Semester: 5th

Programme: B. Tech

Branch/Specialization: CSE/IT/CSSE/CSCE

AUTUMN END SEMESTER EXAMINATION-2023

5th Semester, B.Tech (Programme)

DESIGN AND ANALYSIS OF ALGORITHMS

CS 2012

(Evaluation Scheme)

SECTION-A

1. Answer all the questions. [1 x 10 = 10]

Scheme:

Correct answer: 1 Mark

Wrong answer, but explanation approaches to answer: Step Marking

(a) **Find the worst-case time complexity of the given summation.**

$$\sum_{i=1}^{n}\sum_{j=1}^{i} 1$$

Answer: O(n^2)
Explanation: The expression can be coded as follows –

```
for i = 1 to n do
        for j = 1 to i do
                s = s+1
Display s
```

The complexity can be calculated in terms of 1+2+3+…+n which evaluates to O(n^2).

(b) **The binary search algorithm cannot be applied to which of these data structures?**

I. **Array**
II. **Linked list**
III. **Binary search tree**
IV. **Pointer array**

Answer: (ii) Linked list or (iv) Pointer array
Explanation: To apply Binary Search algorithm, the data structure must be sorted and access to any element of the data structure takes constant time.
Binary search cannot be applied on a linked list since it is a sequential and linear data structure, hence does not allow random access. It can't be applied on pointer array because a pointer array is not sorted, and binary search requires the array to be sorted to work effectively.

(c) **What is the division of the array size N for which merge sort will result best execution time?**

I. **1 and N-1**
II. **N/2 and N/2**
III. **N/3 and 2N/3**
IV. **N/4 and 3N/4**

Answer: (ii) n/2 and n/2

Explanation: Execution time will depend on the larger division; thus, the fastest execution will be possible when the array is divided into equal halves. Any other division will give bigger array sizes.

(d) **Assume you have been given tokens of values 25, 10, 5, and 1 unit. What is the minimum number of tokens required to represent 48 units, using a greedy approach? Justify.**

Answer: 6

Explanation: Using a greedy approach, the largest token will be considered first, if it can be feasibly selected. Thus, the order of selection will be 25 + 10 + 10 + 1 + 1 + 1 = 48 units.

(e) **Consider an array A = [89, 19, 50, 17, 12, 15, 2, 5, 7, 11, 6, 9, 100]. What is the minimum number of swaps required to convert this array into a max heap?**

Answer: 3

Explanation: Applying Build_Heap algorithm, the following swaps will be made –
(12, 100) (19, 100) (89,100)
Final array (max-heap) will be [100, 89, 50, 17, 19, 7, 11, 6, 9, 100].
.

(f) **Define the principle of optimality. Which programming technique uses this principle?**

Answer:
The principle of optimality is the core of dynamic programming. It states that to find the optimal solution of the original problem, a solution of each sub problem also must be optimal. It is not possible to derive optimal solution using dynamic programming if the problem does not possess the principle of optimality.

(g) **A spanning tree of a graph has 100 vertices. By how much will the cost of the spanning tree be increased, if the weight of each edge of the graph is increased by 5 units?**

Answer: 495
Explanation: A spanning tree for a graph with 100 vertices will have 99 edges. If the weight of each edge is increased by 5, the total increase in cost will be 99*5 = 495.

(h) **How many comparisons are required to merge two sorted arrays of lengths p and q into a single sorted array?**

I. O(p)

II. O(q)

III. O(pq)

IV. O(p+q)

Answer: (iv) p + q

Explanation: The number of comparisons required to merge two sorted lists is p+q-1. This is because in the worst case, we will have to compare every element in both lists.

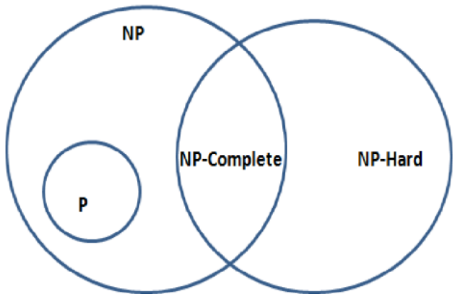
(i) **What is the minimum number of comparisons required to determine if an integer appears more than n/2 times in a sorted array of n integers?**

Answer: θ (log n)

Explanation: A modified binary search can be used to find the first occurrence i of the integer by comparing the current mid element with its previous element. i+(n/2 + 1)th location can be checked to verify if it is present more than n/2 times. Binary search takes place in the range of log n comparisons.

(j) **Represent the relation among P, NP, NP Complete and NP Hard classes of problems using Venn diagram.**

Answer:



SECTION-B

2. [4 x 2 = 8]

(a) A farmer has produced N number of food items in his land. The total expenditure, total selling price for a total weight (Kg) of each item are available. Write an algorithm to sell M kg of items in the market to get maximum profit.

Scheme:

Algorithm or program: 4 Marks

Wrong answer, but explanation approaches to answer: Step Marking

Assumption: Prior to implementing the algorithm, the following computations are performed.
Store the profit or loss of each food item in an array p where **CP** is the cost price and **SP** is the selling price.

```
For each i=1 to n
        p[i] = sp[i] – cp[i]
```

Items are sorted by $\frac{p_i}{w_i}$ in descending order

```
Algorithm  GREEDY_KNAPSACK(W, n)

for i ← 1 to n do
        x[i] = 0.0
//end for
sell = 0
profit=0

i = 1
while (( sell<M) and (i ≤ n)) do
        if( $w_i$ ≤ ( M – sell) ) then
                sell = sell + $w_i$
                x[i] = 1
        else
                r = M – sell
                sell = sell + r
                x[i] = r/$w_i$
        end if
        profit= profit+ p[i]. x[i]
        i = i + 1
//end while
return x and profit
```

(b) Solve the following recurrences-

I. $T(n) = \sqrt{2}\, T\left(\frac{n}{2}\right) + \log n$

II. $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + cn$

Scheme:

I. Correct answer: 2 Marks

II. Correct answer: 2 Marks

Wrong answer, but explanation approaches to answer: Step Marking

i. T(n) = √2 T(n/2) + log n. To solve this recurrence relation using the **Master Theorem**, we need to find the values of a, b, and f(n) in the relation T(n) = aT(n/b) + f(n).

Here, a = √2, b = 2, and f(n) = log n.

$\log_b a = \log_2 \sqrt{2} = 1/2$

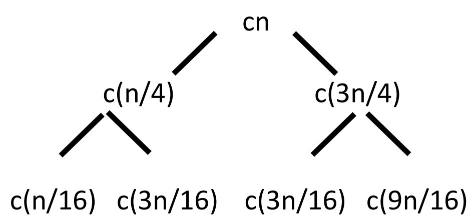
Compare f(n) with n^($\log_b$ a) to determine the time complexity of the recurrence relation. Since f(n) = log n and n^($\log_b$ a) = n^(1/2), we have:

f(n) = O(n^(1/2 - ε)) for ε = 1/2

Therefore, the solution to the recurrence relation T(n) = √2 T(n/2) + log n using the Master Theorem is:

T(n) = Θ(n^(1/2))

The time complexity of the recurrence relation is Θ(n^0.5) = Θ (√n) ii.



Choosing the longest path from root node to leaf node:

$(3/4)^0 n \text{ –> } (3/4)^1 n \text{ –> } (3/4)^2 n \text{ –> } \ldots\ldots\ldots \text{ –> } (3/4)^k n$

Size of problem at last level = $(3/4)^k n$

At last level size of problem becomes 1

$(3/4)^k n = 1$

$(3/4)^k = 1/n$

$k = \log_{4/3}(n)$

Total no of levels in the recursion tree = $k + 1 = \log_{4/3}(n) + 1$

Then $T(n) \leq n \log_{4/3}(n)$

- ⇨ **$T(n) = O(n \log_{4/3} n)$ which can also be expressed as O (nlgn)**

3. [4 x 2 = 8]

(a) Professors of the School of Computer Engineering decided to include cine-stars (Mr Bachan, Mr Sarukh, Mrs Madhuri and Mrs Kartina) in their Ramp Show to make the show more attractive. The student groups have given their event name, start and end time for each event to be performed in the KIIT Fest-24 in one stage on the first day. Write an algorithm to schedule the student events to obtain the followings:

- **i. Maximum number of students' events can be shown.**
- **ii. Start time and end time to accommodate the professors Ramp show headed by Prof Das in the largest free period (no student event occurs), if available between two consecutive student events in the same stage on first day.**

Scheme:

Calculation of Maximum number of events: 2 Marks

Calculation of largest free period: 2 Marks

Wrong answer, but explanation approaches to answer: Step Marking

Activities ai and aj are compatible if the intervals [si, fi) and [sj, fj) are non-overlapping.
i.e ai and aj are compatible if si ≥ fj or sj ≥ fi.

Approach: (Assume: activities are sorted according to their finish time)

Step1: a_1 will be selected first as a_k .

Step 2: Find out the first activity that starts after a_k finishes, add that to the solution set.

Step 3: Continue step 2 till all the activities in the list are exhausted.

Algorithm: GREEDY-ACTIVITY-SELECTOR (s, f)

```
n = length(s)
A = {a1}
k = 1
for m=2 to n
    if s[m] ≥ f[k]
        A = A U {$a_m$}
        k = m
return length(A)
```

Algorithm MAX-FREE-SLOT(A, s, f)

```
1.  n = length(A)
2.  free_start = 0
3.  free_end = 0
4.  max_free = 0
5.  for i=1 to n
6.      j  = A[i]
7.      k = A[i+1]
8.      if s[j] - f[k] > max_free
9.          free_start = f[j]
10.         free_end = s[k]
11.         max_free  = s[j] - f[k]
12.     //end if
13. //end for
14. Display max free slot [free_start, free_end)
```

(b) Mathematically represent the various asymptotic notations used for time complexity analysis. Arrange the following functions in terms of increasing asymptotic complexity.
10, √n, n, $\log_2 n$, 100 / n

Scheme:

Representation of at least 3 notations: 3 Marks

Arrangement in increasing order: 1 Mark

Wrong answer, but explanation approaches to answer: Step Marking

Asymptotic notations are mathematical tools used to represent the time complexity of algorithms for asymptotic analysis.
It compares two algorithms based on changes in their performance as the input size is increased or decreased. Three asymptotic notations are represented below:

1. **Big-O Notation (O-notation)**: Big-O notation specifies the upper bound of the running time of an algorithm. It represents the worst-case complexity of an algorithm.
   Let f(n) and g(n) be two functions defined on the set of natural numbers. We say that f(n) is O(g(n)) if there exist positive constants c and n0 such that $0 \le f(n) \le c * g(n)$ for all $n \ge n0$.
2. **Omega Notation (Ω-notation)**: Omega notation specifies the lower bound of the running time of an algorithm. It represents the best-case complexity of an algorithm.
   Let f(n) and g(n) be two functions defined on the set of natural numbers. We say that f(n) is Ω(g(n)) if there exist positive constants c and n0 such that $0 \le c * g(n) \le f(n)$ for all $n \ge n0$.
3. **Theta Notation (Θ-notation)**: Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analysing the average-case complexity of an algorithm.
   Let f(n) and g(n) be two functions defined on the set of natural numbers. We say that f(n) is Θ(g(n)) if there exist positive constants c1, c2, and n0 such that $c1 * g(n) \le f(n) \le c2 * g(n)$ for all $n \ge n0$.

The increasing order is: **100/n, 10, $\log_2$n, √n, n**

SECTION-C

4. [4 x 2 = 8]

(a) Write a program to find out the sum of the elements present in each of the upper diagonals of a matrix of size N x N and store each diagonal sum in an array of size N. Mention it's execution time.

Scheme:

Program: 3 Marks

Time complexity: 1 Mark

Wrong answer, but explanation approaches to answer: Step Marking

```
#include <stdio.h>
int main(){
    int n,i,j,k;
    printf("Enter the size of the square matrix: ");
    scanf("%d",&n);
    int matrix[n][n];
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            printf("%d %d element: ",i,j);
            scanf("%d",&matrix[i][j]);
        }
    }
    int sum[n-1];
           int index=0;
    for (k=n;k>=0;k--){
        int i=n-k;
        int j=0;
        int temp=k;
        int diagonalSum=0;
        while (temp>0){
            diagonalSum+=matrix[j][i];
            i++;
            j++;
            temp--;
        }
        sum[index]=diagonalSum;
        index++;
    }
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            printf("%d\t",matrix[i][j]);
        }
        printf("\n");
    }
    for(i=0;i<=n-1;i++){
        printf("Sum of upper diagonal %d: %d\n",i+1,sum[i]);
    }
}
```

The time complexity of this program is O(n^2)

Note: Similarly, one can also find the sum of the upper left diagonals.

(b) Generate a Huffman coding scheme for a file containing the following characters and their given frequencies. Show the total number of bits required for encoding a file of 60,000 bits using the generated code.

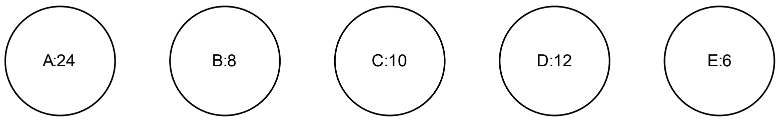
Character	**A**	**B**	**C**	**D**	**E**
Frequency	**24**	**8**	**10**	**12**	**6**

Scheme:

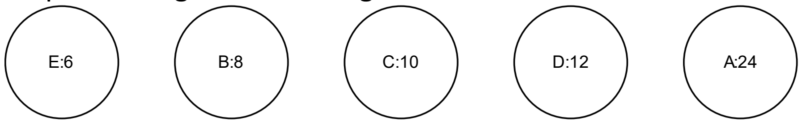
Huffman tree and code generation: 3 Marks

Calculation of total number of bits or total characters: 1 Mark

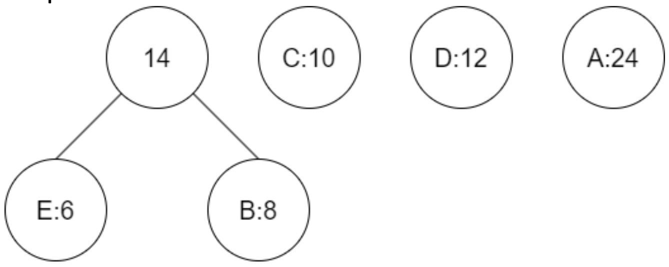
Wrong answer, but explanation approaches to answer: Step Marking



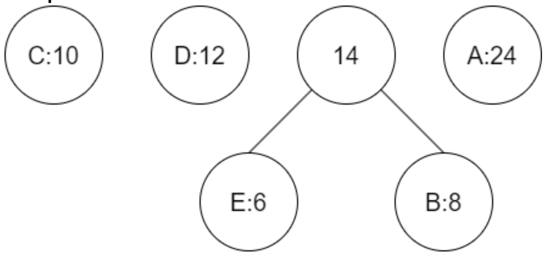
Step 1: Arrange in ascending order.



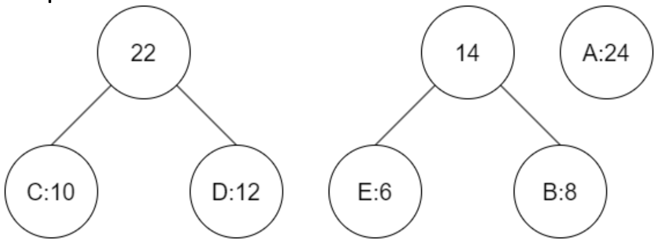
Step 2:



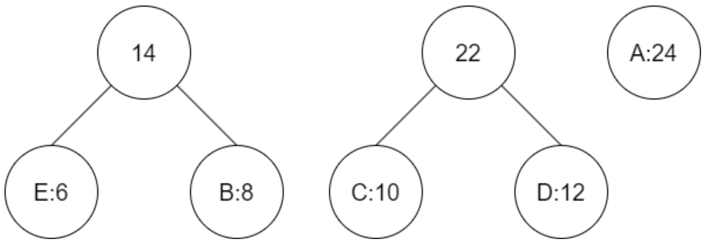
Step 3:



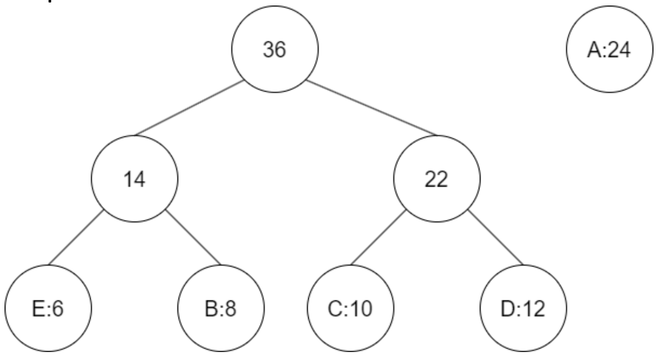
Step 4:



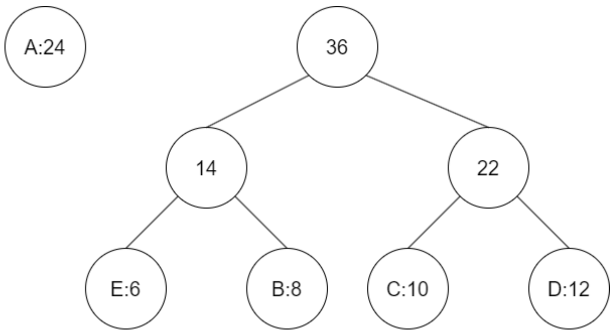
Step 5:



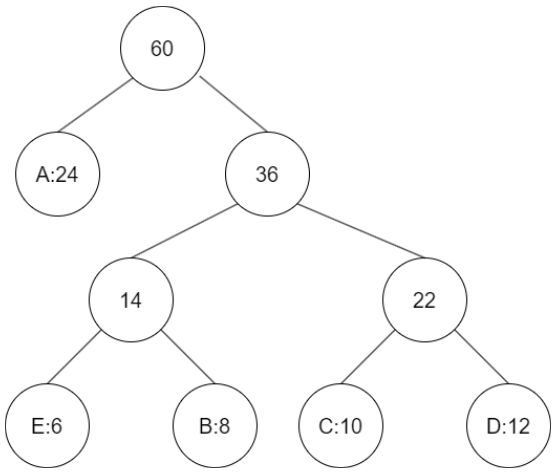
Step 5:



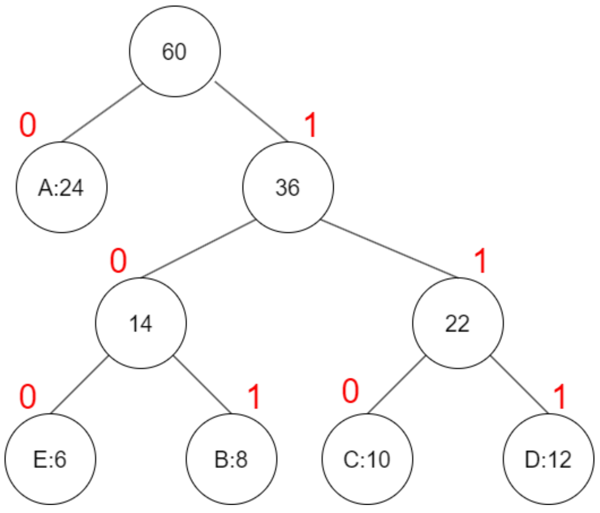
Step 6:



Step 7:



Step 8:



Character	A	B	C	D	E
Frequency	24	8	10	12	6
Code	0	101	110	111	100
Code Length	1	3	3	3	3

Average Code Length=((1*24)+(8*3)+(10*3)+(12*3)+(6*3))/(24+8+10+12+6)
= 2.2

Total number of bits required for encoding a file of 60,000 chars=

2.2*60,000=1,32,000 bits

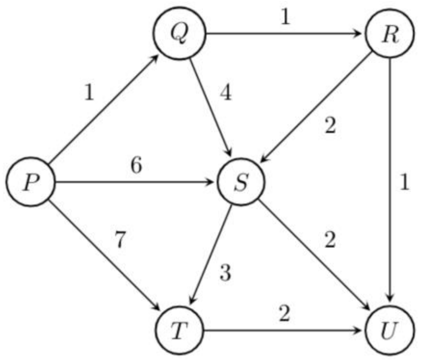
Note# As bits are mentioned in both side of the conversion. One can interpret the solution as below.

Avg. number of chars required for encoded file size of 60,000 bits=

60,000/2.2 ≈ 27273 chars

5. [4 x 2 = 8]

(a) Use a shortest path algorithm to calculate the shortest paths from vertex P to all other vertices in the given graph. Show the data structure when vertex U is added to the shortest path.

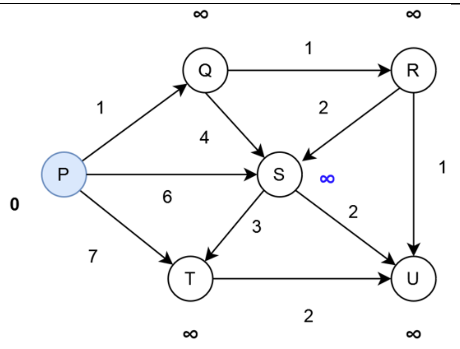


Scheme:

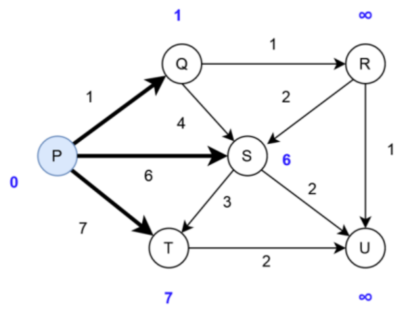
The calculation of the shortest path: 3 Marks

Showing of data structure: 1 Mark

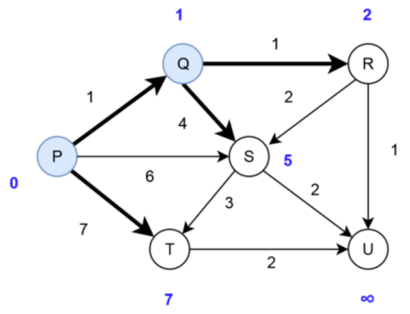
Wrong answer, but explanation approaches to answer: Step Marking



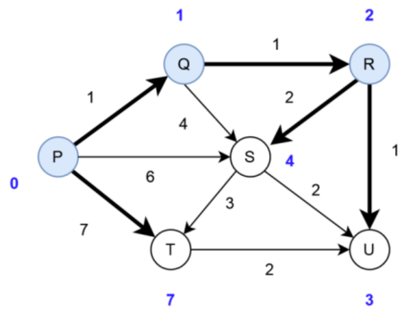
Vertex	P	Q	R	S	T	U
d	0	∞	∞	∞	∞	∞



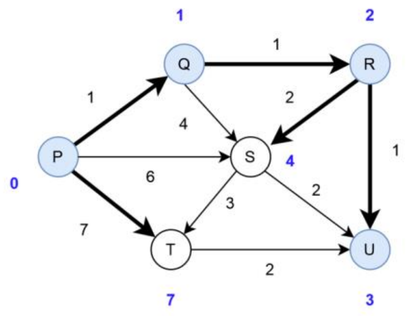
Vertex	Q	R	S	T	U
d	1	∞	6	7	∞



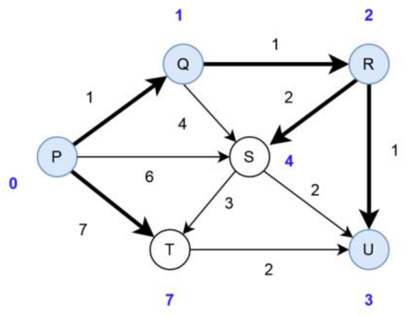
Vertex	R	S	T	U
d	2	4	7	∞



Vertex	S	T	U
d	4	7	3

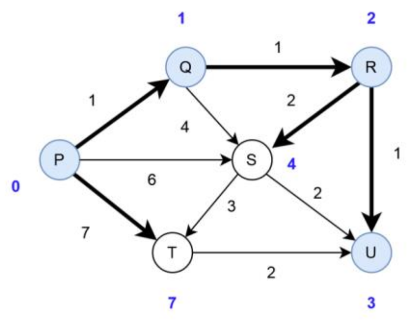


Vertex	S	T
d	4	7



Vertex	T
d	7

Min-priority queue is shaded in the above figure, when vertex U is added to the shortest path.



(b) **Dr. A Samanta, Founder (KIIT and KISS) wants pair of students with highest(H) and lowest (L) CGPA to sit together in the class for improvement the lowest CGPA student. The CGPAs of the N students are stored in Heap. Write an algorithm to find out the roll numbers of the pairs (H, L) of students with minimum number of comparisons. What is the exact number of comparisons.**

Scheme:

Algorithm or Program: 3 Marks

No. of comparisons: 1 Mark

Wrong answer, but explanation approaches to answer: Step Marking

Solution 1: [for only 1st pair]

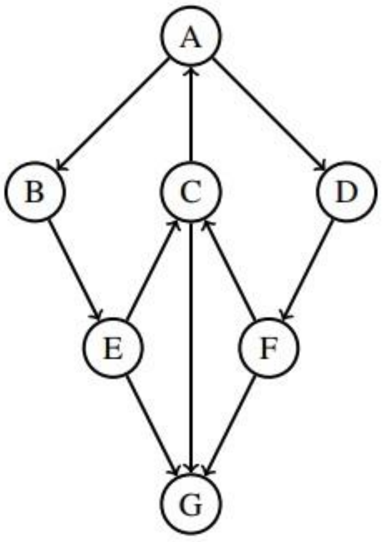
```
/*Sample structure
struct STUDENT
{
    int roll;
    char name[25];
    float CGPA;
} ;

/*Function to return  roll number with highest  CGPA followed by roll numbers with lowest CGPA
from an MAX heap with array of structures A */
EXTRACT-HIGHEST-LOWEST-CGPA (A, n)
{
   H_CGPA_Roll ← GET-MAX-FROM-MAX-HEAP(A, n)
   L_CGPA_Roll ← GET-MIN-FROM-MAX-HEAP(A, n)
   return (H_CGPA_Roll, L_CGPA_Roll)
}

/*Function to return roll number with highest  CGPA  from a  MAX heap with array of structures A */
GET-MAX-FROM-MAX-HEAP(A, n)
{
     rerurn A[1].roll
}
```

```
/*Function to return roll number with lowest CGPA  from a MAX heap with array of structures A */
GET-MIN-FROM-MAX-HEAP(A, n)
{
   min_CGPA ← A[n/2+1].CGPA
   min_roll ← A[n/2+1].roll
   for i  ←  n/2+2 to n
   {
      if (A[i].CGPA<min_CGPA)
      {
           min_CGPA ← A[i].CGPA
            min_roll ← A[i].roll
   }
   return min_roll
}
```

Minimum number of comparisons required for 1 pair= n/2

6. [4 x 2 = 8]

(a) Show the order of traversal of nodes in the given graph using breadth-first search and depth-first search, considering A as the starting vertex. Which technique is more likely to reach the vertex F sooner, considering best case? Justify.



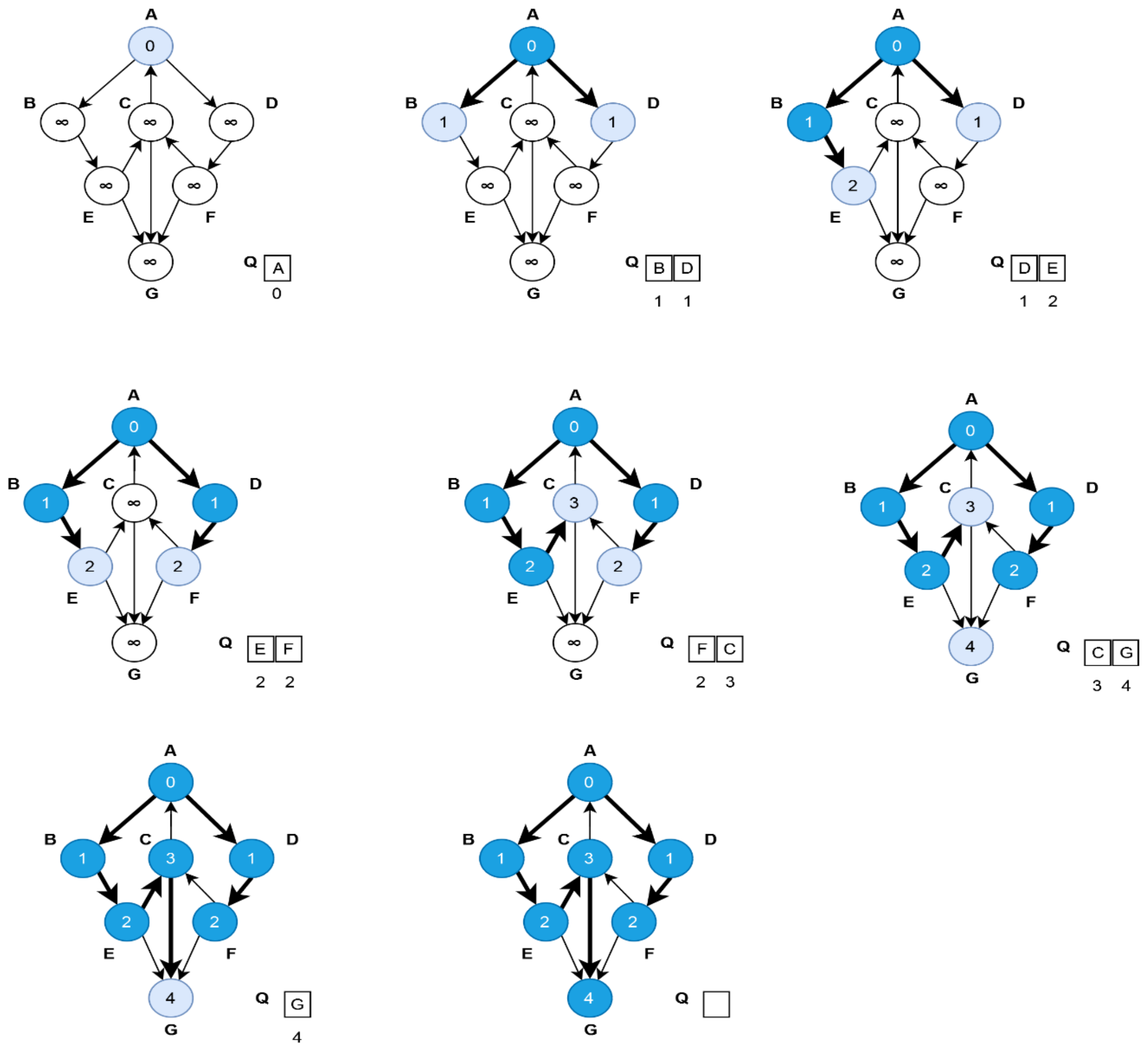
Scheme:

BFS traversal: 1.5 Marks

DFS traversal: 1.5 Marks

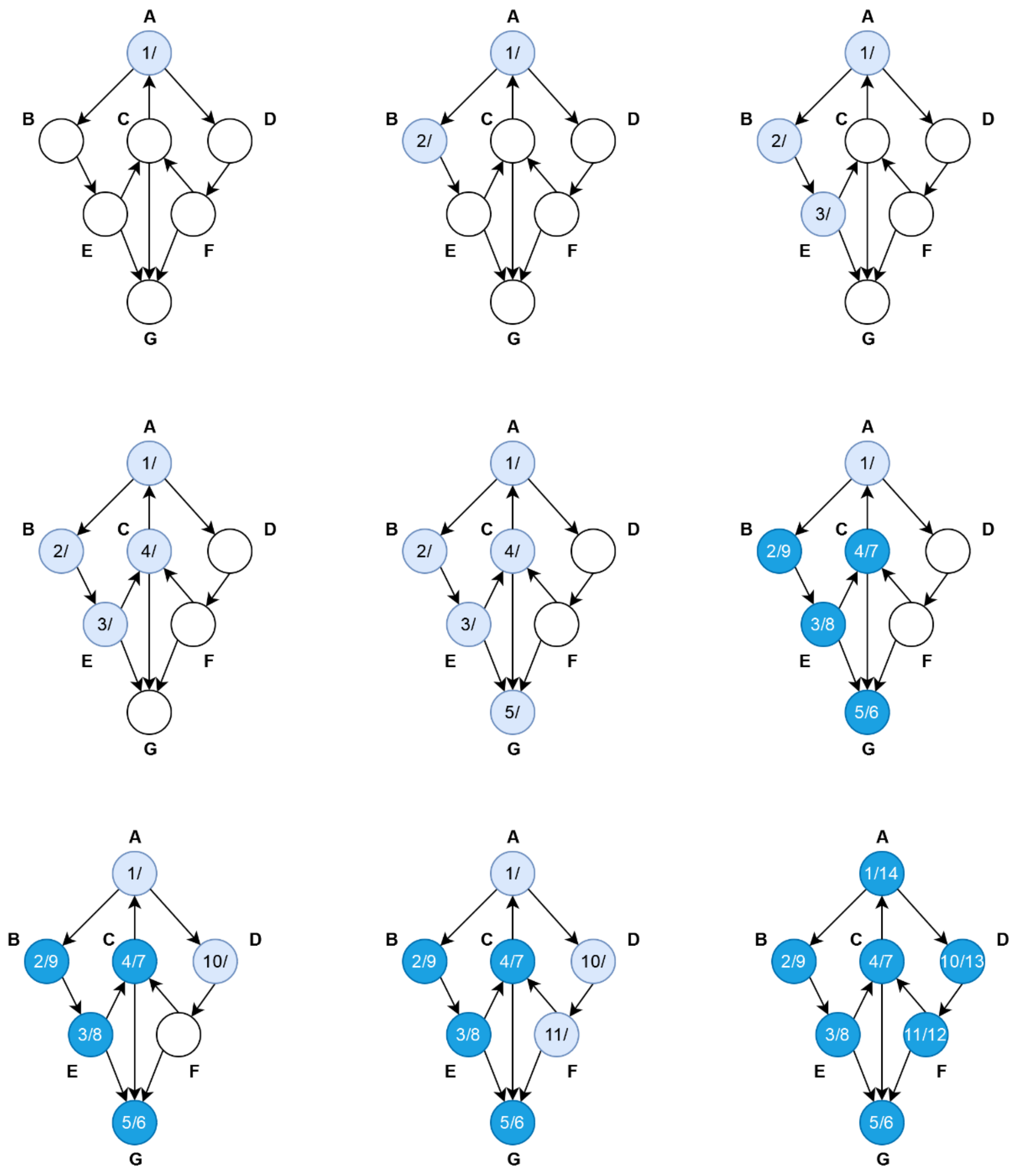
Justification: 1 Mark

Wrong answer, but explanation approaches to answer: Step Marking



BFS order: A B D E **F** C G
The above illustration is for above ordering.

Other possible orders are:
A D B **F** E C G
A D B E **F** C G
A B D **F** E C G



DFS order: A B E C G D **F**

Other possible traversal: A D **F** C G B E, A D **F** G C B E, A B E G C D **F**

Considering the best case, DFS will reach vertex F sooner. However, this is not always the case, as the order in which the nodes are explored depends on the specific implementation of the algorithm and the structure of the graph.

(b) Write an algorithm to determine the frequency of the character ‘T’ present in the longest common sub-sequence between a pair of sequences of sizes ‘n’ each with alphabets A, T, C, G.

Scheme:

Algorithm or Program: 4 Marks

Wrong answer, but explanation approaches to answer: Step Marking

```
LCS_T_frequency(char S1, char S2, int n)
{            int LCS[n+1][n+1], i,j,k=0;
             char S3[n];
             int length, count=0;
             for(i=0;i<=n;i++)
             {            for(j=0;j<=n;j++)
                          {            if(i==0|| j==0)
                                       LCS[i][j]=0;
                                       else if(S1[i-1]==S2[j-1])
                                       {            LCS[i][j]=LCS[i-1][j-1]+1;
                                                    S3[k]=S1[i];
                                                    k++;
                                       }
                                       else
                                                    LCS[i][j]=max(LCS[i-1][j], LCS[i][j-1]);
                          }
             }
             length=LCS[n][n];
             for(i=0;i<length;i++)
             {            if(S3[i]==’T’)
                                       count++;
             }
             return count;
}
```

SECTION-D

7. [4 x 2 = 8]

(a) Explain different ways of handling NP Complete problems. Show the working of insertion sort on the given array A = [5, 3, 1, 8, 4, 2].

.

Scheme:

2 marks for describing the ways to deal with an NP-Complete problem and 2 marks for illustrating the working of Insertion Sort. Full marks in first part may be awarded for describing at least two ways of dealing.

NP-complete problems are some of the most challenging computational problems, and there are several ways to handle them. Here are some common approaches:

1. Approximation algorithms: These algorithms provide an approximate solution to the problem that is guaranteed to be within a certain factor of the optimal solution.
   For example, the traveling salesman problem can be approximated.
2. Heuristics: These are problem-solving techniques that use practical experience and common sense to find a solution that is not guaranteed to be optimal but is often good enough.
   For example, the knapsack problem can be solved using a greedy heuristic that selects items in decreasing order of their value-to-weight ratio.
3. Metaheuristics: These are higher-level problem-solving techniques that can be used to find good solutions to a wide range of problems.
   Examples include simulated annealing, genetic algorithms, and ant colony optimization.
4. Divide and conquer: This approach involves breaking the problem down into smaller subproblems that can be solved independently and then combining the solutions to obtain the final solution. For example, the graph coloring problem can be solved using a divide-and-conquer algorithm that partitions the graph into smaller subgraphs and then colors each subgraph independently.
5. Parameterized complexity: This approach involves analyzing the problem in terms of one or more parameters and then designing algorithms that are efficient with respect to these parameters.
   For example, the vertex cover problem can be solved using a parameterized algorithm that depends on the size of the vertex cover.

The working of Insertion Sort on A = [5, 3, 1, 8, 4, 2]: We explore the proper position for every element of the array starting from 2nd element and insert it in that position as follows.

Iteration 1: Insert 2nd element 3 into the sorted 1-element sub-array [5] of A in its proper position to get an sorted 2-element sub-array of A at the beginning: **[3, 5 | 1, 8, 4, 2]**
Iteration 2: Insert 3rd element 1 into the sorted 2-element sub-array [3, 5] of A in its proper position to get an sorted 3-element sub-array of A at the beginning: **[1, 3, 5 | 8, 4, 2]**
Iteration 3: Insert 4th element 8 into the sorted 3-element sub-array [1, 3, 5] of A in its proper position to get an sorted 4-element sub-array of A at the beginning: **[1, 3, 5, 8 | 4, 2]**
Iteration 4: Insert 5th element 4 into the sorted 4-element sub-array [1, 3, 5, 8] of A in its proper position to get an sorted 5-element sub-array of A at the beginning: **[1, 3, 4, 5, 8 | 2]**
Iteration 5: Insert 6th element 2 into the sorted 5-element sub-array [1, 3, 5, 8] of A in its proper position to get an sorted 6-element sub-array of A at the beginning: **[1, 2, 3, 4, 5, 8 |]**

Since, we explored for the last element of the array, the termination of the algorithm has been reached and we have the Sorted array for A: [1, 2, 3, 4, 5, 8]

(b) **Let S be an NP-complete problem and Q and R be two other problems not known to be in NP. Q is polynomial-time reducible to S and S is polynomial-time reducible to R. What are the probable problem classes of Q and R? Justify.**

Scheme:

2 marks for classification of Q and 2 marks for classification of R. Further, in either part, 1-mark for correct classification and 1 mark for suitable justification logic for the same.

Answer:

Since Q is polynomial-time reducible to S, Q is at least as hard as S.

Since S is NP-complete, Q is also NP-complete.

Similarly, since S is polynomial-time reducible to R, S is at least as hard as R. Since S is NP-complete, R is also NP-hard.

Therefore, the probable problem class of Q is NP-complete or harder, and the probable problem class of R is NP-hard or harder.

It is given further that the problem S is polynomial-time reducible to problem R. That means that, there exists a polynomial-time algorithm f that transforms any instance I of problem S into an instance f(I) of problem R and another polynomial-time algorithm h that maps any solution A of f(I) back into a solution h(A) of I.

If possible, let us assume that R is in P, i.e., R has an algorithm that can determine in polynomial-time whether an Instant J is a solution of R or not. Then serially applying that polynomial time algorithm with polynomial time algorithms f and h we can determine in polynomial time whether an instance I is a solution to problem S or not. In other words, S would have a polynomial time algorithm, i.e., the problem S is in P, A contradiction! So, the problem R must be in NP --- (1)

Also, problem S being NP-Complete, any problem in NP is polynomial reducible to problem S. Now, combining this with polynomial reduction from problem S to problem R, we have polynomial reduction of any problem in NP to problem R. So, problem R is NP-hard --- (2)

Combining conclusions (1) and (2), **we can conclude that problem R is NP-Complete, too.**

Let us now explore the case of problem Q. It is given that the problem Q is polynomial time reducible to problem S, but it is not known whether Q is in NP or not.

Problem S being NP-Complete, any problem in complexity class NP is polynomial-time reducible to problem S. Further, complexity class P being a subset of NP, every problem in class P is polynomial-time reducible to problem S, too. So, we can not decide for sure the membership of problem Q from the given fact that problem Q is polynomial time reducible to problem S. **So, Problem Q can be in P class, or in NP-hard class, or in NP-Complete class.**

8. [4 x 2 = 8]

(a) **Define an NP-complete problem with examples. How can it be proven or disproven that P=NP?**
Suppose we can show that some known NP-complete problem has a polynomial time complexity solution. What will it imply and why?

Scheme:

Definition and Example: 2 Marks

Proving or Disproving: 1 Mark

Justification: 1 Mark

Wrong answer, but explanation approaches to answer: Step Marking

An NP-complete problem is a decision problem that belongs to the set of NP problems and is at least as hard as the hardest problems in NP.
In other words, if we can solve an NP-complete problem in polynomial time, then we can solve any problem in NP in polynomial time.

Some common examples of NP-complete problems:

- Boolean satisfiability problem,
- Clique Decision Problem,
- knapsack problem,
- Hamiltonian path problem,
- traveling salesman problem,
- subset sum problem,
- vertex cover problem
- graph coloring problem.

Proving or disproving P=NP:

It is one of the most important open problems in computer science.

If P=NP, it would mean that every problem in NP can be solved in polynomial time, which would have profound implications for cryptography, optimization, and many other fields.

However, no one has been able to prove or disprove this conjecture yet.

If we can show that some known NP-complete problem has a polynomial time complexity solution, it would imply that P=NP.

This is because all NP problems can be reduced to NP-complete problems in polynomial time, and if we can solve an NP-complete problem in polynomial time, then we can solve any problem in NP in polynomial time.

(b) Consider an array A of integers. Build an algorithm within a complexity of O(n $\log_2$ n), which finds all pairs (x,y), such that x+y = z.
x and y are elements present in A, and z is a value input by the user.

Scheme:

Algorithm or Program: 4 Marks

Wrong answer, but explanation approaches to answer: Step Marking

Algorithm to find all pairs (x,y) in an array A of integers such that x+y = z, with a complexity of O(nlogn):

1. Sort the array A in ascending order using a sorting algorithm with a time complexity of O(nlogn).
2. Initialize two pointers, **left** and **right**, to the first and last elements of the array, respectively.
3. While **left** is less than **right**, do the following:
    - If the sum of the elements at indices **left** and **right** is equal to z, print the pair of elements at indices **left** and **right**.
    - If the sum is greater than z, decrement **right** by 1.
    - If the sum is less than z, increment **left** by 1.

Here's the pseudocode for the algorithm:

```
sort(A)
left = 0
right = length(A) - 1

while left < right{
          if A[left] + A[right] == z{
                print (A[left], A[right])
                left = left + 1
                right = right – 1
          }
          elseif A[left] + A[right] < z
                left = left + 1
          else
                right = right - 1
}
```