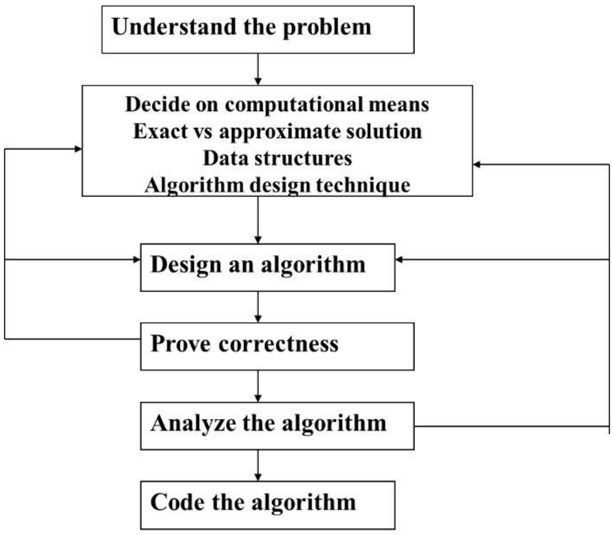
Program: BE

Date	18 June 2024	Maximum Marks	50
Course Code	**CD343AI**	Duration	90 min
7 Sem	IV Semester	CIE-I Scheme and Solution	

Design and Analysis of Algorithms

(Common to AIML/CSE/CD/CY/ISE)

<table>
<tr><th>Sl. No.</th><th>Test Questions</th><th>M</th></tr>
<tr><td>1a</td><td>


For time complexity:

- Measuring an input’s size
- Units for measuring runtime
- Orders of growth
- Worst case, best case, and average case efficiencies.
</td><td>3+2</td></tr>
<tr><td>1b</td><td>
Asymptotic notations:

O-notation: A function $t(n)$ is said to be in $O(g(n))$ if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large $n,$ there exists a constant c and n_0 such that

$$t(n) \leq cg(n) \text{ for all } n \geq n_0$$

Ω **–notation:** A function $t(n)$ is said to be in $\Omega(g(n))$ if $t(n)$ is bounded below by some constant multiple of $g(n)$ for all large $n,$ there exists a constant c and n_0 such that

$$t(n) \geq cg(n) \text{ for all } n \geq n_0$$
</td><td>3+
2 (graph)</td></tr>
</table>

	θ-notation: A function $t(n)$ is said to be in $\theta(g(n))$ if $t(n)$ is bounded above and below by some constant multiple of $g(n)$ for all large n, there exists a constant c_1 & c_2 and n_0 such that $$c_2 g(n) \le t(n) \le c_1 g(n) \text{ for all } n \ge n_0$$	

2a	Algorithm $Sum(n)$
	//input: A positive integer n
	//output: Sum of cubes of first n natural numbers
	$if\ n = 1, \quad return\ 1$
	$else$
	$return\ Sum(n-1) + (n*n*n)$
	
	Since multiplication is basic operation and its executed twice in each call, the recurrence is:
	$\boldsymbol{C(n) = C(n-1) + 2 \quad when\ n > 1}$
	$\boldsymbol{C(1) = 0}$
	
	Solving by method of backward substitution, we get
	$$C(n) = C(n-1) + 2$$
	$$C(n) = C(n-2) + 2 + 2 = C(n-2) + 2(2)$$
	$$C(n) = C(n-3) + 2 + 2(2) = C(n-3) + 2(3)$$
	$$\dots$$
	$$\dots$$
	$$C(n) = C(n-i) + 2 + 2 + \cdots . = C(n-i) + 2(i)$$
	Put $i = n-1$,
	We get $\boldsymbol{C(n)} = C(1) + 2(n-1) = 0 + 2(n-1) = \boldsymbol{2n-2}$
	Therefore, its Linear complexity.
		Algo (2)+
		Recur(1)+
		Solve (2)

2b	Algorithm $Selection\ Sort\ (A[0 \dots . n-1])$
	//input: An array of 'n' random numbers
	//output: sorted array in ascending order
	$for\ i \leftarrow 0\ \ to\ n-2\ \ do$
	$\quad min \leftarrow i$
	$\quad for\ j \leftarrow i+1\ to\ n-1\ \ do$
	$\quad\quad if\ A[j] < A[min] \quad min \leftarrow j$
	$\quad swap\ \ A[i]\ and\ A[min]$
	
	Basic operation: Comparison
	$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$
	Solving, we get quadratic time complexity.
	
	Selection sort is a brute force technique with $\boldsymbol{\theta(n^2)}$
	Merge-sort is a divide-and-conquer algorithm, has $\boldsymbol{\theta(n \log n)}$.
		Algo (2)+
		Solve (2)+
		Compare(1)

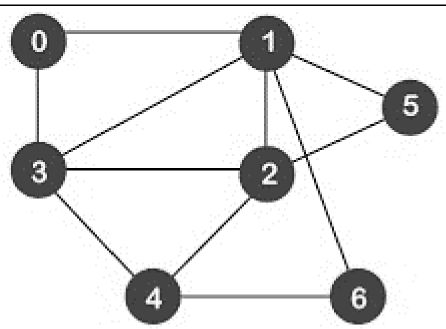
3a	Increasing order of growth: $log_5 n,\quad log_2 n,\quad \sqrt{n},\quad 3n,\quad n\, log\, n,\quad n^3,\quad 2^n$ If its argument is increased four-fold, we get $\log_4 5 + \log_5 n\,,\; 2+\log_2 n\,,\; 2\sqrt{n},\; 12n,\; 8n + 4n\log n\,, 64n^3, 16 * 2^n$	**2+3**

3b	**ALGORITHM** *Mergesort*$(A[0..n-1])$
	//Sorts array $A[0..n-1]$ by recursive mergesort
	//Input: An array $A[0..n-1]$ of orderable elements
	//Output: Array $A[0..n-1]$ sorted in nondecreasing order
	if $n > 1$
	copy $A[0..\lfloor n/2\rfloor - 1]$ to $B[0..\lfloor n/2\rfloor - 1]$
	copy $A[\lfloor n/2\rfloor..n-1]$ to $C[0..\lceil n/2\rceil - 1]$
	Mergesort$(B[0..\lfloor n/2\rfloor - 1])$
	Mergesort$(C[0..\lceil n/2\rceil - 1])$
	Merge(B, C, A) //see below
	
	ALGORITHM *Merge*$(B[0..p-1], C[0..q-1], A[0..p+q-1])$
	//Merges two sorted arrays into one sorted array
	//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted
	//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C
	$i \leftarrow 0;\; j \leftarrow 0;\; k \leftarrow 0$
	while $i < p$ **and** $j < q$ **do**
	if $B[i] \le C[j]$
	$A[k] \leftarrow B[i];\; i \leftarrow i+1$
	else $A[k] \leftarrow C[j];\; j \leftarrow j+1$
	$k \leftarrow k+1$
	if $i = p$
	copy $C[j..q-1]$ to $A[k..p+q-1]$
	else copy $B[i..p-1]$ to $A[k..p+q-1]$
	
	Recurrence: $C(n) = 2C\left(\frac{n}{2}\right) + n - 1 \quad for\;\; n > 1, \quad C(1) = 0$
		3+2

4a	Master's theorem. *The recurrence* $T(n) = aT(n/b) + cn^k$ $T(1) = c,$ *where a, b, c, and k are all constants, solves to:* $T(n) \in \Theta(n^k)$ *if* $a < b^k$ $T(n) \in \Theta(n^k \log n)$ *if* $a = b^k$ $T(n) \in \Theta(n^{\log_b a})$ *if* $a > b^k$ i. $T(n) = 2\,T\left(\frac{n}{2}\right) + n$ $a = 2, b = 2, d = 1$ Since $a = b^d$, its case 2. $\boldsymbol{T(n) \in \theta(n \log n)}$ ii. $T(n) = 8\,T\left(\frac{n}{2}\right) + 5n^2$	**2+2**

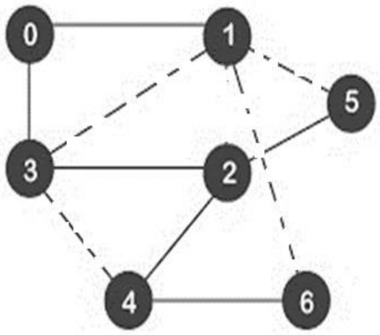
<table>
<tr><td></td><td>$$a = 8, b = 2, d = 2$$
Since $a > b^d$, its case 3. $T(n) \in \theta\left(n^{\log_2 8}\right) \Rightarrow \boldsymbol{T(n) \in \theta(n^3)}$</td><td></td></tr>
<tr><td>4b</td><td>Worst-case efficiency class for the quick sort happens when the array is almost sorted either ascending or descending.

Before the split, $(n+1)$ comparison done for an array of size n. This split is uneven and we are left with subarray of size $(n-1)$
$$C(n) = (n+1) + n + (n-1) + \cdots \ldots 3 = \frac{(n+1)(n+2)}{2} - 3$$
Hence quadratic efficiency class.

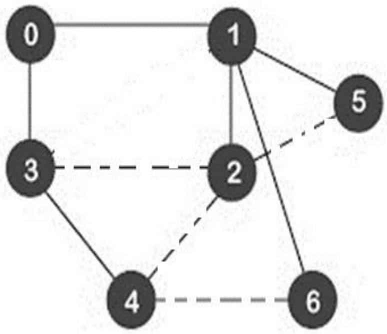
The first split happens as follows:
38, 81, 22, 48, 18, 50, 31, 58
38, 31, 22, 48, 18, 50, 81, 58
38, 31, 22, 18, 48, 50, 81, 58
18, 31, 22, **38**, 48, 50, 81, 58</td><td>**3+3**</td></tr>
<tr><td>5a</td><td>3 variations of decrease-and-conquer:
• Decrease by constant (DFS/BFS)
• Decrease by constant factor (Binary search)
• Variable size decrease (Euclid’s GCD algorithm)</td><td>**04**</td></tr>
<tr><td>5b</td><td>
i. DFS: 1, 0, 3, 2, 4, 6, 5
Stack is as shown below.
<table><tr><td>$6_{6,1}$</td><td></td></tr><tr><td>$4_{5,2}$</td><td></td></tr><tr><td>$2_{4,4}$</td><td>$5_{7,3}$</td></tr><tr><td>$3_{3,5}$</td><td></td></tr><tr><td>$0_{2,6}$</td><td></td></tr><tr><td>$1_{1,7}$</td><td></td></tr></table></td><td>**4+2**</td></tr>
</table>

<table>
<tr>
<td></td>
<td>

ii. BFS: 0, 1, 3, 2, 5, 6, 4



DFS forest



BFS forest

</td>
<td></td>
</tr>
</table>