GTU

**DEPARTMENT OF COMPUTER ENGINEERING**

**CSE 321 – Autumn 2022**

**HOMEWORK 4  
REPORT**

SÜLEYMAN GÖLBOL 1801042656

# 2D MAP GAME

For the time complexity of first question;  
I’ve created a class called Q1 with methods path\_generator() and find\_highest\_points(). First I’ve generated all the paths using iterative approach. Instead recursion, I used a stack to put the route point and values of cell. In while loop every step is constant so it took O(n) time complexity.  
Then to find highest points I used nested loops so complexity is O(n2).

1. **MEDIAN OF UNSORTED ARRAY**

For the complexity of median of an unsorted array, I created a class Q2. This class uses decrease-and-conquer with lomuto partition. To get partition index, lomuto\_partition() compares pivot with elements on left which takes O(n) time. Find\_kth\_smallest() function is a recursive function and calls itself when indexes are not satisfied.

So, for complexity; it uses O(n) and also there was O(n) from lomuto; at the end it takes O(n2) time.

**3) CIRCULAR PLAYER ELIMINATION**

**Part a:**

# As it wanted in the question; for part a, I made it linear time by keeping current element’s info in a variable so that every time I want to remove next element; I don’t need to iterate until that element. I have a function winner() and it has a while loop that continues until size of linked list is bigger than one; since complexity is O(n).

**Part b:**

# In this problem, I realized in the first step, all the players with even numbers are eliminating.

# I realized that every time n is power of 2; answer is 1. Because after 2^n, it starts with non-eliminated players again.

# So, for example; if n is 2^2 then winner is 1. If n is 2^2+1=5, then winner is 3 because first 2 and 4 eliminates. Then 5 eliminates 1 and 3 eliminates 5.

# So;

|  |  |
| --- | --- |
| *NUMBER OF PLAYERS* | *WINNER* |
| *2^a* | *1* |
| *2^a+1* | *3* |
| *2^a+2* | *5* |
| *.* | *.* |
| *.* | *.* |
| *2^b=n* | *Again, it’s 1.* |

# As it seen, winner id is increasing 2-by-2. This continues until the number 2^a = N again. In this time, winner is player 1 again.

# For decrease and conquer, after finding biggest 2x which is smaller than n; we can find the winner in log(n) time.

# For example:

# n = 6, biggest 2x = 22 = 4 (smaller than n=6).

# This means; If n was 4, winner would be 1.

# If n was 5, winner would be 3.

# N is 6, so winner is 5.

# -- After finding x -> winner = 2x+1.

# metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# While loop here takes log(n) complexity. log\_finder() function also takes log(n) complexity. So, time complexity is 2log(n) = log(n)

# 

**4) SEARCH COMPARISONS**

When we compare the complexities of these algorithms;

Ternary search may have bigger divisor so that means for a small array; it will be faster. But if compare them asymptotically (which means when the input sizes are getting bigger), log2n will be faster than log3n because time complexity of log3n has faster growing. (Faster growing means that time complexity will be bigger). With Log3n in every part, element which are searched will be bigger. So, dividing into n parts will make it O(n) which is not more efficient. Because we still need to search for n elements.

**5) INTERPOLATION SEARCH**

1. It happens when distribution of the keys are uniform. For example if element in the middle is the key that we are searching for, we found the element in the first step we dig so best case is Ω(1).
2. Interpolation sort is a changed version of binary search algorithm. The purpose of this job is to get the seek value in a less number of steps then binary search. It uses a mid-index finder formula to get the seek value. Also, if the key is closer to the element at the end; it may start from the back side for searching.

In binary search, we were using middle element and comparing with target key so that we make search space smaller.

For smaller arrays, interpolation search performs worse than binary search because it uses more computation for mid formula. But since growth rate is smaller than binary search; it will perform better than binary search.

For the time complexity, in the worst case it may perform O(n) if key numbers increase in an exponential way. Similar for binary search, if elements are added to binary tree unbalanced, we need to perform O(n). For the best case binary search performs Ω(1) when searched element in the middle, and for interpolation search it’s also Ω (1).

Only difference is average cases; because in average case of interpolation search it uses uniform distribution keys. But for binary search, average case is O(nlogn/(n+1))= O(logn).