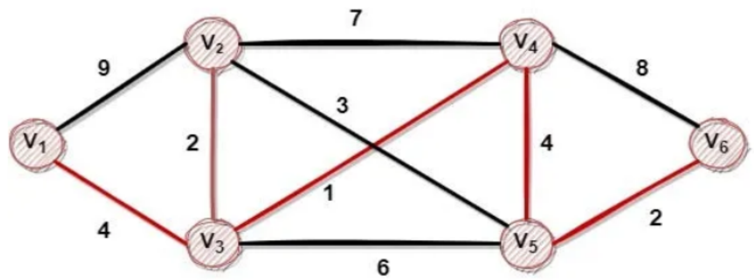
USN | | | | | | | | | | |

Internal Assessment Test 1-March 2025

Sub:	Analysis and Design of Algorithms			Sub Code:	BCS401	Branch:	CSE
Date:	26-03-2025	Duration:	90 mins	Max Marks: 50	Sem / Sec:	IV (A,B & C)	OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1(a)	Apply quicksort on the array with keys [21, 12, 45, 23, 12, 15, 28, 26]. Show the array contents at each level of recursive calls.	5	CO2	L3
1 (b)	Solve the Brute force knapsack problem where sack capacity m=20, n=5, Profits=(30,42,25,18,15), Weights=(6,14,5,2,3).	5	CO1	L3
2 (a)	Explain the Decrease and Conquer with its variations.	3	CO2	L2
2(b)	Illustrate the topological sorting algorithm(DFS method) by taking the example.	7	CO2	L2
3(a)	What are the various Asymptotic efficiency classes.Explain them in detail.	6	CO1	L1
3(b)	What is an Algorithm.Explain the criteria to be satisfied.	4	CO1	L1
4	Anita went to a fruit shop and she wanted to purchase apples based on the size. In Total there were 5 apples. First she picked an apple compared to the second one. She felt the second smallest. Again she started comparing one more apple with the second. She felt the second is the smallest when compared to the third. Next time she compared the second apple with one more, again she felt the second was small. After all the comparison she came to know the second apple is the smallest one she kept in the first place. She repeated the process for the rest of the apples to get apples arranged in the order according to the size. Purchased all 5 apples and happily she came out of the shop. a. Which algorithm can be used and why? b.Write the algorithm? c.Compute the time complexity?	2+4+4	CO1	L3
5	What is the need of Kruskal's algorithm.Explain with an example.Apply the same for the following graph.	10	CO4	L3

6(a)	Consider the following algorithm and derive the time complexity of the following algorithm using analytical framework:
	
	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 $n \times n$ matrices A and B
	//Output: Matrix $C = AB$
	for $i \leftarrow 0$ **to** $n-1$ **do**
	    for $j \leftarrow 0$ **to** $n-1$ **do**
	        $C[i, j] \leftarrow 0.0$
	        for $k \leftarrow 0$ **to** $n-1$ **do**
	            $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$
	return C
		6	CO1	L3

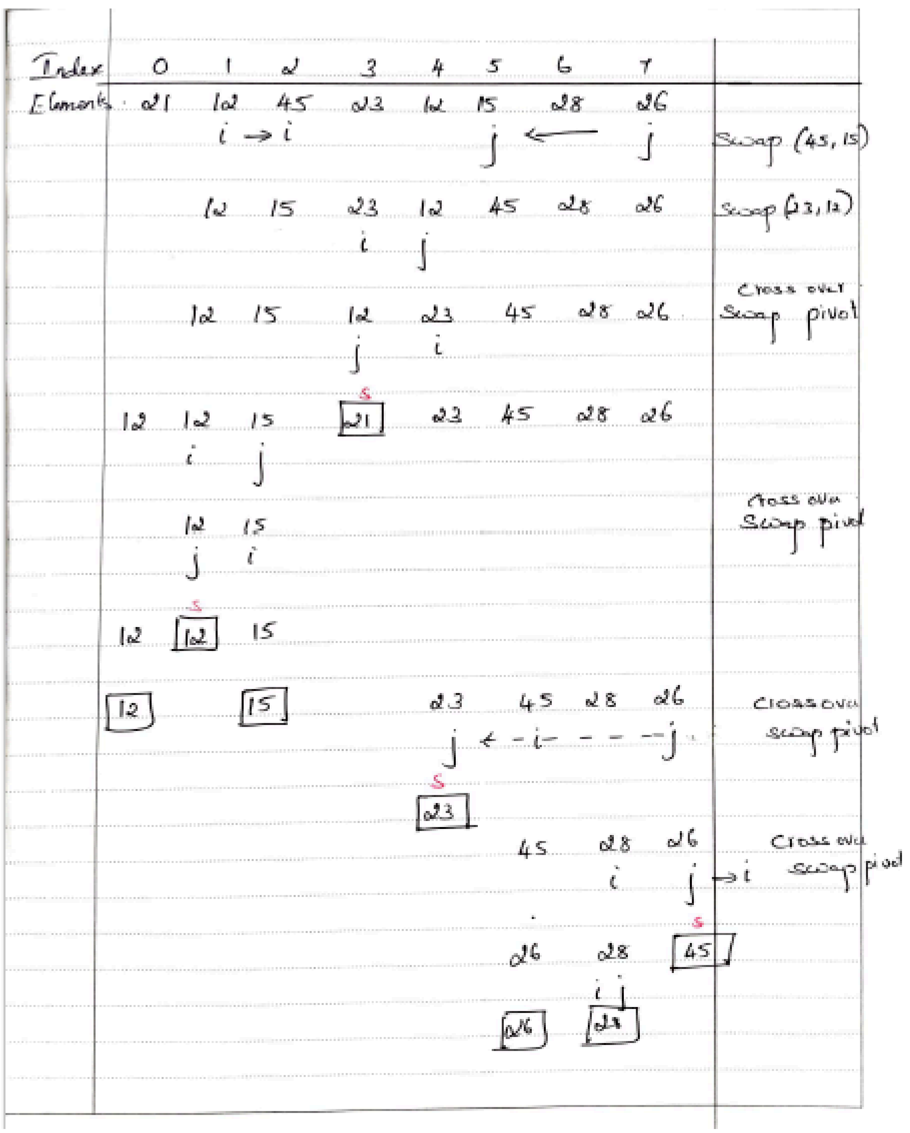
6(b)	**ALGORITHM** *Enigma (A [0… n-1, 0…. n-1])*
	    for i ← 0 **to** n - 2 **do**
	        for j ← i + 1 **to** n – 1 **do**
	            if A [i, j] ≠ A [j, i]
	                return **false**
	            end **for**
	        end **for**
	    return true
	
	Consider the above algorithm and answer the below questions.
	i. What does this algorithm compute?
	ii. Identify the basic operation.
	iii. Calculate how many times the basic operation is executed.
	iv. Derive the efficiency class of this algorithm.
		4	CO1	L3

CI **CCI** **HOD**

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

Internal Assessment Test 1-March 2025

Sub:	Analysis and Design of Algorithms			Sub Code:	BCS401	Branch:	CSE
Date:	26-03-2025	Duration:	90 mins	Max Marks:	50	Sem / Sec:	IV (A,B & C)	OBE

	Solution	MARKS	CO	RBT
1(a)	Apply quicksort on the array with keys [21, 12, 45, 23, 12, 15, 28, 26]. Show the array contents at each level of recursive calls. 	5	CO2	L3

<table>
<tr>
<td>1 (b)</td>
<td>Solve the Brute force knapsack problem where sack capacity m=20, n=5, Profits=(30,42,25,18,15), Weights=(6,14,5,2,3).
<table>
<tr><th>Items</th><th>Weight</th><th>Profit</th><th>Valid?</th></tr>
<tr><td>None</td><td>0</td><td>0</td><td>✅</td></tr>
<tr><td>{5}</td><td>3</td><td>15</td><td>✅</td></tr>
<tr><td>{4}</td><td>2</td><td>18</td><td>✅</td></tr>
<tr><td>{4,5}</td><td>5</td><td>33</td><td>✅</td></tr>
<tr><td>{3}</td><td>5</td><td>25</td><td>✅</td></tr>
<tr><td>{3,5}</td><td>8</td><td>40</td><td>✅</td></tr>
<tr><td>{3,4}</td><td>7</td><td>43</td><td>✅</td></tr>
<tr><td>{3,4,5}</td><td>10</td><td>58</td><td>✅</td></tr>
<tr><td>{2}</td><td>14</td><td>42</td><td>✅</td></tr>
<tr><td>{2,5}</td><td>17</td><td>57</td><td>✅</td></tr>
<tr><td>{2,4}</td><td>16</td><td>60</td><td>✅</td></tr>
<tr><td>{2,4,5}</td><td>19</td><td>75</td><td>✅</td></tr>
<tr><td>{2,3}</td><td>19</td><td>67</td><td>✅</td></tr>
<tr><td>{2,3,5}</td><td>22</td><td>82</td><td>❌ (exceeds 20)</td></tr>
<tr><td>{2,3,4}</td><td>21</td><td>85</td><td>❌ (exceeds 20)</td></tr>
<tr><td>{2,3,4,5}</td><td>24</td><td>100</td><td>❌ (exceeds 20)</td></tr>
<tr><td>{1}</td><td>6</td><td>30</td><td>✅</td></tr>
<tr><td>{1,5}</td><td>9</td><td>45</td><td>✅</td></tr>
<tr><td>{1,4}</td><td>8</td><td>48</td><td>✅</td></tr>
<tr><td>{1,4,5}</td><td>11</td><td>63</td><td>✅</td></tr>
<tr><td>{1,3}</td><td>11</td><td>55</td><td>✅</td></tr>
<tr><td>{1,3,5}</td><td>14</td><td>70</td><td>✅</td></tr>
<tr><td>{1,3,4}</td><td>13</td><td>73</td><td>✅</td></tr>
<tr><td>{1,3,4,5}</td><td>16</td><td>88</td><td>✅</td></tr>
<tr><td>{1,2}</td><td>20</td><td>72</td><td>✅</td></tr>
<tr><td>{1,2,5}</td><td>23</td><td>87</td><td>❌ (exceeds 20)</td></tr>
<tr><td>{1,2,4}</td><td>22</td><td>90</td><td>❌ (exceeds 20)</td></tr>
<tr><td>{1,2,4,5}</td><td>25</td><td>105</td><td>❌ (exceeds 20)</td></tr>
<tr><td>{1,2,3}</td><td>25</td><td>97</td><td>❌ (exceeds 20)</td></tr>
<tr><td>{1,2,3,5}</td><td>28</td><td>112</td><td>❌ (exceeds 20)</td></tr>
<tr><td>{1,2,3,4}</td><td>27</td><td>115</td><td>❌ (exceeds 20)</td></tr>
<tr><td>{1,2,3,4,5}</td><td>30</td><td>130</td><td>❌ (exceeds 20)</td></tr>
</table>

Weight = 16
Profit = 88

Thus, the optimal solution is choosing items {1,3,4,5} with a total profit of 88.</td>
<td>5</td>
<td>CO1</td>
<td>L3</td>
</tr>
<tr>
<td>2 (a)</td>
<td>Explain the Decrease and Conquer with its variations.</td>
<td>3</td>
<td>CO2</td>
<td>L2</td>
</tr>
</table>

•In the decrease-by-a-constant variation, the size of an instance is reduced by the same constant on each iteration of the algorithm. Typically, this constant is equal to one, although other constant size reductions do happen occasionally.

Decrease –by –one and Conquer

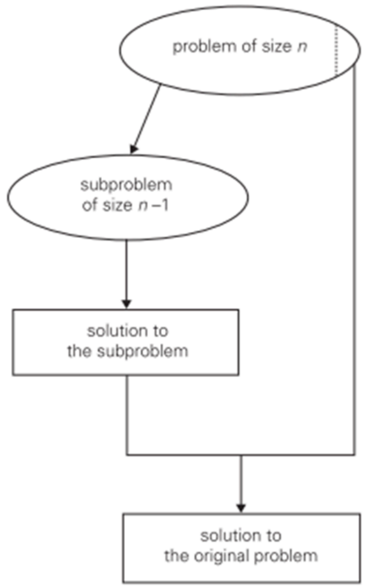
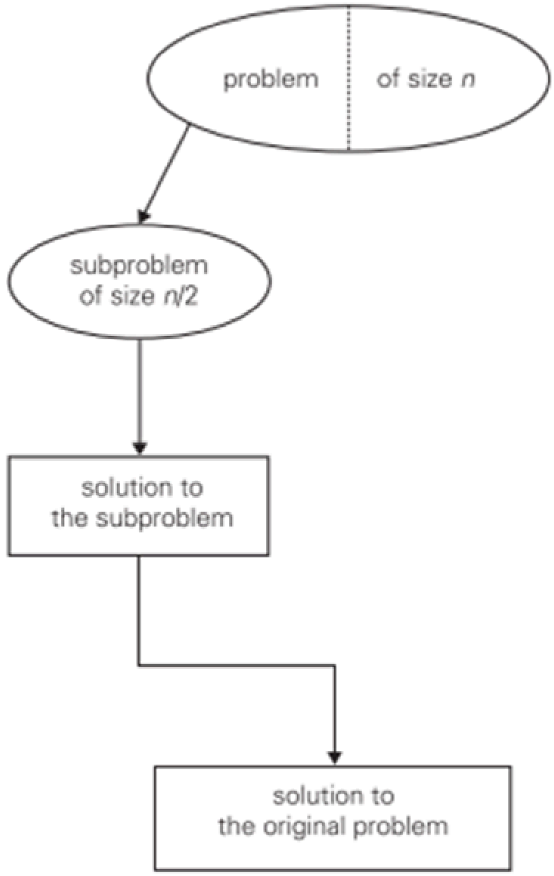


FIGURE 4.1 Decrease-(by one)-and-conquer technique.

In the decrease-by-a-constant variation, the size of an instance is reduced by the same constant on each iteration of the algorithm. Typically, this constant is equal to one, although other constant size reductions do happen occasionally.

Decrease-(by half) and conquer

	 **FIGURE 4.2** Decrease-(by half)-and-conquer technique. variable-size-decrease variety of decrease-and-conquer, the size-reduction pattern varies from one iteration of an algorithm to another.			
2(b)	Illustrate the topological sorting algorithm(DFS method) by taking the example. ● Perform a DFS traversal and note the order in which vertices become dead-ends (i.e., popped off the traversal stack). ● Reversing this order yields a solution to the topological sorting problem, Note: no back edge has been encountered during the traversal. If a back edge has been encountered, the digraph is not a dag, and topological sorting of its vertices is impossible	7	CO2	L2

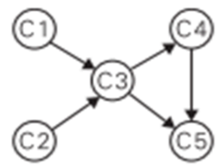


FIGURE 4.6 Digraph representing the prerequisite structure of five courses.

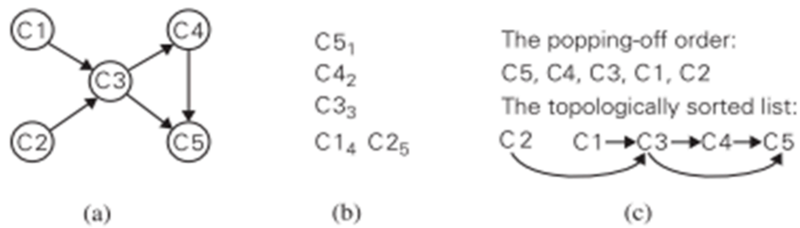


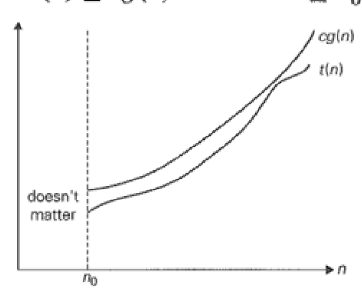
FIGURE 4.7 (a) Digraph for which the topological sorting problem needs to be solved. (b) DFS traversal stack with the subscript numbers indicating the popping-off order. (c) Solution to the problem.

3(a)	What are the various Asymptotic efficiency classes.Explain them in detail.	6	CO1	L1

O-notation

DEFINITION 1 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

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



- **upper bound** of an algorithm's running time.
- Measures the **worst case time complexity** or longest amount of time an algorithm can possibly take to complete

FIGURE 2.1 Big-oh notation: $t(n) \in O(g(n))$

Ω-notation

DEFINITION 2 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

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

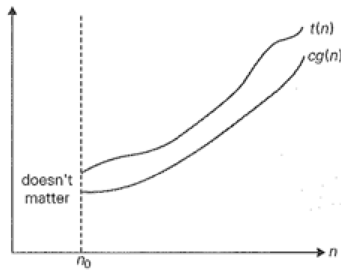


FIGURE 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$

- **<u>lower bound</u>** of an algorithm's running time.
- It measures the **<u>best case time complexity</u>** or best amount of time an algorithm can possibly take to complete

Θ-notation

DEFINITION 3 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 constant c_1 and c_2 and some nonnegative integer n_0 such that

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

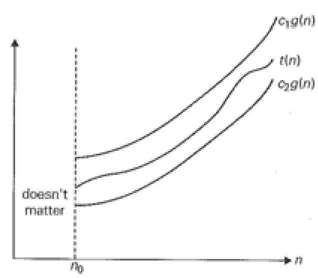


FIGURE 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

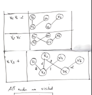
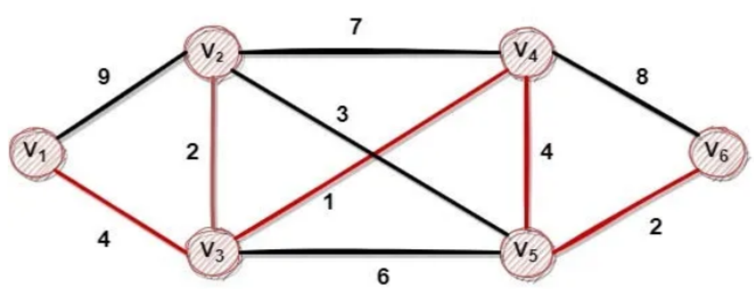
- **<u>express both the lower bound and upper bound</u> of an** algorithm's running time.
- **Average Case**

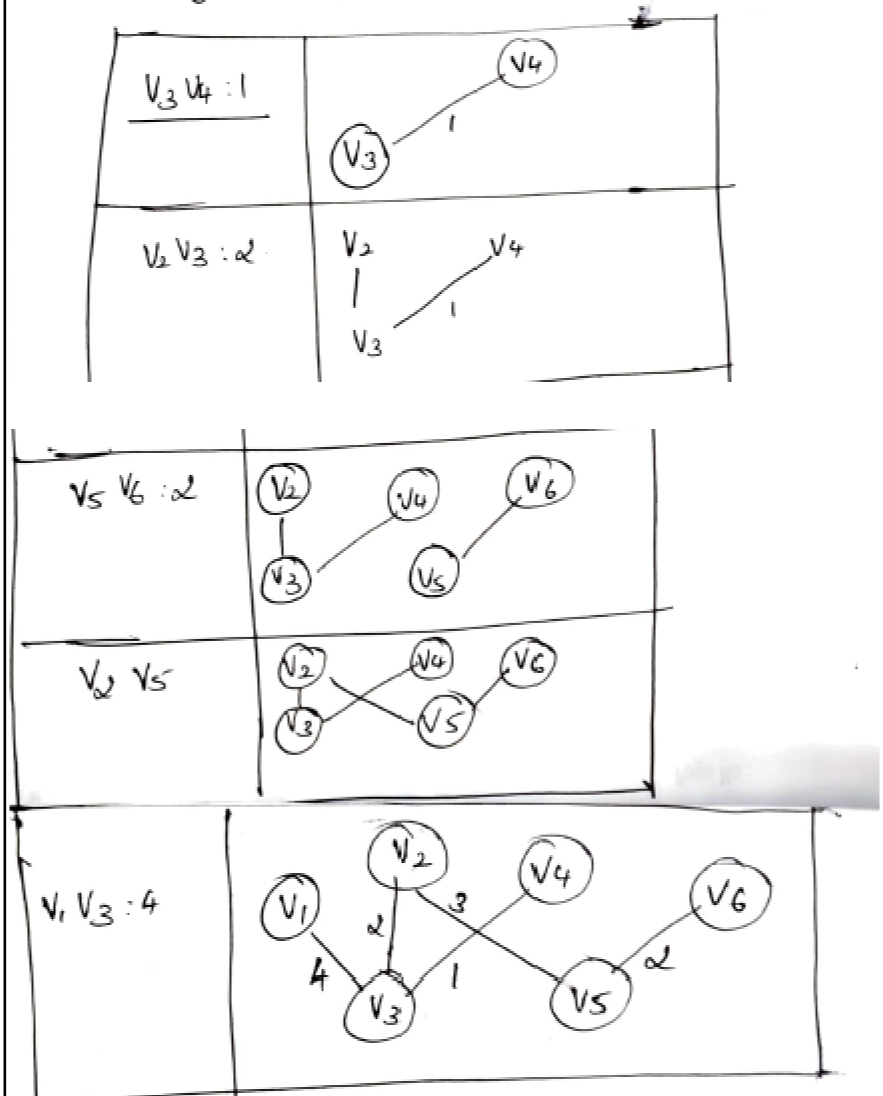
TABLE 2.2 Basic asymptotic efficiency classes

Class	Name	Comments
1	*constant*	Short of best-case efficiencies, very few reasonable examples can be given since an algorithm's running time typically goes to infinity when its input size grows infinitely large.
$\log n$	*logarithmic*	Typically, a result of cutting a problem's size by a constant factor on each iteration of the algorithm (see Section 4.4). Note that a logarithmic algorithm cannot take into account all its input or even a fixed fraction of it: any algorithm that does so will have at least linear running time.
n	*linear*	Algorithms that scan a list of size n (e.g., sequential search) belong to this class.
$n \log n$	*linearithmic*	Many divide-and-conquer algorithms (see Chapter 5), including mergesort and quicksort in the average case, fall into this category.
n^2	*quadratic*	Typically, characterizes efficiency of algorithms with two embedded loops (see the next section). Elementary sorting algorithms and certain operations on $n \times n$ matrices are standard examples.
n^3	*cubic*	Typically, characterizes efficiency of algorithms with three embedded loops (see the next section). Several nontrivial algorithms from linear algebra fall into this class.
2^n	*exponential*	Typical for algorithms that generate all subsets of an n-element set. Often, the term "exponential" is used in a broader sense to include this and larger orders of growth as well.
$n!$	*factorial*	Typical for algorithms that generate all permutations of an n-element set.

3(b)	What is an Algorithm.Explain the criteria to be satisfied. Definition】An algorithm is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria: (1) Input There are zero or more quantities that are externally supplied. (2) Output At least one quantity is produced. (3) Definiteness Each instruction is clear and unambiguous. (4) Finiteness If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after finite number of steps. (5) Effectiveness Every instruction must be basic enough to be carried out, in principle, by a person using only pencil and paper. It is not enough that each operation be definite as in(3); it also must be feasible.	4	CO1	L1
4	Anita went to a fruit shop and she wanted to purchase apples based on the size. In Total there were 5 apples. First she picked an apple compared to the second one. She felt the second smallest. Again she started comparing one more apple with the second. She felt the second is the smallest when compared to the third. Next time she compared the second apple with one more, again she felt the second was small. After all the comparison she came to know the second apple is the smallest one she kept in the first place. She repeated the process for the rest of the apples to get apples arranged in the order according to the size. Purchased all 5 apples and happily she came out of the shop. a. Which algorithm can be used and why? b.Write the algorithm?	2+4+4	CO1	L3

	c.Compute the time complexity?
	Solution:
	a. Selection Sort
	Algorithm SelectionSort(a, n)
	// Sort the array $a[1:n]$ into nondecreasing order.
	{
	for $i := 1$ **to** n **do**
	{
	$j := i$;
	for $k := i+1$ **to** n **do**
	if $(a[k] < a[j])$ **then** $j := k$;
	$t := a[i]$; $a[i] := a[j]$; $a[j] := t$;
	}
	}
	b.
	c. Time Complexity (All Cases): $O(n^2)$			

 5	What is the need of Kruskal's algorithm.Explain with an example.Apply the same for the following graph.  	10	CO4	L3

	Sorted Order: V_3V_4 1 ✓, V_2V_3 2 ✓, V_5V_6 2 ✓, V_2V_5 3, V_1V_3 4, V_4V_5 4 V_3V_5 6, V_2V_4 7, V_4V_6 8, V_1V_2 9  All nodes → visited MST : cost : 12			
6(a)	Consider the following algorithm and derive the time complexity of the following	6	CO1	L3

algorithm using analytical framework:

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 $n \times n$ matrices A and B
//Output: Matrix $C = AB$
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** $n-1$ **do**
    $C[i, j] \leftarrow 0.0$
    for $k \leftarrow 0$ **to** $n-1$ **do**
      $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$
return C

Solution:

$$\sum_{k=0}^{n-1} 1,$$

ınd the total number of multiplications $M(n)$ is expressed by the following riple sum:

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1.$$

Now, we can compute this sum by using formula (S1) and rule (R1) given ıbove. Starting with the innermost sum $\sum_{k=0}^{n-1} 1$, which is equal to n (why?), we get

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3.$$

If we now want to estimate the running time of the algorithm on a particular machine, we can do it by the product

$$T(n) \approx c_m M(n) = c_m n^3,$$

where c_m is the time of one multiplication on the machine in question. We would get a more accurate estimate if we took into account the time spent on the additions, too:

$$T(n) \approx c_m M(n) + c_a A(n) = c_m n^3 + c_a n^3 = (c_m + c_a) n^3,$$

<table>
<tr>
<td>6(b)</td>
<td>

ALGORITHM *Enigma (A [0… n-1, 0…. n-1])*

```
for i ← 0 to n - 2 do
    for j ← i + 1 to n – 1 do
        if A [i, j] ≠ A [j, i]
            return false
        end for
    end for
return true
```

Consider the above algorithm and answer the below questions.
i. What does this algorithm compute?
ii. Identify the basic operation.
iii. Calculate how many times the basic operation is executed.
iv. Derive the efficiency class of this algorithm.

a. The algorithm returns "true" if its input matrix is symmetric and "false" if it is not.

b. Comparison of two matrix elements.

c. $C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1)$
$= \sum_{i=0}^{n-2} (n-1-i) = (n-1) + (n-2) + ... + 1 = \frac{(n-1)n}{2}$.

d. Quadratic: $C_{worst}(n) \in \Theta(n^2)$ (or $C(n) \in O(n^2)$).

</td>
<td>4</td>
<td>CO1</td>
<td>L3</td>
</tr>
</table>

CI **CCI** **HOD**