GTU Department of Computer Engineering
CSE 321 - Fall 2022
Homework 4 Report

Emirkan Burak Yılmaz
1901042659

1. Maximum Number of Total Points

   For position A_iB_j there could be two possible movements. Next position could be A_iB_{j+1} by right movement or $A_{i+1}B_j$ by down movement.

   - Start from position A_1B_1 and try each movement recursively by considering the boundaries of the game map until A_nB_m is reached.
   - After applying different movements and finding the end position, we have two paths that are constructed with the selected movements. At this step select the path that contains more points.
   - Add the current position to the selected path and add the point at the position.
   - Return the selected path and the sum of points on this path.

   For n by m game board there are n – 1 down movement m – 1 right movement. Total number of paths can be found by

   $$((n – 1) * (m - 1))! / ((n - 1)! * (m – 1)!)$$

   In this way, the algorithm tries all the paths and selects the path that contains maximum point gain. This brute force solution has exponential time complexity.

2. Median of Unsorted Array

   The problem can be solved by applying partition(s) till finding pivot at the median position.

   Select:

   - Select a pivot p, out of array A.
   - Apply partition by splitting A into S1 and S2 where all elements in S1 are less than p and all elements in S2 are larger than p. After partition p is the place that is in sorted order.
   - If p is at i^{th} place, then return the position value p.
   - Else if the position of p is smaller than i then repeat the process recursively on S1 looking the i^{th} smallest element.
   - Else i^{th} smallest element somewhere in S2. Repeat the process recursively looking for the i^{th} smallest element.

   Median:

   - N is the length of the array. If N is odd, apply select algorithm for i = N / 2. If N is even, apply select algorithm two times for the middle two value and get the average of two value.

   In the best case, 1 partition is enough to find the median of array which takes O(n) time complexity. In the worst case, array is reverse ordered there needs to be n/2 partition for input size n, n-1, n-2 … n/2 which takes $O(n^2)$ time complexity. For the average case it takes O(nlogn).

3. Circular Elimination
    a. Brute Force Approach by Circular Linked List
        - First create the circular list from 1 to n.
        - Start from the 1st node and set its next field to the 3rd node. With this assignment the 2nd node is eliminated. Continue this elimination process till only one node remains.
        - The remainder is the winner

        Firstly, algorithm traverse n node to eliminate half of the players then n/2, n/4 … Total number of iterations is n + n/2 + n/4 … + 1. This resulted in a linear time complexity O(n).

    b. Decrease-and-Conquer Approach

        Let's say the number of players is n. Though those players are ordered from 1 to n. After each elimination process the number of players becomes half. The player at first position is becomes winner when n becomes 1 after the consecutive eliminations. So, observing the first position is enough to find the winner.

        - Initialize variable base2 as 2, initialize first as 1. First indicates the player in the first position.
        - If n is even, then every player at even position (2, 4, 6…) will be eliminated and the player at first position will not change. If it's odd, then the first player in addition every player at an even position eliminated and the new first player becomes winner + base2.
        - After the elimination process is done according to odd or even case, set n = n/2 and base2 = 2*base2. Apply this process till n becomes 1.
        - Return first as the winner of the game.

        In each Iteration half of the players are eliminated and the number of players decreases by half. If we say $n = 2^k + m$ ($0 <= m < 2^k$). There needs to be $k = \log_2 n$ iteration needs to eliminate all the players except the winner. So, the time complexity of this algorithm is O(logn).

4. Ternary Search vs Binary Search
    - In asymptotic analyze constants before the asymptotic classes are ignored. Since the effect of base of logarithm is constant, it can be ignored. However, ternary search seems better even if both have same asymptotic classes, since $O(\log_3 n) < O(\log_2 n)$. But to be make sure we need to analyze both algorithms.
    - Ternary search divides the array into 3 parts, and there should be at most 4 comparisons to decide the next step. On the other hand, binary search divides the array into 2 parts and there should be at most 2 comparisons to decide the next step. Basic operation is comparison and the recursive formula of the two algorithms can be derived as follows.

      $T(n) = T(n/2) + 2, T(1) = 1$ (Binary Search)

$T(n) = 2\log_2 n$

$T(n) = T(n/3) + 4, T(1) = 1$ (Ternary Search)
$T(n) = 4\log_3 n$

$2\log_2 n / 4\log_3 n = \log_2 3 / 2 = 1.58 / 2 = 0.79$

- As it can see $4\log_3 n$ is bigger than $2\log_2 n$. Which means even if $O(\log_3 n)$ $O(\log_2 n)$ seems better at the beginning, the constants that we ignore in asymptotic notation break the equality and resulted in worse performance for ternary search due to larger number of comparisons.

Dividing array into n parts requires 2n – 2 comparisons. If we divide the array has n elements, then the worst-case time complexity of the algorithm becomes O(n) which has same asymptotic class with linear search. However linear search algorithm would be better solution since required comparisons are less than the new algorithm.

5. Interpolation Search
    a. What is the best-case scenario of interpolation search? What is the time complexity of it?
        - In the best case, the target can be find just 1 probe making a constant time complexity O(1). In average case, algorithm time complexity becomes O(log(logn)). Result can be found analyzing the expected number of probes needed to reduce search space and the probability of a key to be less than or equal to y which can be found by

$$p = (y – arr[0]) / (arr[n - 1] – arr[0])$$

    b. What is the difference between interpolation search and binary search in terms of the manner of work and the time complexity?
        - Binary search chooses the middle element of the search space discarding half of the list depending on the comparison made at each iteration.
        - Interpolation search goes through different locations according to the search key. At each search step, the algorithm calculates the remaining search space where the target element might be based on the low and high values of the search space and target value. A comparison is made between the target and the value found at this position, if it is not equal then the search space is reduced to the part before or after the estimated position.

Pos(mid) = low + ((target – arr[low]) * (high – low) / (arr[high] – arr[low]))

        - The above formula is used in interpolation search for finding mid position. It returns a smaller value of pos(mid) when target is close to arr[low] and higher value pos(mid) when target is closer to arr[high].

- When we consider the time complexities of two algorithms interpolation search gives O(log(logn)) and binary search gives O(logn) in the average case. Since interpolation search reduced the search space more wisely, this resulted in a better performance compared to binary search.