17/20 — 0439

National University of Computer and Emerging Sciences

Lahore Campus

D & A of Algorithms (CS2009)

Sessional-I Exam

Total Time (Hrs): 1
Total Marks: 15
Total Questions: 3

Date: Sep 22 2025

Course Instructor(s)

Dr. MB, Dr. SK, Dr. MB, Dr. MAQ, AA, UH, SK

Mohammad Ibraheem

23L-0771 — Roll No

BCS-5J — Section

M. — Student Signature

Instructions: Answer in the space provided. Do not attach rough sheets with this exam. 6.5+8

***CLO 2:* Analyze** the time and space complexity of different algorithms by using standard asymptotic notations for recursive and non-recursive algorithms.

Question 1: [2+ 2 + 3 = 7 Marks]

(a) $f(n) = 30*2^n + 15*4^n + 3*16^n$
Which of the following statements are true about f(N)
i. $f(n) = O(4^{2n})$
ii. $f(n) = O(8^{n+2})$
iii. $f(n) = O(4^{n+4})$
iv. $f(n) = O(2^{4n})$

$(30)2^n + (15)4^n + (3)16^n \leq c.16^n$
$(30)16^n + (15)16^n + (3)16^n$
$= (48)16^n \quad c = 48$
$(30)2^n + (15)4^n + (3)16^n \leq 48(16)^n$
4^{2n}

1.5

(b) Derive a recurrence relation for the running time T(n) of Mystery. Do not solve the recurrence.

```
FUNCTION Mystery(A[1..n]):
  IF n ≤ 1:
    Return 0

  m = floor(n/3)
  L = A[1..m]
  R = A[m+1..n]

  leftResult  = Mystery(L)
  rightResult = Mystery(R)

  cross = 0
  for i from 1 TO m:
    for j from 1 TO n - m:
      If (L[i] + R[j]) MOD 7 == 0:
        cross += 1

  Retrun leftResult + rightResult + cross
```

$n/3$ $1 - n/3$
$2n/3$ $n/3 — n = \frac{2n}{3}$
$T(n/3)$
$T(2n/3)$
$n/3$
$2n/3$
1

$\frac{2n}{3}\left(\frac{n}{3}\right) = \frac{2n^2}{9}$
$= O(n^2)$

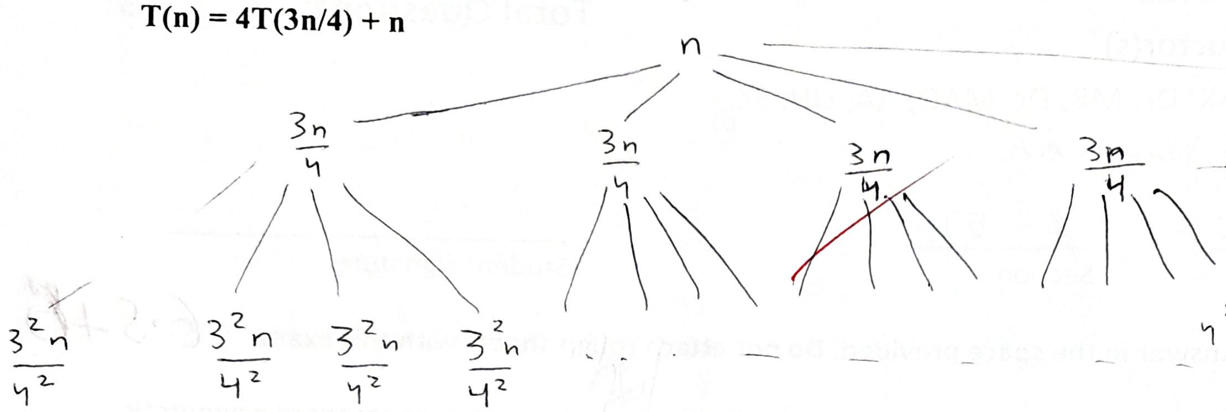
$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n^2)$

2

04

(c) Apply the recursion tree technique to solve the given recurrence and state the asymptotic time complexity of the solution.

T(n) = 4T(3n/4) + n



4^0 nodes level 0

4^1 nodes $3n$ level 1

4^2 nodes level 2 $3^2 n$

finding number of levels.

$n\left(\frac{3}{4}\right)^k = 1$

$n = \left(\frac{4}{3}\right)^k$

$k = \log_{4/3}(n)$

↓

number of levels.

at k^{th} level we get 4^k nodes

each node has weight $n\left(\frac{3}{4}\right)^k$

sum of each level $= 4^k (n)\left(\frac{3}{4}\right)^k$

$= \cancel{4^k}(n)\left(\frac{3^k}{\cancel{4^k}}\right)$

$= 3^k n$ — level cost

$$\text{Total cost} = \sum_{k=0}^{k=\log_{4/3} n} 3^k n$$

$$= n \underbrace{\sum_{k=0}^{k=\log_{4/3} n} 3^k}$$ → can be solved using geometric series

$$\frac{1\left(3^{\log_{4/3}(n)} - 1\right)}{3-1}$$

$$= \frac{1}{2}\left(n^{\log_{4/3}(3)} - 1\right)$$

$$= n^{\log_{4/3}(3)}$$

3

$O(n) = n\left(n^{\log_{4/3}(3)}\right)$

$O(n) = \cancel{n}\left(n^{43.81}\right)$

National University of Computer and Emerging Sciences

0439

Lahore Campus

CLO #1: Design algorithms using different algorithms design techniques i.e. Brute Force, Divide and Conquer, Dynamic Programming, Greedy Algorithms and apply them to solve problems in the domain of the program

Question 2: [2 Marks] Quick Sort's efficiency depends heavily on how the pivot is chosen. Describe a specific input and pivot-selection strategy that causes Quick Sort to run in $\Theta(n^2)$ time.

O(n²) when input array is already sorted and pivot is always the last (n-1) element of the array.

2

Question 3: [2+4 = 6 Marks] Recall the problem of finding the number of inversions. We are given a sequence of n numbers $a_1, \ldots, a_n$, which we assume are all distinct, and we define an inversion to be a pair $i < j$ such that $a_i > a_j$. We motivated the problem of counting inversions as a good measure of the sortedness of an array. However, one might feel that this measure is too sensitive. Let's call a pair a significant inversion **if $i < j$ and $a_i > 2a_j$**.

(a) Give an $O(n^2)$ algorithm to count the number of significant inversions between two orderings.[2 Marks]

This can be done by simply using 2 nested loops.

```
count = 0
for (int i=0; i<n-1; i++) {
    for (int j = i+1; j<n; j++){
        if (array[i] > 2*array[j]){
            count = count + 1;
        }
    }
}
```

2

 0439

(b) Give an O(n log n) algorithm to count the number of significant inversions between two orderings.[4 Marks]

we can use a modified vrsion of merge sort to solve this

```
sort_and_count(Array, low, high) {
    if (low < high) {
        mid = (high + low) / 2
        left_count = sort_and count(Array, low, mid)
        right_count = sort_and count(Array, mid + 1, high)
        whole_count = merge_and_count(Array, low, mid, high)
        return left_count + right_count + whole_count
    }
    return 0;
}
```

```
merge_and_count(Array, low, mid, high) {
  count = 0
  i = low
  j = mid + 1        Temp[high - low]
  k = 0;
  while (i ≤ mid and j ≤ high) {
      if (Array[i] < Array[j]) {
          Temp[k] = Array[i];
          k++; i++
      }
      if (Array[j] > Array
      if (Array[i] > Array[j]) {
          Temp[k] = Array[j]
          k++
          if (Array[i] > 2 * Array[j]) {
              count = count + (mid - i + 1)
          }
          j++
      }
      else {
          // Both are same copy either
          Array
          Temp[k] = Array[i]
          k++
          i++
      }
  }
```

```
  // copy the remaining arrays
  while (i < mid) {
      Temp[k] = Array[i]
      k++
      i++
  }
  while (j < high) {
      Temp[k] = Array[j]
      k++;
      j++;
  }
  // copy temp back to array
  i = low   k = 0
  while (i ≤ high) {
      Array[i] = Temp[k]
      i++
      k++
  }
  return count;
}
```