivot, and a right subarray containing the elements that are larger than the pivot.
   If the pivot is the kth smallest element, it is returned. If the kth smallest element is in the left subarray, the algorithm is applied recursively to the left subarray. If it is in the right subarray, it is applied recursively to the right subarray, with k adjusted to account for the number of elements in the left subarray.
   The time complexity of Quick select is `O(n)` in the average case and `O(n^2)` in the worst case, where n is the number of elements in the array. This is because the pivot element is chosen randomly, and in the average case, it will split the array into roughly equal-sized left and right subarrays. In the worst case, the pivot element will either be the smallest or largest element in the array, leading to one subarray containing all the elements and the other being empty, resulting in a time complexity of `O(n^2)`.

   Median is the n/2'th smallest element of the list, so, to solve this problem I used quick select algorithm to find it. In average, it has `O(n)` time complexity.

3. a. I used a circular linked list to solve finding the winner of the game in this part of question. There is a loop inside the 'find_winner' function, and it iterates until one element remains in the list. It basically works in this way:
Each time the loop, next pointer of the current node points to one after its neighbor instead of pointing to its neighbor, and then the pointer is moved one forward.
By this solution, in each iteration size of list decreases by one, then the winner of the game founds if one element is remained. So, time complexity will be $O(n)$

   b. In this part of question, I used an array to represent the players. It recursively decreases size of players with eliminating players by dividing 2 the array of players. Since in each call, number of elements decreased by 2, time complexity of this solution is $O(\log n)$

4. While ternary search looks better compared to binary search, it's actually not like that. To understand this, we can take a look at the recurrence relations and solutions of both searches:



   When we divide the result from the binary search's solution by the ternary search's solution, we get this result:
   $2.\log_3 2$

   As it seen, it is larger than the 1.
   In conclusion, we can say although the tree's levels are reduced, ternary search is less efficient than binary search as the increase in the number of comparisons will be greater in ternary search.

   By the same logic, with an array size n, if we divide array into the n part, result will be find in one step, however, number of comparison will be 'n' and it is same as the linear search $O(n)$.

5. In interpolation search, position probing is calculation by this formula:

```
mid = Lo + ((Hi - Lo) / (A[Hi] - A[Lo])) * (X - A[Lo])
```

In uniformly distributed array, this formula will find very close position to the element we are searching.

a. In best case `mid` will be the index of the element that we are searching, and it is `O(1)`.

b. Binary search works in `O(logn)` even in the worst scenario, regardless of how the numbers are distributed. But in interpolation search, when items are distributed exponentially, the time complexity will be O(n) in worst case scenario. However, if elements are uniformly distributed, time complexity of interpolation search will be O(log(logn)) which is much faster than binary search.