Muhammad Mazharul Islam
ID - 23341053
Assignment - 01
Section - 10

Part 1: Iterative Time Complexity

Q1.

For the first part of the given snippet -

Outer loop time complexity — $O(n)$

Inner loop time complexity — $O(n)$

For the second part of the given snippet -

Outer loop time complexity — $O(\log_4 n)$

Inner loop time complexity — $O(n)$

Hence, time complexity of the given snippet = $O(n^2) + O(n \log n)$

$= O(n^2)$

Q2.

(a)

i) The loop evantually stops when $x >= n$. Meaning,

$x = 0+1+2+3+4+ \cdots\cdots +k >= n$

$\Rightarrow \frac{k(k+1)}{2} = n$

$\Rightarrow k = \sqrt{n}$

Hence, the loop iterates for k times.

So, time complexity = $O(\sqrt{n})$

ii) The innermost loop of last part never ends as k is initialized as 0. So, the loop runs indefinitely.

Hence, the time complexity of this code can't be determined.

(b)

Total students = n

As not sorted, time complexity = O(n)

(c)

Given Expression,

$\sqrt[4]{10n^4 + 9n^3 + 8n}$

In $10n^4 + 9n^3 + 8n$ the dominiting term is $10n^4$. Omitting the constant it's simply n^4. Hence, in Big-Oh notation it can be written as -

$$O\left(\sqrt[4]{n^4}\right) = O(n)$$

Lower bound of a linear class function can be the linear function itself. Hence, the lower bound is also $\Omega(n)$

As we can see for the given term

$$\Omega(n) = O(n)$$

Hence, we can conclude by saying,

$$\sqrt[4]{10n^4 + 9n^3 + 8n} = \Theta(n)$$

(Proved)

Part 2: Recursive Time Complexity

Q1.

$$T(n) = 2T\left(\frac{n}{2}\right) + \frac{1}{n}$$

$$= 2\times\left[2T\left(\frac{n}{2^2}\right) + \frac{2}{n}\right] + \frac{1}{n}$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + \frac{2^2}{n} + \frac{1}{n}$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + \frac{2^2}{n}\right] + \frac{2^2}{n} + \frac{1}{n}$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + \frac{2^4}{n} + \frac{2^2}{n} + \frac{1}{n}$$

$$\vdots$$

$$= 2^k T\left(\frac{n}{2^k}\right) + \frac{2^0}{n} + \frac{2^2}{n} + \frac{2^4}{n} + \cdots + \frac{2^{2(k-1)}}{n}$$

$\frac{n}{2^k} = 1$
$\Rightarrow n = 2^k$
$\Rightarrow \boxed{k = \log_2 n}$

$$= n + \frac{1}{n}\left[2^0 + 2^2 + 2^4 + \cdots + 2^{2(k-1)}\right]$$

$$= n + \frac{1}{n}\times\left[\frac{4^{\log_2 n} - 1}{4 - 1}\right]$$

$$= n + \frac{1}{3n}\times(n^2 - 1)$$

$$= n + \frac{n}{3} - \frac{1}{3n}$$

$$= \Theta(n)$$

(Ans)

$T(n) = 625\,T\left(\frac{n}{5}\right) + n$

Here, $a = 625$; $b = 5$; $k = 1$; $p = 0$ and $625 > 5^1$

Applying master Theorem $a > b^k$ (case-01)

$$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log_5 625}\right) = \Theta\left(n^4\right)$$

(Ans:)

$T(n) = 2T\left(\frac{n}{6}\right) + n^3$

Here, $a = 2$; $b = 6$; $k = 3$; $p = 0$; $2 < 6^3$

Applying master theorem, $(a < b^k)$ (case-03)

$p >= 0$ hence,

$$T(n) = \Theta\left(n^k \log^p n\right) = \Theta(n^3)$$

(Ans:)

Q2.

i) Outer loop complexity = $O(n)$

Middle loop complexity = $O(n)$

Inner loop complexity = $O(1)$

Hence, time complexity = $O(n \times n) = O(n^2)$

(Ans:)

ii) Recurrence Relation —

$$T(n) = \begin{cases} 1 & ; \ n = 1 \\ T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{6}\right) + n^2 & ; \ n > 1 \end{cases}$$

iii) Here, recurrence eqn -

$$T(n) = T(\tfrac{n}{2}) + T(\tfrac{n}{4}) + T(\tfrac{n}{6}) + n^2$$

$$= T(\tfrac{n}{2}) + n^2 \quad \left[\text{As } T(\tfrac{n}{4}), T(\tfrac{n}{6}) \text{ exhaust before } T(\tfrac{n}{2}) \text{ and I'm looking for the worst case T.C.}\right].$$

Here,

$a = 1$; $b = 2$; $k = 2$; $p = 0$ and $1 < 2^2$

Applying master Theorem, $a < b^k$ (case-03)

Here, $p >= 0$

$$\therefore T(n) = O(n^k \log^p n) = O(n^2) \quad \text{(Ans:)}$$

Part 3: Searching & Sorting Algorithms

Q3.

Given, $T(n) = T(\sqrt{n}) + O(n)$

No, we can't solve this using master Theorem.

The structure of this recurrence doesn't fit the requirements of the master Theorem because, $b = 1$. But $b > 1$ is required to apply master Theorem.

Part 3: Searching & Sorting Algorithms

Q1.

i)

```
Algorithm OptimizedSearchToFindSqrt(key) {
        i ← 0
        while (i*i <= n) {
            i ← i+1
        }
        return i-1
}
```

ii)

```
Algorithm FindSqrt(key) {
        low ← 0
        high ← key
        ans ← -1
        while (low <= high) {
            mid = (low + high)//2
            if (mid * mid <= key) {
                ans ← mid
                low ← mid+1
            }
            else {
                high ← mid-1
            }
        }
        return ans
}
```

Q2.

STEP BY STEP LOGICAL INSTRUCTION –

01. Suppose, we have the temperature reading per hour in an array.

02. Initialize two pointers; one at index=0 (low) and other at index = arr.size-1 (high)

03. Run a while loop with condition (low <= high)

04. At every iteration, we calculate the mid with the formula – mid = (low + high)//2

05. Then we we check if temparature at index mid is less than both of it's previous and next index temparature. If so, we return this temparature as this is what we are looking for

06. If step-5 is false then either we're left of the minimum temparature or right of the minimum temp.

07. IF temp at mid index > temp. at mid index+1 AND temp at mid index < temp. at mid index-1 that means the minimum temp. is on our right. So, fixing the low pointer low = mid+1

08. Otherwise we fix the high pointer as high = mid-1.

Q3.

(a) As n total cows — Let's represent white cows as 0 and black cows as 1. So, an array of integers representing the given info —

$$A = [0, 0, 0, 1, 1, 1]$$

(b) To find the serial number of the first black cow, I need to find the occurrence of the first 1.

```
Algorithm firstBlackCow(arr) {
    low ← 0
    high ← arr.size - 1
    fOccurrence ← len(arr)
    while(low <= high) {
        mid ← (low+high)//2
        if(arr[mid] >= 1) {
            fOccurrence ← mid
            high ← mid-1
        }
        else {
            low ← mid+1
        }
    }
    return fOccurrence
}
```

(c)

A 7-element array that triggers the worst-case is:-

$$[1, 5, 3, 2, 4, 6, 7]$$

Brief Explanation: The array is designed so that the median of three rules always pick the 2nd smallest element of the current subarray to be the pivot. This creates $O(n^2)$ worst-case behavior.

(d)

Name of the algorithm:- Counting Sort

Time Complexity: $O(n+m)$ or simply $O(N)$

(e)

STEP BY STEP LOGICAL INSTRUCTIONS:-

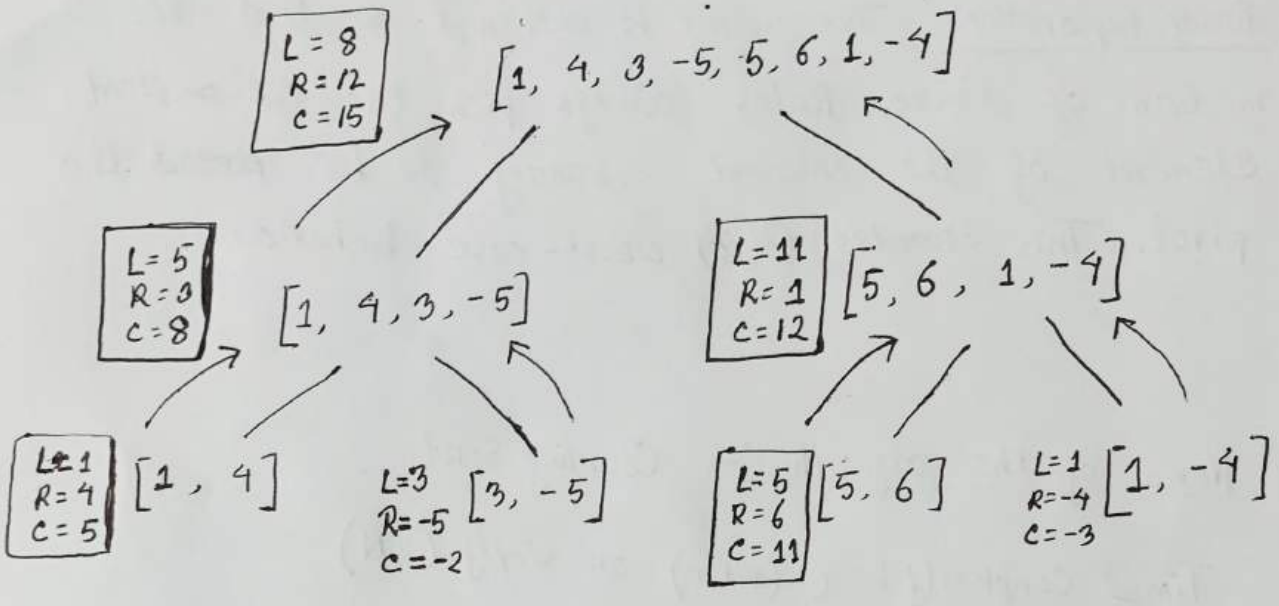
01. Create a new empty list called SortedRobots.
02. Let n be the number of robots in the input list A
03. Repeat the following n time with a for loop -
    (i) call the given function t = find_tallest(A)
    (ii) Append the robot t to the SortedRobots list.
    (iii) Remove the robot t from the input list A.

Here Time Complexity is $O(n)$ assuming appending, removing takes constant time.

Part 4: Divide & Conquer

Q1.

(i) Given input array,



So, as you can see, 5 times the cross-sum exceeds both the left-sum and the right-sum (shown in box)

(ii)

```
Algorithm maxPositiveDifStreak(A, P, r){
    if (P == r){
        if (A[P] > 0){
            return 1
        }
        return 0
    }
    q = (P+r)//2
    L = maxPositiveDifStreak(A, P, q)
    R = maxPositiveDifStreak(A, q+1, r)
    C = maxCrossStreak(A, P, q, r)

    return max(L, R, C)
}
```

```
Algorithm maxCrossStreak(A, p, q, r) {
    left_consecutive = 0
    for(i ← q to p){
        if(A[i] > 0){
            left_consecutive += 1
        }
        else{
            break
        }
    }
    right_consecutive = 0
    for(i ← q+1 to r){
        if(A[i] > 0){
            right_consecutive += 1
        }
        else{
            break
        }
    }
    cross_consecutive ← left_consecutive + right_consecutive
    return cross_consecutive
}
```

Q2.

i)

Let's assume, the numbers are in base b (e.g., $b = 10$).

Now given that A_1, A_2, A_3 are the three parts of A, each with $n/3$ digits –

$$A = A_1 \times b^{2n/3} + A_2 \times b^{n/3} + A_3$$

Similarly, $B = B_1 \times b^{2n/3} + B_2 \times b^{n/3} + B_3$

ii)

The Product,

$$AB = \left(A_1 \cdot b^{2n/3} + A_2 \cdot b^{n/3} + A_3\right) \times \left(B_1 \cdot b^{2n/3} + B_2 \cdot b^{n/3} + B_3\right)$$

$$= A_1B_1 \cdot b^{4n/3} + \left(A_1B_2 + A_2B_1\right) \cdot b^{n} + \left(A_1B_3 + A_2B_2 + A_3B_1\right) \cdot b^{2n/3}$$
$$+ \left(A_2B_3 + A_3B_2\right) \cdot b^{n/3} + A_3B_3$$

(iii)

Now, if Benjamin Implemented this directly, it would require 9 recursive multiplications of size $n/3$. This leads to –

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

which is $\Theta(n^2)$ – no faster than the standard algorithm.

To make this split actually faster than karatsuba's $\left(O(n^{1.58})\right)$, Benjamin must reduce the 9 multiplications to 5. This is possible using a method called the Toom-Cook Algorithm.

STEP-BY-STEP LOGICAL INSTRUCTIONS :-

01. Define polynomials – Let, $P(x) = A_1x^2 + A_2x + A_3$ and

$$Q(x) = B_1x^2 + B_2x + B_3$$

The final product, $AB = R\left(b^{n/3}\right)$

where, $R(x) = P(x) \times Q(x) = c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$

02. Evaluate – Choose 5 simple points to evaluate P and Q.

The standard choice is $x = 0, 1, -1, 2, \infty$

$P(0) = A_3$

$P(1) = A_1 + A_2 + A_3$

$P(-1) = A_1 - A_2 + A_3$

$P(2) = 4A_1 + 2A_2 + A_3$

$P(\infty) = A_1$

Similarly, for B we'll get -

$Q(0)$, $Q(1)$, $Q(-1)$, $Q(2)$, $Q(\infty)$

03. Conquer – Now, recursively multiply the 5 pairs of evaluated points.

$$m_1 = P(0) \times Q(0) = R(0)$$

[illegible]

$$m_2 = P(1) \times Q(1) = R(1)$$

$$m_3 = P(-1) \times Q(-1) = R(-1)$$

$$m_4 = P(2) \times Q(2) = R(2)$$

$$m_5 = P(\infty) \times Q(\infty) = R(\infty)$$

04. Combine (Interpolation) – As we now have 5 points on the product polynomial $R(x)$, we can solve a system of 5 linear equations for the 5 unknown coefficients $(c_0, c_1, c_2, c_3, c_4)$.

05. Reconstruct Final Answer – Once, we have the 5 coefficients, build the final Product :-

$$AB = c_4 \times b^{4n/3} + c_3 \times b^{3n/3} + c_2 \times b^{2n/3} + c_1 \times b^{n/3} + c_0$$

Now, this algorithm has a time complexity of

$$T(n) = 5T\left(\frac{n}{3}\right) + n$$

which is $\Theta(n^{1.46})$.

This is indeed faster that Karatsuba's $\Theta(n^{1.58})$.