National University of Computer and Emerging Sciences

FAST School of Computing **Summer-2023** **Islamabad Campus**

CS-2009: Design and Analysis of Algorithms

Serial No:

Mid Term Exam

Total Time: 2 Hours

Total Marks: 90

Monday, 10th July, 2023

_____

Signature of Invigilator

Course Instructors

Nirmal Tariq

_____	_____	_____	_____
Student Name	Roll No.	Course Section	Student Signature

DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.

Instructions:

1. Attempt on question paper. Attempt all of them. Read the question carefully, understand the question, and then attempt it.
2. No additional sheet will be provided for rough work. Use the back of the last page for rough work.
3. If you need more space write on the back side of the paper and clearly mark question and part number etc.
4. After asked to commence the exam, please verify that you have **Eleven (11)** different printed pages including this title page. There are a total of **5** questions.
5. Calculator sharing is strictly prohibited.
6. Use permanent ink pens only. Any part done using soft pencil will not be marked and cannot be claimed for rechecking.

	Q-1	Q-2	Q-3	Q-4	Q-5	Total
Marks Obtained						
Total Marks	**20**	**20**	**20**	**20**	**10**	**90**

Question 1 [12+ 5 + 3=20 Marks]

Time Complexity Analysis

a) Find the time complexity of the following pseudocodes and provide justification. [3 x 4 Marks]

	Pseudocodes/Algorithms	Time Complexity (Justification)
1.	Function (int n) { for (int i = 1; i< n* n; i++) { sum ++; if (i % 2 == 0) break; } }	O (1)

2.	for (int i = 1; i < n; i++)
	for (int k = 1; k < 3; k++)
	for (int j = n; j < i; j ++)
	count++;
		O (infinity)

3.	i = n;
	while (i > 1)
	{ j = i;
	while (j < n)
	{ k= 0;
	while (k < n)
	{ k = k + 2; }
	j = j * 2 : }
	i = i / 2 ; }
		O (N log ^{2} N)

4.	Algorithm : Exam_Procedure A(S : string) n = length [S] j = 2 x n begin loop break when j <= 0 i = n begin loop break when i <=1 i = i/2 end loop j = j – 1 end loop	O (N log N)

b) Sort all the functions below in increasing order of asymptotic (big-O) growth. If some have the same asymptotic growth, then clearly mention that. Note that lg means, base 2. [5 Marks]

2n , 2 lg n , 2 lg lg n , n ^2 , n ^ lg n , 2 ^ n , 2

2, 2 lg lg n, 2 lg n, 2 n, n ^ 2, n ^ lg n, 2 ^ n

c) Fill in the table with True or False. [3 Marks]

	Statement	True / False
1.	$n (100 n^2 + 5 n / 5 + 20) \neq O (n^3)$	False
2.	$n (n^3 + 1) / 2 = O (1)$	False
3.	$7n + 3 n \lg n + n = \Theta (n^2)$	False

Question 2 [5 + 10 + 5 = 20 Marks]
Recurrence Relations

a) You are provided with two algorithms based on divide and conquer technique. You have to suggest the best algorithm to your programming team using Master's Method. Suppose you are choosing between the following two algorithms:

i. Algorithm X solves problems by dividing them into eight subproblems of half the size, recursively solving each subproblem, and then combining the solutions in constant time.

ii. Algorithm Y solves problems of size n by dividing them into seven subproblems of size n/3, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

Write a recursive relation of each of the above algorithms. What are the running times of each of these algorithms (in asymptotic notation), and which would you choose? [Note: Solution with direct answer will not be acceptable] [5 Marks]

i) T(n) = 8 T(n/2) + O(1); By the master method, T(n) = Θ(n^3), Since $\log_2 8 = 3 > 1$.

ii) T(n) = 7T(n/3) + n ^2 : By the master method, T(n) = Θ(n^2), since $\log_3 (7) < \log_3 (9) = 2$.

b) Solve the following Recurrence relations by using Recursion Tree Method and Iteration Method. [5 + 5 = 10 Marks]

T(n) = 4 T (n / 2) + n , n > 1

Recursion Tree:

Level #	#of node	Recursion tree	Level Sum
0	1	n	n
1	4	n/2 n/2 n/2 n/2	4(n/2) = 2n
2	4^2	n/4 n/4 n/4 n/4 -------	4^2 (n/4) = 4n
3	4^3	n/8 n/8 ----------	4^3(n/8) = 8n
.	.	.	.
.	.	.	.
.	.	.	.
i	4^i	$n/2^i$	$4^i(n/2^i) = 2^i n$
			$4^{h-1}(n/2^{h-1}) = 2^{h-1} n$
h	4^h	T(1) T(1) T(1) T(1)	4^h

$$T(n) = 4^h T(1) + \sum_{i=0}^{h-1} 2^i n$$

$$\frac{n}{2^h} = 1 \rightarrow h = log_2 n \leftarrow equation\ \#1$$

from *equation* #1

$$T(n) = 4^{log_2 n} T(1) + \sum_{i=0}^{h-1} 2^i n$$

$$T(n) = n^{log_2 4} T(1) + \sum_{i=0}^{h-1} 2^i n$$

$$n^{log_2 4} = n^2 \ since\ log_2 4 = 2 \leftarrow equation\ \#2$$

from *equation* #2

$$T(n) = n^2 T(1) + n \sum_{i=0}^{h-1} 2^i$$

$$T(n) = n^2 T(1) + n\,[1 + 2 + 2^2 + \cdots .. + 2^{h-1}]$$

Increasing Geometric Series $r = 2 \quad a = 1$

$$S = a + ar + ar^2 + \ ... + ar^{k-1} = \frac{a(r^k - 1)}{r - 1} \qquad r > 1$$

Since $r = 2 \quad a = 1$

$$n\,[1 + 2 + 2^2 + \cdots + 2^{h-1}] = \frac{1(2^h - 1)}{2 - 1} = 2^h - 1$$

from Equation #1

$$= 2^{log_2 n} - 1$$

$$= n^{log_2 2} - 1 \quad = n - 1 \quad Since\ log_2 2 = 1$$

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

$$T(n) \ \ = \theta(n^2)$$

c) An algorithm solves problem by dividing a problem of size n into 3 sub-problems of one-fourth the size and recursively solves the smaller sub-problems. It takes constant time to combine the solutions of the sub-problem. Write recurrence relation of such problem. Also provide the time complexity of derived recurrence relation using any method of your choice. [5 Marks]

T (n) = 3 T (n / 4) + O (1) , n > 1

O (n ^ 0.79) using Master's Theorem

Question 3 [10+10 = 20 Marks]
Divide-n-Conquer

a) Write a divide-and-conquer algorithm that finds the maximum difference between any two elements of a given array of n numbers in O(n) time. Justify briefly that your algorithm is correct and runs within the required time bound

For example:
Input: A[] = {1, 4, 5, 7, 9}
Output: 8
For full marks, your answer must make use of the divide-and-conquer method. Partial marks will be given for an O(nlogn) divide-and-conquer method. [10 Marks]
Solution:

SAMPLE SOLUTION:

The maximum difference is simply the difference between the maximum and the minimum elements. We find the pair (minimum element, maximum element) recursively (using divide-and-conquer), then return their difference.

```
MaxDiff(A):
    min, max ← MinMax(A)
    return max − min

MinMax(A):
    if n = 1:        // n = size of A
        return A[1], A[1]
    else:
        m ← ⌊n/2⌋
        n1, m1 ← MinMax(A[1...m])
        n2, m2 ← MinMax(A[(m+1)...n])
        return min(n1, n2), max(m1, m2)
```

Procedure MinMax runs in time $\Theta(n)$ because its worst-case running time satisfies the recurrence $T(n) = 2T(n/2) + \Theta(1)$, so MaxDiff also runs in time $\Theta(n)$.

b) Following questions are based on QUICK SORT algorithm. [3+3+4=10 Marks]

i. For sorting an input list of size 32 by QuickSort recursive algorithm, how many QuickSort calls will be made? Presume the Pivot choice policy is perfect and QuickPartition always divides the list in half. Explain in one or two lines. [3]

If the input list size is 32, and the QuickSort algorithm with perfect pivot choice policy always divides the list in half during QuickPartition, the number of QuickSort recursive calls made will be 5.

Explanation: Each recursive call divides the list in half, resulting in a tree with a depth of 5 (log base 2 of 32 is 5). Therefore, there will be 5 recursive calls to sort the input list of size 32 using QuickSort.

ii. What value of q does PARTITION return when all the elements in the array A[p . . . r] have the same value? Explain it briefly. [3]

The value of q will be the rightmost index of the pivot element. Since all elements are the same, there will be no actual partitioning happening, and the partition function will just return the same array with q being the rightmost index (r) of the pivot element.

iii. How can you improve the performance of the partition if all the elements in array are same? Write your improved partition algorithm? [4]

Create a check that returns -1 if values are same.
Individual solution also accepted if right.

```
Partition(A, p, r)
{
    x = A[r]              // x is pivot
    i = p - 1
    for j = p to r – 1
    {
        do if A[j] <= x
            then
            {
              i = i + 1
              exchange A[i] ↔ A[j]
            }
    }
    exchange A[i+1] ↔ A[r]
    return i+1
}
```

Question 4 [5+5 +10= 20 Marks]
Heaps

i. Is the array with the values {23, 17, 14, 6, 13, 10, 1, 5, 7, 12} a max-heap? Show with the help of the binary heap data structures. If not, then clearly mention why.[5]

Not a heap as 7 at leaf is greater than 6 which is the parent.

ii. Illustrate the operation of MAX-Heapify (A , 3) on the following array. [5]

A = { 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0}

Resulting array will be {27, 17, 10, 16, 13, 9, 1, 5, 7, 12, 4, 8, 3, 0}

```
Alg: MAX-HEAPIFY(A, i, n)
1.  l ← LEFT(i)
2.  r ← RIGHT(i)
3.  if l ≤ n and A[l] > A[i]
4.    then largest ←l
5.    else largest ←i
6.  if r ≤ n and A[r] > A[largest]
7.    then largest ←r
8.  if largest ≠ i
9.    then exchange A[i] ↔ A[largest]
10.         MAX-HEAPIFY(A, largest, n)
```

iii. Prove Mathematically why is the running time of Build-MaxHeap is not O(nlgn)? [10]

Done in class.

Alg: BUILD-MAX-HEAP(A)

1. n = length[A]
2. for i ← ⌊n/2⌋ downto 1 } O(n)
3. do MAX-HEAPIFY(A, i, n) O(lgn)

Question 5 [7+3 = 10 Marks]
String Matching Algorithms

i. Working modulo q =11, how many spurious hits does the Rabin-Karp matcher encounter in the text T = 3141592653589793 when looking for the pattern P = 26? You are required to write complete list of running hashes that are calculated. [7]

For pattern P =26,
26 mod 11 = 4
For Text T= 3141592653589793
31 mod 11 = 9
14 mod 11 = 3
41 mod 11 = 8
15 mod 11 = 4
59 mod 11 = 4
92 mod 11 = 4
26 mod 11 = 4
65 mod 11 = 10
53 mod 11 = 9
35 mod 11 = 2
58 mod 11 = 3
89 mod 11 = 1
97 mod 11 = 9
79 mod 11 = 2
93 mod 11 = 5
So, 3 spurious hits on 15, 59, 92.

```
RABIN-KARP-MATCHER(T, P, d, q)
1  n ← length[T]
2  m ← length[P]
3  h ← d^(m−1) mod q
4  p ← 0
5  t0 ← 0
6  for i ← 1 to m                    ▷ Preprocessing.
7      do p ← (dp + P[i]) mod q
8         t0 ← (dt0 + T[i]) mod q
9  for s ← 0 to n − m                ▷ Matching.
10     do if p = ts
11           then if P[1..m] = T[s + 1..s + m]
12                   then print "Pattern occurs with shift" s
13        if s < n − m
14           then ts+1 ← (d(ts − T[s + 1]h) + T[s + m + 1]) mod q
```

ii. What is the best way according to you that can help reduce spurious hits in the above example? [3]

By increasing the value of q in above example, we can avoid spurious hits as the range will be wider.