Design and Analysis of Algorithms (CS2009)

Sessional-I Exam

Total Time (Hrs): 1
Total Marks: 65
Total Questions: 6

Course Instructor(s):
Ms. Noor ul Ain, Mr. Muhammad Owais Idrees, Ms. Hira Mastoor
Section(s): (A,B,C,D,E,F,G)-CS

Date: Sep 23, 2025

23I-0782 — **Roll No**
G — **Course Section**
Umar — **Student Signature**

Do not write below this line.

Attempt all the questions.

1. Read the question carefully, understand the question, and then attempt your answers. You can write any assumptions clearly along with the answer if required.
2. **Solve Q1, 2, 3, 4 on the question paper. Solve Q5, 6 on the answer sheet.**
3. Verify that you have **Seven (07)** printed page of the question paper including this page.
4. Use answer sheet for the rough work for questions 1,2,3,4
5. Overwriting and cutting in the final answer on the question paper will result in zero score.
6. Clearly **mark your question number and part number in the answer sheet**. Attempt the questions on your answer sheet in **sequential order**. Answer all parts of questions together.
7. There are two bonus marks if you solve questions in sequential order.
8. Avoid long stories and irrelevant code while answering your question

[CLO 2. Analyze the time and space complexity of different algorithms by using standard asymptotic notations for recursive and nonrecursive algorithms]

Q1: Assume that each of the expressions below gives the processing time T(n) spent by an algorithm for solving a problem of size n. Select the dominant term(s) having the steepest increase in n and specify the Big-Oh time complexity of each algorithm. 05 **[05 marks]**

Expression	Dominant term(s)	O(·)
$n^2 \log_2 n + n(\log_2 n)^2$	$n^2 \log_2 n$ ✓	$O(n^2 \log_2 n)$
$n \log_3 n + n \log_2 n$	$n \log_2 n$ ✓	$O(n \log_2 n)$
$3 \log_8 n + \log_2 \log_2 \log_2 n$	$3 \log_8 n$ ✓	$O(\log_8 n)$
$0.01n \log_2 n + n(\log_2 n)^2$	$n(\log_2 n)^2$ ✓	$O(n \log_2^2 n)$
$0.003 \log_4 n + \log_2 \log_2 n$	$0.003 \log_4 n$ ✓	$O(\log_4 n)$

[CLO 2. Analyze the time and space complexity of different algorithms by using standard asymptotic notations for recursive and nonrecursive algorithms]

Q2: You have the following algorithm. Please fill in the following table to find out the functions in terms of the best and worst cases **[10 marks]**

Part A: Fill in the table, enter the number of times the algorithm will run, considering worst-case analysis **[2]**

Line	Statement	Cost	Times
1	`for i = 1 to A.length - 1`	c_1	$n-1$
2	`for j = 1 to A.length - i`	c_2	$\frac{n^2-n}{2}$
3	`if A[j] > A[j+1]`	c_3	$\frac{n^2-n}{2}$
4	`swap A[j] and A[j+1]`	c_4	$\frac{n^2-n}{2}$

0.2

6.5

Part B: Write the equation of the algorithm derived after the worst-case analysis. **[2]**

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

1.5

Part C: What is the lower bound of the above equation? **[2]**

$\Omega(n)$ 0.2

Part D: The table will change if you consider the best-case scenario. Based on that, rewrite the equation obtained after performing the best-case analysis. **[2]**

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

2

Part E: What is the upper bound of the above equation? **[2]**

$O(n^2)$ ✗

2

[CLO 2. Analyze the time and space complexity of different algorithms by using standard asymptotic notations for recursive and nonrecursive algorithms]

Q3: You are given some code snippets below. Please provide the time complexity of each snippet in terms of Big-O notation. **[3+3+3 = 9 marks]**

a. A Smiley Function

```
int smiley(int n) {
    if (n < 5)
        return n * n;
    else {
        for (int i = 0; i < 10,000; i++) {
            print i;
        }
        return smiley(n / 2);
    }
}
```

Time Complexity : O(log n)

b. A Sunny Function

```
void sunny(int n, int sum) {
    for (int i = 1; i < n * n; i++) {
        {
            for (int j = 0; j < i; j++) {
                sum++;
            }
        }
    }
}
```

Time Complexity : $O(n^4)$

c. A Happy Function

```
void happy(int n, int sum) {
    int k = 1;
    sum=1
    while (k < n) {
        for (int i = 0; i < k; i++) {
            sum++;
        }
        k=k*2;
    }
    for (int j = sum*sum; j > 0; j--) {
        k++;
    }
}
```

Time Complexity : $O(n^2)$

[CLO 2. Analyze the time and space complexity of different algorithms by using standard asymptotic notations for recursive and nonrecursive algorithms]

Q4: Find the recurrence relation for the given code snippet and determine its time complexity using the Master Theorem. **[7+7+7 = 21marks]**

Code snippet	Recurrance Relation	Time Complexity

`Fun(n)`
`{`
`  if (n<=1)`
`  return n`
 	
`  sum=0;`
`  for (i=2; i<=n; i=i++)`
`  for (j=1; j<=i; j=j*2)`
`  sum=sum + i*j;`
 	
`  for(int i=1; i<=5; i++)`
`  sum = sum + (7 * Fun(n/10)`
`  return sum;`
`}`
	(1)
	$T(n) = \begin{cases} T(1) = O(1) & n < 1 \\ 7T(\frac{n}{10}) + O(n \log n) & n \geq 1 \end{cases}$
		$O(n^{0.845} \log n)$

`fun(n)`
`{`
`  if (n<=1)` → (1)
`  return n`
 	
`  sum=0;` → log(n)
`  for (i=1; i<=n; i=i*2)`
`  sum++`
 	
`  sum=sum+(4+fun(n/4))+(7* fun(n/4))`
`  return sum;`
`}`
	(1)
	$T(n) = \begin{cases} T(1) = O(1) & n < 1 \\ 8T(\frac{n}{4}) + O(n \log n) & n \geq 1 \end{cases}$
		$O(n^{3/2})$

`Fun(n)`
`{`
`  if (n<=1)`
`  return n` → (1)
`  sum =2`
`  while(n>1)`
`  sum+=n`
`  n=n/6` → $\log_6 n$
 	
`  sum = fun(n/6)+fun(n/6)+fun(n/6)`
`  return sum;`
`}`
	(1)
	$T(n) = \begin{cases} T(1) = O(1) & n < 1 \\ 3T(\frac{n}{6}) + O(n \log_6 n) & n \geq 1 \end{cases}$
		$O(n^{0.613})$

Q5: Solve the following recurrence using recursion tree method and provide the tighter bound (big theta) of the following recurrence relation **[10 marks]**

$$\mathbf{T(n) = T(n/4) + T(3n/4) + O(n)}$$

Note: Clearly mark cost at each level along with the tree, show complete derivation of the answer.

Q6: Use iteration method to solve the following recursive relation **[10 marks]**

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + \log n & n > 0 \end{cases}$$

Formula Guide:

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) .$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Master Method: 2

The following theorem can be used to determine the running time of divide and conquer algorithms. For a given program (algorithm), first we try to find the recurrence relation for the problem. If the recurrence is of the below form then we can directly give the answer without fully solving it. If the recurrence is of the form $T(n) = aT(\frac{n}{b}) + \Theta(n^k log^p n)$, where $a \geq 1, b > 1, k \geq 0$ and p is a real number, then:

1) If $a > b^k$, then $T(n) = \Theta\left(n^{log_b^a}\right)$
2) If $a = b^k$
   a. If $p > -1$, then $T(n) = \Theta\left(n^{log_b^a} log^{p+1} n\right)$
   b. If $p = -1$, then $T(n) = \Theta\left(n^{log_b^a} loglogn\right)$
   c. If $p < -1$, then $T(n) = \Theta\left(n^{log_b^a}\right)$
3) If $a < b^k$
   a. If $p \geq 0$, then $T(n) = \Theta(n^k log^p n)$
   b. If $p < 0$, then $T(n) = O(n^k)$

Important Summation Formulas

1. $\sum_{i=l}^{u} 1 = \underbrace{1+1+\cdots+1}_{u-l+1 \text{ times}} = u-l+1$ (l, u are integer limits, $l \le u$); $\sum_{i=1}^{n} 1 = n$
2. $\sum_{i=1}^{n} i = 1+2+\cdots+n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$
3. $\sum_{i=1}^{n} i^2 = 1^2+2^2+\cdots+n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$
4. $\sum_{i=1}^{n} i^k = 1^k+2^k+\cdots+n^k \approx \frac{1}{k+1}n^{k+1}$
5. $\sum_{i=0}^{n} a^i = 1+a+\cdots+a^n = \frac{a^{n+1}-1}{a-1}$ $(a \ne 1)$; $\sum_{i=0}^{n} 2^i = 2^{n+1}-1$
6. $\sum_{i=1}^{n} i2^i = 1\cdot 2+2\cdot 2^2+\cdots+n2^n = (n-1)2^{n+1}+2$
7. $\sum_{i=1}^{n} \frac{1}{i} = 1+\frac{1}{2}+\cdots+\frac{1}{n} \approx \ln n+\gamma$, where $\gamma \approx 0.5772\ldots$ (Euler's constant)
8. $\sum_{i=1}^{n} \lg i \approx n \lg n$

Sum Manipulation Rules

1. $\sum_{i=l}^{u} ca_i = c\sum_{i=l}^{u} a_i$
2. $\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$
3. $\sum_{i=l}^{u} a_i = \sum_{i=l}^{m} a_i + \sum_{i=m+1}^{u} a_i$, where $l \le m < u$
4. $\sum_{i=l}^{u} (a_i - a_{i-1}) = a_u - a_{l-1}$

Properties of Logarithms

1. $\log_a 1 = 0$
2. $\log_a a = 1$
3. $\log_a x^y = y \log_a x$
4. $\log_a xy = \log_a x + \log_a y$
5. $\log_a \frac{x}{y} = \log_a x - \log_a y$
6. $a^{\log_b x} = x^{\log_b a}$
7. $\log_a x = \frac{\log_b x}{\log_b a} = \log_a b \log_b x$

Approximation of a Sum by a Definite Integral

$$\int_{l-1}^{u} f(x)dx \le \sum_{i=l}^{u} f(i) \le \int_{l}^{u+1} f(x)dx \quad \text{for a nondecreasing } f(x)$$

$$\int_{l}^{u+1} f(x)dx \le \sum_{i=l}^{u} f(i) \le \int_{l-1}^{u} f(x)dx \quad \text{for a nonincreasing } f(x)$$

Floor and Ceiling Formulas

The *floor* of a real number x, denoted $\lfloor x \rfloor$, is defined as the greatest integer not larger than x (e.g., $\lfloor 3.8 \rfloor = 3$, $\lfloor -3.8 \rfloor = -4$, $\lfloor 3 \rfloor = 3$). The *ceiling* of a real number x, denoted $\lceil x \rceil$, is defined as the smallest integer not smaller than x (e.g., $\lceil 3.8 \rceil = 4$, $\lceil -3.8 \rceil = -3$, $\lceil 3 \rceil = 3$).

1. $x - 1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x + 1$
2. $\lfloor x + n \rfloor = \lfloor x \rfloor + n$ and $\lceil x + n \rceil = \lceil x \rceil + n$ for real x and integer n
3. $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$
4. $\lceil \lg(n+1) \rceil = \lfloor \lg n \rfloor + 1$

Miscellaneous

1. $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ as $n \to \infty$ (Stirling's formula)
2. Modular arithmetic (n, m are integers, p is a positive integer)

$$(n + m) \bmod p = (n \bmod p + m \bmod p) \bmod p$$

$$(nm) \bmod p = ((n \bmod p)(m \bmod p)) \bmod p$$