CS-203 Design and analysis of algorithms

Theory Assignment Prakhar Jain 20225099

Q1

a) So, let's discuss the algorithm that take time complexity $O(n^2)$

Algorithm-1 : Count Distinct Pairs

```
1  Input : list of time interval [(a1,b1), (a2,b2) --- (an,bn)]
2  Output: Count of distinct pairs of users who are
           on the site at the same time
3  Initialize a variable count and set it to 0
4  for i = 1 to n-1
5        for j = i+1 to n
6              if max(a[i], a[j]) <= min(b[i], b[j])
7                    count = count + 1
8  return count
   end
```

This algorithm uses two nested loop to compare all the possible pairs of users. The outer loop iterate from i equal 1 to n-1 and inner loop from i+1 to n. This algorithm has two nested loop So,

$T(n) = O(n^2)$ → asymptotic upper bound

(b) Now we need to find a solution with $O(n \log n)$ time complexity)

```
COUNT_USER (A,B,n):
    r=0
    // we will do sorting to maintain the mapping b/w A & B
    SORT(A,B)
    count=0
    for i = 0 to n-1
        left =0
        right = n-1     j=0
// Binary Search while (left < right)
                mid = (left + right)//2

                if (A[mid] == B[i])
                    j=mid, count = count + mid - r
                    break
                else if (A[mid] < B[i])
                    left = mid+1
                else
                    right = mid-1

        if j == 0
            count = count + (left-1) - r
            r++
        j=0
    return count
```

Sorting will take $O(n \log n)$ time and binary search ($O(\log n)$) for n elements will take $O(n \log n)$ time. So the overall time complexity will be

$$T(n) = O(n \log n)$$

Q2 This algorithm is Selection sort algorithm which sort an array of integers in ascending order. To prove its correctness, we need to proof three things

A+B

(i) Initialization: Before the first iteration (i=1) the minIndex is set to 1. ~~which means that~~

(ii) Maintenance: Now the inner loop starts (j = i+1) to find the minimum element in the ~~ascending order~~ remaining unsorted array (from A[i+1] to A[n-1])

(a) if A[j] >= A[minIndex], ~~this~~ algorithm update minIndex to minIndex +1 because A[minIndex] remains the minimum element of array

(b) if A[j] < A[minIndex] the algorithm correctly swap the A[j] and A[minIndex] ~~so that~~ to make array sorted upto minIndex

After the inner loop completes we will have the sorted array upto minIndex.

(iii) Termination: when outer loop terminates, all the elements in the array are in ascending order.
As a result loop terminates and we can conclude that the algorithm correctly sort the array

Q3

Algorithm - 2

```
1. Input: List A = {a0, a1, a2, a3, ... an-1} of n items
2 Output: min{a0, a1, a2, ... an-1}
3 if n=0 then
4     return None
5 end
6 min_val = A[0]
7 for i = 1 to A.length - 1
8     min_val = min(minval, A[i])
9 return min_val
```

Q4 The algorithm is correct. Here is the brief argument for its correctness:

(a) Base case: if list A has only one item (n=1) the algorithm returns the only item which is minimum The base case ensure the correctness of algorithm for the smallest input

(b) Recursive Step: For larger list (n>1) the algorithm divides the list into two halves A_1 and A_2 and recursively apply Find minimum func^n to these sub list. Ans

The algorithm correctly works and return the correct minima because the minimum value exist in either of A_1 or A_2 and finding minimum from A_1 and A_2 correctly identifies the minimum in original list.

Running time analysis

(a) In base case algorithm perform constant amount of work so $T(1) = O(1)$

(b) In recursive step the algorithm divide the list into two sublist. This step considered as

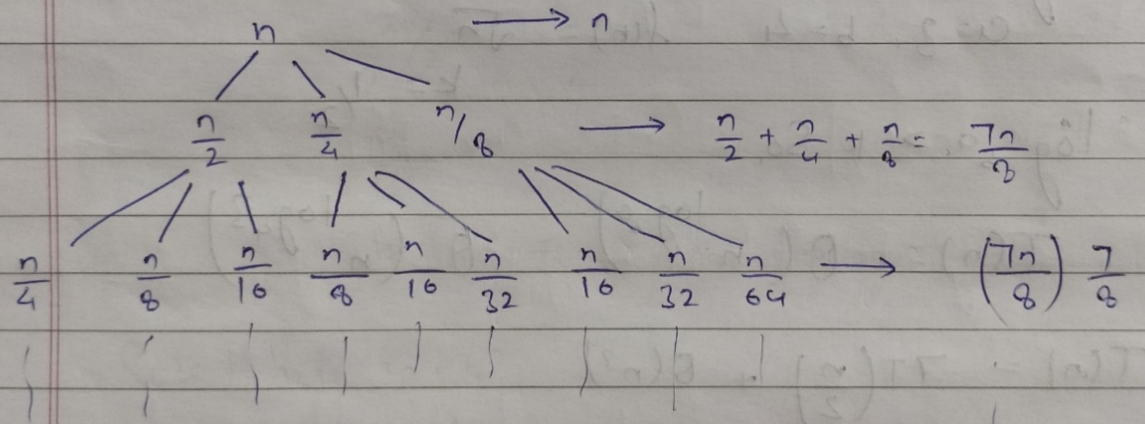
$$T(n) = 2 \times T(n/2)$$

$a = 2, b = 2$

Using masters theorem

$$T(n) = O(n \log n)$$

Q5 (a) $T(n) = T(n/2) + T(n/4) + T(n/8) + n$



Let's assume it goes upto k steps $\left(\frac{7}{8}\right)^k n$

for base case $\frac{n}{2^k} = 1 \Rightarrow k = \log n$

$$T(n) = n\left[1 + \frac{7}{8} + \left(\frac{7}{8}\right)^2 + \cdots \left(\frac{7}{8}\right)^{\log n}\right]$$

(we considered the sum of the decreasing G.P to be equal to 1 or some constant)

$$= n$$

$$T(n) = O(n)$$

(b) $T(n) = 2T(n/2) + 3n$

Using masteris theorem

$a = 2, \; b = 2 \quad f(n) = 3n$

$k = 1$

~~$T(n) = n^{\log}$~~

$\log_b a = k$ So,

$T(n) = \theta(n^k \log n)$

$T(n) = \theta(n \log n)$

(c) $T(n) = 3T(n/4) + \sqrt{n}$

Using masteris theorem

$a = 3, \; b = 4 \quad f(n) = \sqrt{n}$

$k = 1/2$

$\log_b a > k$ So,

$T(n) = \theta(n^{\log_b a}) = \theta(n^{\log_4 3})$

(d) $T(n) = 7T(\frac{n}{2}) + \theta(n^3)$

$a = 7, \; b = 2, \quad k = 3$

Using masteris theorem

$\log_b a < k$

$T(n) = \theta(n^k \log^p n)$ [p is power of log in the expression of f(n)]

$T(n) = \theta(n^3)$

(e) $T(n) = 2T(\sqrt{n}) + 1$

$\Rightarrow T(n) = 2T(n^{1/2}) + 1$

$T(n^{1/2}) = 2T(n^{1/2^2}) + 1$

Now $T(n) = 2^2 T(n^{1/2^2}) + 2 + 1$

{ up to k iteration

$T(n) = 2^k T(n^{1/2^k}) + 1 + 2 + 2^2 + \cdots 2^{k-1}$

for base case

$T(n^{1/2^k}) = T(2)$

let $n = 2^m$

$2^{m/2^k} = 2$

$\frac{m}{2^k} = 1$

$k = \log m \Rightarrow k = \log \log n$

Substitute in eqn

$T(n) = 2^{\log\log n} T(2) + \frac{1(2^k - 1)}{(2-1)}$

$= \log n + 2^{\log\log n}$

$= \log n$

$T(n) = O(\log n)$

Q6 $n|\sin(\pi n/2)| = O(n|\cos(\pi n/2)|)$

According to definition if $f(n) = O(g(n))$ then

$$f(n) \leq c_1 g(n)$$

Therefore,
we need to prove that

$$n|\sin(\frac{n\pi}{2})| \leq c_1 n|\cos(\frac{n\pi}{2})|$$

Let's take a case when $n=1$

$$n|\sin\frac{n\pi}{2}| = 1 \text{ and } c_1 n|\cos\frac{n\pi}{2}| = 0$$

So for odd values of n the relationship is invalid
Hence, it is FALSE

Q7 $n|\sin(n\pi/2)| = \Omega(n|\cos(n\pi/2)|)$

According to definition if $f(n) = \Omega(g(n))$ then $f(n) \geq c_2 g$

Therefore,
we need to prove that $n|\sin(\frac{n\pi}{2})| \geq c_2 n|\cos(\frac{n\pi}{2})|$

let's take a case when $n = 2$

$$n|\sin(\frac{n\pi}{2})| = 0 \text{ and } c_2 n|\cos\frac{n\pi}{2}| = 2c_2$$

So, for even value of n the relationship is invalid
Hence it is FALSE

Q9 Pseudo-code

```
Insertion_SORT(A,n)
    // Base case
    if n<=1
        return
    else
        INSERTION_SORT(A,n-1)
        // finding the correct position of A[n] into sorted A[1:n-1]
        x = A[n]
        i = n-1
        while i>=1 and A[i]>x
            A[i+1] = A[i]
            i--
        A[i+1] = x
```

The recurrence relation for worst case scenario is

$$T(n) = T(n-1) + n$$
$$T(n-1) = T(n-2) + n-1$$
$$\Rightarrow T(n) = T(n-2) + (n-1) + n$$

⋮ for k times

$$T(n) = T(n-k) + n + (n-1) + (n-2) + \cdots + (n-k+1)$$

for base case $n = k$

$$T(n) = n + (n-1) + (n-2) + \cdots + 1$$
$$= \frac{n(n+1)}{2}$$
$$T(n) = O(n^2)$$

Q(10) MULTIPLY (a, b, c, d)

$x = a + b$

$y = c + d$

$z = x * y$ // one multiplication

$w = (b - a) * y$ // two multiplication

$v = x * y$ // three multiplication

real = $w - z$

img = $z - v$

return real, img

Q11 To achieve matrix multiplication in $O(n^{\omega})$ time we can use a matrix multiplication algorithm based on Strassen algorithm or divide and conquer approach. This algorithm reduce the number of scaler multiplication by breaking down the matrices into smaller submatrices and using addition and subtraction in place of some multiplication

(1) Splitting :- Split the matrix A and B into half

$A \rightarrow \frac{n}{2} \times \frac{n}{2}$ $B \rightarrow \frac{n}{2} \times \frac{n}{2}$

$(A_{11}, A_{12}, A_{21}, A_{22})$ $(B_{11}\ B_{12}\ B_{21}\ B_{22})$

(2) Multiplication

$P_1 = A_{11}(B_{12} - B_{22})$

$P_2 = (A_{11} + A_{12}) * B_{22}$

$P_3 = (A_{21} + A_{22}) * B_{11}$

$P_4 = A_{22} * (B_{21} - B_{11})$

$P_5 = (A_{11} + A_{22}) * (B_{11} + B_{22})$

$P_6 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$P_7 = (A_{11} - A_{21}) * (B_{11} + B_{12})$

③ Value of Sob matrix

$C_{11} = P_5 + P_4 - P_2 + P_6$

$C_{12} = P_1 + P_2$

$C_{21} = P_3 + P_4$

$C_{22} = P_5 + P_1 - P_3 - P_7$

④ Combine the four matrix and get the result

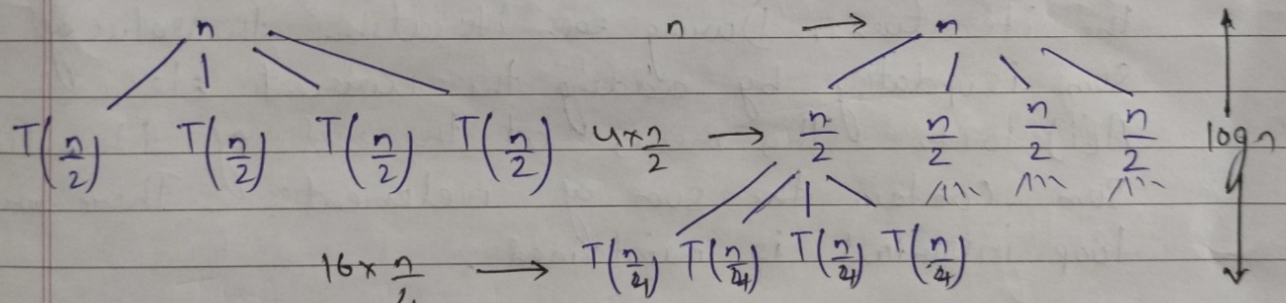
Time complexity

$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$

$T(n) = \Theta(n^{\alpha})$

$\alpha = 2.81$

(12) $T(n) = 4T\left(\frac{n}{2}\right) + n$



$T(n) = n + 4 \times \frac{n}{2} + 16 \times \frac{n}{4} + \cdots$

$= n(1 + 2 + 4 + \cdots \{2^{\log n})$

$= n\left(\frac{2^{\log n} - 1}{(2-1)}\right)$

$= n(n-1)$

$T(n) = O(n)$ → upper bound

Q8 Loop invariants: At the start of each iteration of loop the variable sum contains the sum of first i elements in the array.

loop invariant help us to understand why an algorithm is correct. In this we show three things

(i) Initialization: Before the loop starts, sum is initialized to 0. Therefore, the loop invariats is satisfied as it hold that sum of element in the subarray is 0

(ii) Maintenance: Assoming the loop invariant is true at the start of an iteration (for some i), we need to show that it remain true at the end of the iteration. During each iteration the value of sum is updating by adding the current element A[i] to it. This means at the end of iteration, sum contain the sum of i+1 elements. Therefore loop invariant is maintained.

(iii) Termination :- The loop invariant guarantees that it holds at the end of each iteration when the loop terminates, i has the value n+1. This means the loop invariant is still true.

Therefore by loop invariant we show that SUM-ARRAY return the sum of numbers in A[1:n]