USN | | | | | | | | | | |

Internal Assessment Test 1 – June 2024

Sub:	Analysis and Design of Algorithms				Sub Code:	BCS401	Branch: CSE
Date:	03-06-2024	Duration:	90 mins	Max Marks: 50	Sem / Sec:	IV (A, B & C)	OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1 (a)	Consider the following algorithm and derive the time complexity of the following algorithm using analytical/general framework:	6	CO1	L3

```
ALGORITHM MatrixMultiplication(A[0..n − 1, 0..n − 1], B[0..n − 1, 0..n − 1])
    //Multiplies two square matrices of order n by the definition-based algorithm
    //Input: Two n × n matrices A and B
    //Output: Matrix C = AB
    for i ← 0 to n − 1 do
        for j ← 0 to n − 1 do
            C[i, j] ← 0.0
            for k ← 0 to n − 1 do
                C[i, j] ← C[i, j] + A[i, k] ∗ B[k, j]
    return C
```

SCHEME:

Writing about general framework – 3 Marks

Writing the mathematical expression – 1 Mark

Solving and finding the time complexity – 2 Marks

SOLUTION:

GENERAL PLAN FOR ANALYZING TIME EFFICIENCY OF NON-RECURSIVE ALGORITHMS:

1. Decide on the parameter indicating an input's size.
2. Identify the algorithm basic operation.
3. Check whether the number of times the basic operation is executed depends only on the input size.
4. If it depends on another property, check best-case, worst case & average case.
5. Set up a sum expressing the no. of times the algorithm's basic operation is executed.
6. Represent it using standard formulae.

$$\sum_{i=1}^{u} c a_i = c \sum_{i=1}^{u} a_i$$

$$\sum_{i=1}^{u} (a_i \pm b_i) = \sum_{i=1}^{u} a_i \pm \sum_{i=1}^{u} b_i$$

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

Analysis

1. The input size is the order of the matrix 'n'.
2. The basic operations are the two arithmetical operations i.e; multiplication and addition.
3. $M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1$

$= n^3$.

$T(n) \simeq C_m M(n) = C_m(n^3)$.

$= C_m M(n) + C_a A(n)$

$= C_m n^3 + C_a n^3 = (C_m + C_a)n^3$.

</td><td></td><td></td><td></td></tr>
<tr><td>1 (b)</td><td>

Design an algorithm for performing sequential search and compute best case, worst case and average case efficiency with suitable notations.

SCHEME:

Algorithm – 2 Marks

All cases efficiency and time complexity – 2 Marks

SOLUTION:

Seaquential Search:

It is a general searching problem. It compares the sucessive elements of a given list with a given search key untill either a match is encountered or the list is exhausted without finding a match.

Algorithm: SeaquentialSearch (A[0....n], K)

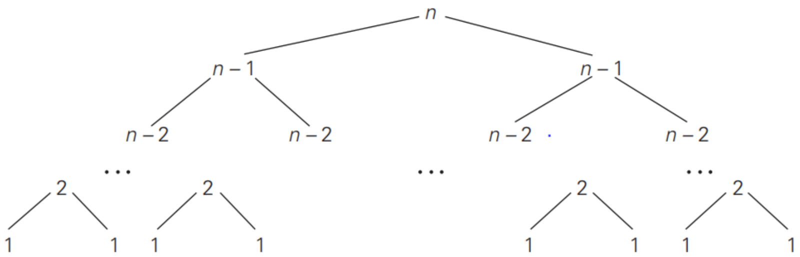
// Implements a seaquential search with a search key

// Input: An array A of n elements and a search key K.

// Output: The index of the element in A[0....n-1] or -1 if element is not found.

```
i ← 0
while A[i] ≠ K do
    i ← i+1
if i<n return i
    else return -1
```

</td><td>4</td><td>CO1</td><td>L2</td></tr>
</table>

<table>
<tr>
<td>2 (a)</td>
<td>

The above diagram represents a recursive tree with the recursive calls. From the diagram, identify the recurrence relations and solve the same using analytical framework to find the time efficiency.
SCHEME:
Writing the recurrence relation - 2 Marks
Solving and finding the time complexity – 3 Marks

SOLUTION:

SOLUTION
The recurrence relations from the above tree are written as follows:
$M(n) = M(n-1) + 1 + M(n-1) \quad \text{for } n > 1.$

$M(n) = 2M(n-1) + 1 \quad \text{for } n > 1,$
$M(1) = 1.$
$M(n) = 2M(n-1) + 1 \qquad \text{sub. } M(n-1) = 2M(n-2) + 1$
$= 2[2M(n-2) + 1] + 1 = 2^2M(n-2) + 2 + 1 \quad \text{sub. } M(n-2) = 2M(n-3) + 1$
$= 2^2[2M(n-3) + 1] + 2 + 1 = 2^3M(n-3) + 2^2 + 2 + 1.$
$M(n) = 2^iM(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2 + 1 = 2^iM(n-i) + 2^i - 1.$

$M(n) = 2^{n-1}M(n-(n-1)) + 2^{n-1} - 1$
$= 2^{n-1}M(1) + 2^{n-1} - 1 = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1.$

The time efficiency is O (2^n)</td>
<td>5</td>
<td>CO1</td>
<td>L3</td>
</tr>
<tr>
<td>2 (b)</td>
<td>Explain Big- O, Big – Ω, Big -Θ notations with examples.

SCHEME:
Notations and examples – 2 Marks each for Big 0 and Big omega
1 Mark for big theta

SOLUTION:</td>
<td>5</td>
<td>CO2</td>
<td>L2</td>
</tr>
</table>

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

O-notation

DEFINITION A function $t(n)$ is said to be in $O(g(n))$, denoted $t(n) \in O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large n, i.e., if there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n) \le cg(n) \quad \text{for all } n \ge n_0.$$

$$100n + 5 \le 100n + 5n \text{ (for all } n \ge 1) = 105n$$

Ω-notation

DEFINITION A function $t(n)$ is said to be in $\Omega(g(n))$, denoted $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below by some positive constant multiple of $g(n)$ for all large n, i.e., if there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n) \ge cg(n) \quad \text{for all } n \ge n_0.$$

Here is an example of the formal proof that $n^3 \in \Omega(n^2)$:

$$n^3 \ge n^2 \quad \text{for all } n \ge 0,$$

Θ-notation

DEFINITION A function $t(n)$ is said to be in $\Theta(g(n))$, denoted $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded both above and below by some positive constant multiples of $g(n)$ for all large n, i.e., if there exist some positive constants c_1 and c_2 and some nonnegative integer n_0 such that

$$c_2 g(n) \le t(n) \le c_1 g(n) \quad \text{for all } n \ge n_0.$$

The definition is illustrated in Figure 2.3.

For example, let us prove that $\frac{1}{2}n(n-1) \in \Theta(n^2)$. First, we prove the right inequality (the upper bound):

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \le \frac{1}{2}n^2 \quad \text{for all } n \ge 0.$$

Second, we prove the left inequality (the lower bound):

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \ge \frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n \text{ (for all } n \ge 2) = \frac{1}{4}n^2.$$

Hence, we can select $c_2 = \frac{1}{4}$, $c_1 = \frac{1}{2}$, and $n_0 = 2$.

</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3(a)</td>
<td>Design an algorithm for performing Merge sort. Apply the same to the following set of numbers 4, 9, 0, -1, 6, 8, 9, 2, 3, 12

SCHEME:
Algorithm – 5 Marks
Solving the problem – 5 Marks</td>
<td>5 + 5</td>
<td>CO2</td>
<td>L3 & L2</td>
</tr>
</table>

SOLUTION:

ALGORITHM Mergesort (A [0 ...... n-1])

// sorts array A[0.... n-1] by recursive merge sort

// Input: An array A[0...... n-1] of orderable elements.

// Output: Array A [0.... n-1] sorted in non-decreasing order

if n>1

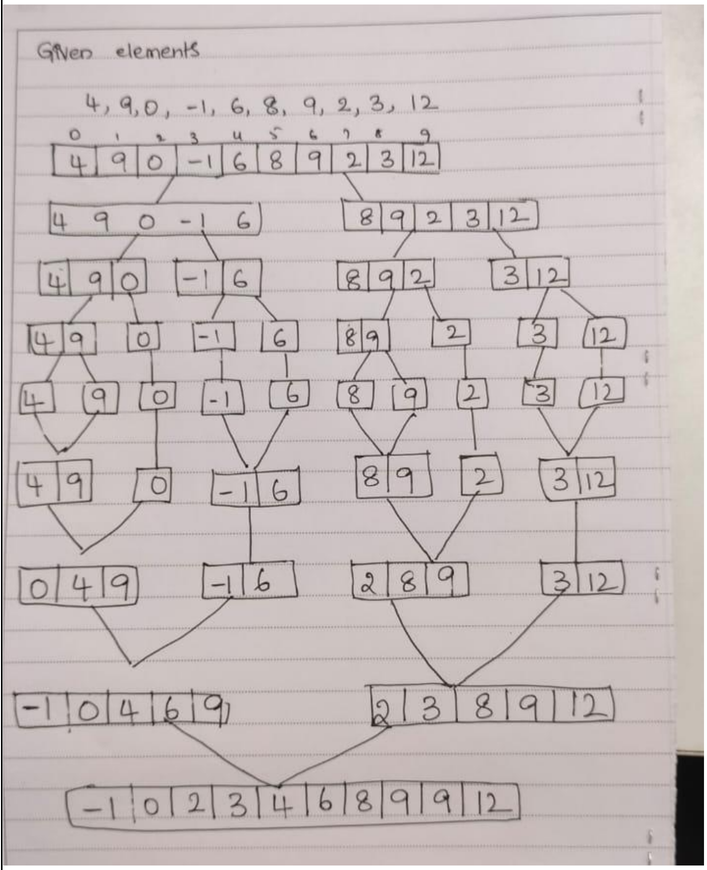
copy A [0..... (n/2)-1] to B [0.....(n/2]-1]

copy A [(n/2)+1 ..... n-1] to C [(n/2)+1 ...... n-1]

Mergesort (B[0 ..... (n/2)-1])

Mergesort (C [0...... (n/2]-1])

merge (B, C, A)



<table>
<tr>
<td>4(a)</td>
<td>

Consider the elements of array {E, X, A, M, P, L, E}. Perform bubble sort and selection sort on these elements and list out the number of comparisons taken for each sort. Identify which algorithm performed better.

SCHEME:

Bubble sort – 3 Marks

Selection sort – 3 Marks

SOLUTION:

BUBBLE SORT

$$
\begin{array}{ccccccccccccc}
E & \overset{?}{\leftrightarrow} & X & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E \\
E & & A & & X & \overset{?}{\leftrightarrow} & M & & P & & L & & E \\
E & & A & & M & & X & \overset{?}{\leftrightarrow} & P & & L & & E \\
E & & A & & M & & P & & X & \overset{?}{\leftrightarrow} & L & & E \\
E & & A & & M & & P & & L & & X & \overset{?}{\leftrightarrow} & E \\
E & & A & & M & & P & & L & & E & & |X \\
 & & & & & & & & & & & & \\
E & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E & & \\
A & & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & P & \overset{?}{\leftrightarrow} & L & & E & & \\
A & & E & & M & & L & & P & \overset{?}{\leftrightarrow} & E & & \\
A & & E & & M & & L & & E & & |P & & \\
 & & & & & & & & & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & L & & E & & & & \\
A & & E & & L & & M & \overset{?}{\leftrightarrow} & E & & & & \\
A & & E & & L & & E & & |M & & & & \\
 & & & & & & & & & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & \overset{?}{\leftrightarrow} & E & & & & & & \\
A & & E & & E & & |L & & & & & & \\
 & & & & & & & & & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & & & & & &
\end{array}
$$

SELECTION SORT:

$$
\begin{array}{l}
\vert\, E \quad X \quad \mathbf{A} \quad M \quad P \quad L \quad E \\
A \,\vert\, X \quad \mathbf{E} \quad M \quad P \quad L \quad E \\
A \quad E \,\vert\, X \quad M \quad P \quad L \quad \mathbf{E} \\
A \quad E \quad E \,\vert\, M \quad P \quad \mathbf{L} \quad X \\
A \quad E \quad E \quad L \,\vert\, P \quad \mathbf{M} \quad X \\
A \quad E \quad E \quad L \quad M \,\vert\, \mathbf{P} \quad X \\
A \quad E \quad E \quad L \quad M \quad P \,\vert\, X
\end{array}
$$

Number of Comparisons in Bubble sort and Selection sort are: 21

</td>
<td>6</td>
<td>CO2</td>
<td>L3</td>
</tr>
<tr>
<td>4(b)</td>
<td>

Design an algorithm for checking whether all elements in a given array are distinct.

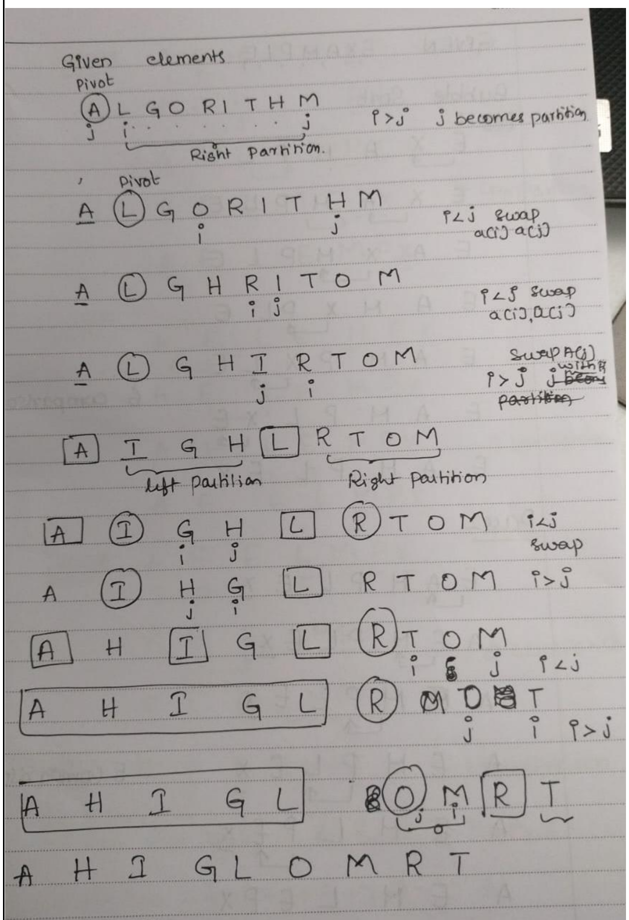
SCHEME:

Algorithm – 4 Marks

SOLUTION:

</td>
<td>4</td>
<td>CO2</td>
<td>L2</td>
</tr>
</table>

	ALGORITHM *UniqueElements*($A[0..n-1]$)
	//Determines whether all the elements in a given array are distinct
	//Input: An array $A[0..n-1]$
	//Output: Returns "true" if all the elements in A are distinct
	// and "false" otherwise
	for $i \leftarrow 0$ **to** $n-2$ **do**
	for $j \leftarrow i+1$ **to** $n-1$ **do**
	if $A[i] = A[j]$ **return false**
	return true			

5(a)	Sort the following keyword "ALGORITHM" by applying Quick Sort technique. **SCHEME:** Solving the problem – 5 Marks **SOLUTION:** 	5	CO2	L2

Given elements
Pivot
A L G O R I T H M i > j j becomes partition
Right partition
Pivot
A L G O R I T H M i < j swap a[i] a[j]
A L G H R I T O M i < j swap a[i], a[j]
A L G H I R T O M swap A(j) with pivot i > j
A I G H L R T O M
Left partition Right partition
A I G H L R T O M i < j swap
A I H G L R T O M i > j
A H I G L R T O M i < j
A H I G L R M O T i > j
A H I G L O M R T
A H I G L O M R T

5(b)	Give recursive algorithm to solve towers of Hanoi problem. Show that the efficiency of this algorithm is exponential.
	
	SCHEME:
	Algorithm – 2 Marks
	Recurrence Relation – 1 Mark
	Solving and finding the efficiency – 2 Marks
	
	SOLUTION:
	Ex2: Tower of Hanoi
	ALGORITHM: Tower of Hanoi(n)
	// Disks on one peg is moved to thi
	Peg with the help of auxilary 2nd
	// Input: N disks on 1st peg
	// output: N disks on 3rd peg.
	if n=1 move disc
	else
	tower of Hanoi (n-1)
	move disc
	tower of Hanoi (n-1)
		5	CO1	L2

Analysis:

1. The number of disks 'n' is the input.
2. The number of moves is written as.

$$M(1) = 1$$

$$M(n) = M(n-1) + 1 + M(n-1) \quad \text{for } n > 1$$

$$= 2M(n-1) + 1$$

3. Solving the recurrence relation by backward substitution.

$$M(n) = 2M(n-1) + 1$$

$$= 2\{2M(n-2)+1] + 1 = 2^2M(n-2) + 2 + 1$$

$$= 2^2[2M(n-3)+1] + 2 + 1$$

$$\to 2^3 M(n-3) + 2^2 + 2 + 1$$

$$= 2^3[2M(n-4)+1] + 2^2 + 2 + 1$$

$$\to 2^4 M(n-4) + 2^3 + 2^2 + 2 + 1.$$

$$M(i) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots 2 + 1$$

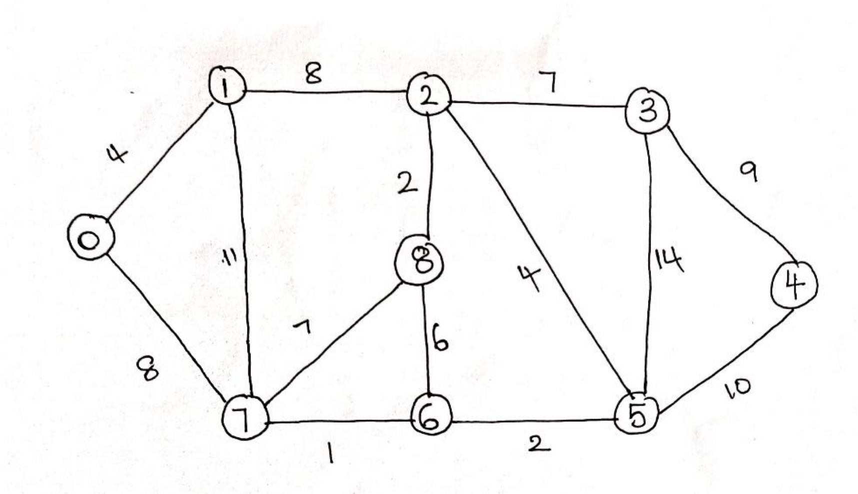
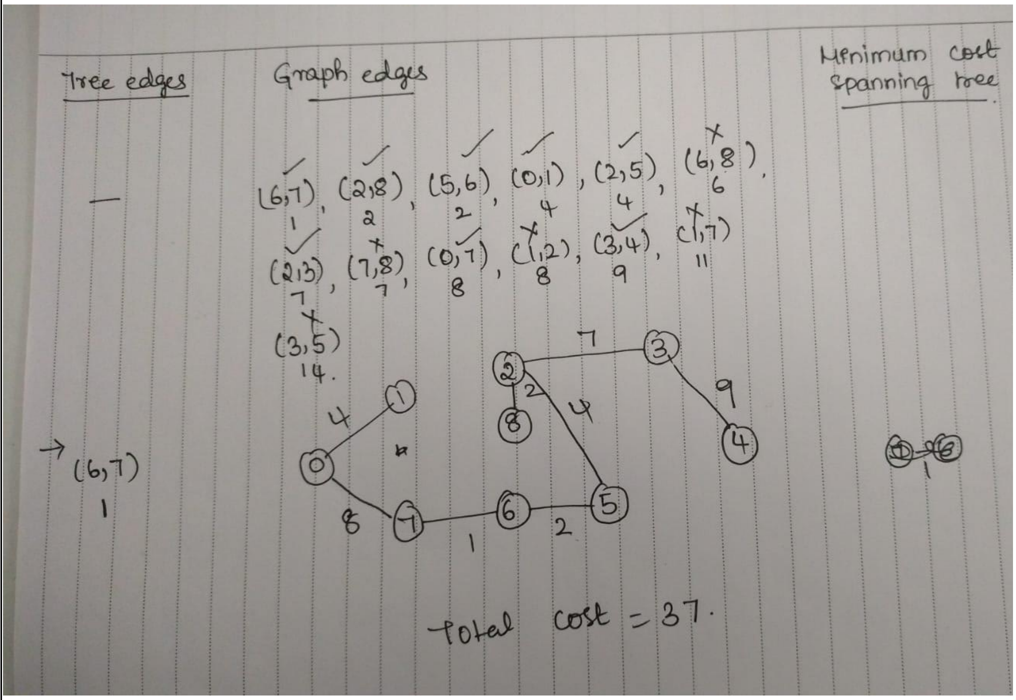
$$M(i) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots 2 + 1$$

$$= 2^i M(n-i) + 2^i - 1$$

$$M(n) = 2^n (M(n-(n-1))) + 2^n - 1$$

$$= 2^n M(1) + 2^{n-1} - 1$$

$$= 2^n - 1.$$

6	Find the Minimum Cost Spanning Tree using Kruskal's Algorithm  **SCHEME:** Solving the problem – 10 Marks **SOLUTION:** 	10	CO4	L3

CI **CCI** **HOD**