INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE – 247 667
Data Structures (CSC-102)

Assignment 01 Spring 2026

---

1. Rank the following functions by increasing order of growth (i.e., the slowest-growing first, the fastest-growing last):

   $(\log\log(n))^{1.9},\ \log(n!),\ \sqrt{n},\ n!,\ n^{1.05},\ n\log(n),\ 3^n,\ (\log(n))^{0.5},\ n^{1.9}, 8\log(n),\ 2^{\log(n)},\ n^3$

   where all the logarithms are to the base 2. If two functions have equal orders of growth then list them grouped together.

2. (a) Compute the asymptotic time complexity of the following algorithm. You may assume that n is a power of 2. (NOTE: It doesn't matter what this algorithm does!)

```
float Arbitary_task(Arr){
  n = Arr.size;
  if (n-1 == 0){
    return Arr[0];
  }
  let Arr1,Arr2 be arrays of size n/2
  for (i=0; i <= (n/2)-1; i++){
    Arr1[i] = Arr[i]*2;
    Arr2[i] = (Arr[n/2 + i])*0.5;
  }
  for (i=0; i<=(n/2)-1; i++){
    for (j=i+1; j<=(n/2)-1; j++){
      if (Arr1[i] == 0.5*Arr2[j])
      Arr2[j] = 10;
    }
  }
  temp1 = Arbitary_task(Arr1);
  temp2 = Arbitary_task(Arr2);
  return min(temp1,temp2);
}
```

   (b) Suppose the above algorithm takes 5 seconds to complete for an input size of 32. Approximately how long does it take in minutes to solve a problem of size 128?

3. True/False (with justification): Can you say that the function $f(n) = 2^{\sqrt{\log(n)}}$ is $O(n)$, $O(n^{1.9})$, $O(\log(n))$, $o(n)$, $o(\log(n))$, $\Omega(n)$, $\Omega(1)$, $\Omega(\log(n))$, $\omega(n^{0.5})$, $\omega(1)$, $\Theta(n)$, $\Theta(n^{1.9})$, $\Theta(\log(n))$.

4. Prove that:

   (a) $log(n!) = \Theta(nlog(n))$

(b) $n! = \omega(2^n)$

(c) $n! = O(n^n)$

(d) $\sum_{i=0}^{n} i^2 = \Theta(n^3)$

(e) $n^{1.001} + nlog(n) = \Theta(n^{1.001})$

5. Determine the time complexity of the following code segments. In each case, justify your answer.

(a)
```
sum = 0;
for (int i=1; i<= n; i++)
  sum += rand() + i;
```

(b)
```
sum = 0;
for (int i=1; i<= n; i++)
  for (int j=i; j<= n; j++)
    sum += i * j;
```

(c)
```
sum = 0;
i = n;
while (i > 1) {
  sum += i;
  i /= 3;
}
```

(d)
```
sum = 0;
for (int i=1; i<= n; i++)
  sum += rand();
for (int j=1; j<= 2*m; j++)
  sum += rand();
```

6. Given two functions $p(n)$ and $q(n)$, find the function which is asymptotically bigger

(a) $p(n) = 3\,n^{1.5} + 15, \quad q(n) = 150\,n^2 + 90$

(b) $p(n) = 40\,n^2\,2^n, \quad q(n) = 70\,n\,2^n + 25\,n$

(c) $p(n) = n, \quad q(n) = n^{(1+\cos(n))}$

7. Express the following functions in terms of $O,\ \Omega,\ \Theta,\ o,\ \omega$ notations

(a) $15\,n^{1.5} + 25\,n - 35$

(b) $30\,n^2\log(n) + 40\,n^2$

(c) $22\,n^3 + 35\,n^2\log(n)$

(d) $100\,n\,2^n - 120\,n^3$

8. Use the definitions of Big-O and Big-Omega notations show the following:

(a) $25\,n^3\ \log(n) + 7\,n^2 \neq O(n^3)$

(b) $14\,n^3 + 9\,n^2 \neq O(n^2)$

(c) $18\,n^3\ \log(n) + 6\,n^2 \neq \Omega(n^4)$

(d) $12\,n^3 + 7\,n^2 \neq \Omega(n^3 \log(n))$

(e) $\frac{n^2}{\log(n)} \neq \Theta(n^2)$

9. Let $A(n)$ be an algorithm's time complexity function where $A(n)$ is defined as follows: for $n > 1$, $A(n) = 2A(\sqrt{n}) + log^2(n)$, and $A(1) = 1$. Prove by induction or otherwise that $A(n)$ is $O(log^2(n))$. Also describe the choice of constant from the definition of 'time complexity'.

10. Consider a probabilistic algorithm whose running time depends on the input size n and a random variable X, which follows a geometric distribution with the probability of success p. The time complexity function $P : \mathbb{N} \to \mathbb{R}$ is given by:

$$P(n) = \mathbb{E}[X] \cdot (n^2 log(n) + n log^2(n))$$

where $\mathbb{E}[X]$ is the expected value of X.

(a) Determine $P(n)$ explicitly in terms of n and p.

(b) Analyze and describe the time complexity of the algorithm in Big-O notation as p approaches 0.

(c) Discuss how the time complexity of the algorithm varies with different values of p and whether there is a threshold value of p that makes the algorithm run in polynomial time.

11. Consider an algorithm that divides an input of size n into $\lceil \sqrt{n} \rceil$ subproblems, each of size at most $\lfloor \frac{n}{\lceil \sqrt{n} \rceil} \rfloor$, recursively solves each subproblem, and then combines the solutions in $nlog(n)$ time.

(a) Write a recurrence relation for the time complexity $C(n)$ of this algorithm.
(Hint: $C(n) \approx \lceil \sqrt{n} \rceil \cdot C\left(\left\lfloor \frac{n}{\lceil \sqrt{n} \rceil} \right\rfloor\right) + nlog(n)$)

(b) Solve the recurrence relation to find the Big-O time complexity of $C(n)$, providing a step-by-step explanation of your method. (Hint: Use tools from Problem 9 to your advantage.)

(c) Discuss how the use of the ceiling and floor functions affects the analysis compared to if n were divisible by $\sqrt{n}$ without remainder.

12. Consider an algorithm that searches for an element within a sorted matrix of size $n \times n$ using a modified binary search. The matrix is sorted both row-wise and column-wise. The algorithm selects the middle column, performs a binary search to find if the element is in that column. If it is not found, it divides the matrix into two submatrices: the left side includes columns up to the floor of the middle column, and the right side starts from the ceiling of the next column to the end. The search continues recursively on the submatrix that could contain the element based on the last checked value. The merge step consists of preparing the two submatrices for further recursive calls and takes constant time.

(a) Write a recurrence relation for the time complexity $M(n)$ of this algorithm.

(b) Solve the recurrence relation to find the Big-O time complexity of $M(n)$, providing a step-by-step explanation of your method.

(c) Analyze the impact of the ceiling and floor functions on the time complexity, specifically in comparison to a standard binary search on a one-dimensional array. Discuss any potential inefficiencies that arise from the two-dimensional nature of the problem.

13. An algorithm has a complex time complexity function characterized by multiple terms and varying degrees of polynomial and logarithmic components. The time complexity $T(n)$ of the algorithm is given by the following expression:

$$T(n) = a\,n^k\,log^k n + b\,n^{k-1}\,log^{k-1} n + c\,n^k + d\,n^{k-1} + \frac{e}{log(n)}$$

where a, b, c, d, e are constants and k is a positive integer greater than 1.

(a) Analyze the terms of $T(n)$ and determine the dominant term as n approaches infinity.

(b) Prove or disprove the following statements for the general case of the function $T(n)$:

i. $T(n) \neq O(n^{k-1})$
ii. $T(n) \neq O(n^k)$
iii. $T(n) \neq \Omega(n^{k+1})$
iv. $T(n) \neq \Omega(n^k\,log(n))$
v. $T(n)$ is such that $\lim_{n\to\infty} \frac{T(n)}{n^k}$ exists and is non-zero. What does this imply about the tightness of $\Theta(n^k)$ for $T(n)$?

(c) For $k = 3$, specialize the above analysis to the given function and compare the resulting tightness of bounds to the inequalities provided in the question prompt.

14. Given two functions, f and g, defined as follows:

$$f(n) = n^{n^{n^n}}$$

$$g(n) = 2^{2^{n^n}}$$

Using the definitions of Big-O and Big-Omega notations, determine whether:

(a) $f(n) = O(g(n))$

(b) $f(n) = \Omega(g(n))$

15. Using the definitions of Big-O and Big-Omega notations, prove or disprove the following statements for positive functions f and g, where n approaches infinity:

(a) $f(n) = O(g(n))$ and $g(n) = O(f(n))$ implies $f(n) = \Theta(g(n))$.

(b) If $f(n) = O(g(n))$ and $g(n) = \Omega(f(n))$, then $f(n) = \Theta(g(n))$.

(c) If $f(n) = O(g(n))$ and $g(n) = \Theta(f(n))$, then $f(n) = O(f(n)^2)$.

(d) If $f(n) = O(g(n))$ and $g(n) = \Omega(f(n))$, then $f(n) = \Omega(f(n)log(g(n)))$.