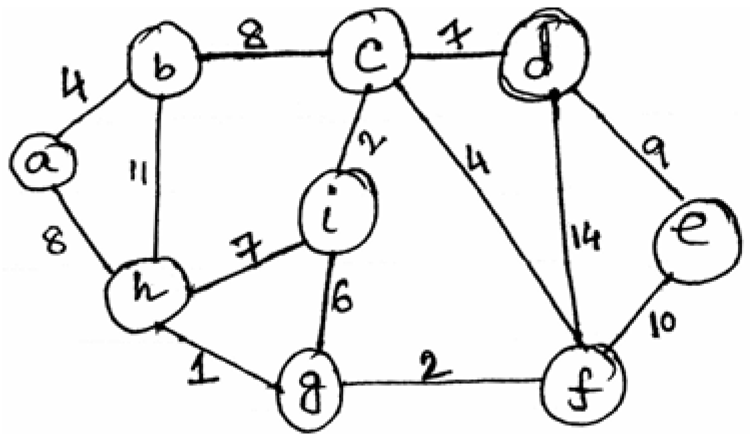
PYQ of (Analysis and Design of Algorithms) 5 Mark

2019

(a) Write Prims algorithm. Construct a minimum spanning tree for the following graph using prim's algorithm.



What is the minimum weight of your spanning tree? Is there any other MST with same weight? 3+4+1+2=10

b) Compare between breadth first search and depth first search. Write pseudo code for BFS. Explain your pseudo-code with suitable example. 2+5+3=10

2021

a) Describe insertion sort with a suitable example considering at least 5 inputs.

b) Write the merge sort algorithm. Explain the algorithm with suitable example. 5+5.

2022

a. State some distinguishing features of insertion sort. Write insertion sort algorithm and trace its execution sequence with a suitable example.

b. Write KMP algorithm for string processing and trace its execution sequence with a suitable example.

2022-23

a) Write quick sort algorithm and use it to sort the file 2, 7, 9, 1, 11, 6, 5, 43, 12, 10

b) What is precondition for applying binary search algorithm? Write binary search algorithm and determine its time complexity.

ANSWERS

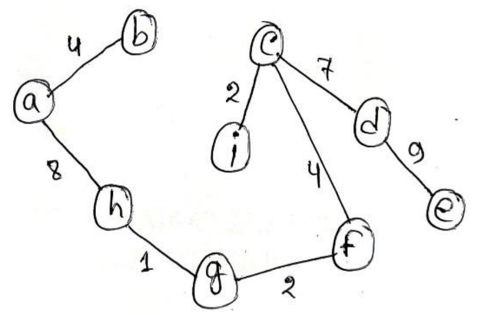
2019

a) Write Prim's algorithm. Construct a minimum spanning tree for the following graph using Prim's algorithm. What is the minimum weight of your spanning tree? Is there any other MST with the same weight?

Prim's Algorithm:

1. Initialize a set MST to store the vertices included in the MST.
2. Choose an arbitrary starting vertex and add it to MST.
3. While MST does not include all vertices:
    - Find the edge with the minimum weight that connects a vertex in MST to a vertex outside MST.
    - Add the selected edge and the vertex to MST.
4. Repeat until all vertices are included in MST.

MST for the following Graph:



Total Weight of MST:

4+8+1+2+4+2+7+9=37

Is there any other MST with the same weight?

Yes. Prim's algorithm is **not unique** in path selection — another MST with **different edges but same total weight (37)** may exist, depending on the choice of edges when weights are equal. For example, starting from a different vertex or choosing a different edge of equal weight can still yield the same total weight MST.

b) Compare between breadth-first search and depth-first search. Write pseudo code for BFS. Explain your pseudo-code with a suitable example.

Comparison:

Feature	Breadth-First Search (BFS)	Depth-First Search (DFS)
Traversal Method	Level by level	Deep into a branch before backtracking

Feature	Breadth-First Search (BFS)	Depth-First Search (DFS)
Data Structure Used	Queue	Stack (or recursion)
Completeness	Finds shortest path in unweighted graphs	May not find shortest path
Space Complexity	O(V)	O(V)
Time Complexity	O(V + E)	O(V + E)

BFS Pseudo Code:

pseudo

```
function BFS(graph, start):
  create a queue Q
  create a set visited
  enqueue start into Q
  add start to visited

  while Q is not empty:
    node = dequeue from Q
    process node
    for each neighbor of node:
      if neighbor not in visited:
        enqueue neighbor into Q
        add neighbor to visited
```

Explanation:

- Initialize a queue and a visited set.
- Start from the given node, mark it as visited, and enqueue it.
- While the queue is not empty:
  - Dequeue a node, process it.
  - For each unvisited neighbor, enqueue it and mark it as visited.

Example:

For the graph:

```
A---B---C
|   |
D--------E
```

Starting from A:

- Queue: [A]
- Dequeue A, process A, enqueue B, D.
- Queue: [B, D]
- Dequeue B, process B, enqueue C.
- Queue: [D, C]
- Dequeue D, process D, enqueue E.

- Queue: [C, E]
- Dequeue C, process C.
- Queue: [E]
- Dequeue E, process E.

Visited Order: A, B, D, C, E

2021

a) Describe insertion sort with a suitable example considering at least 5 inputs.

Insertion Sort Algorithm:

1. Start from the second element. Compare it with the elements before it.
2. Shift all elements greater than the current element to the right.
3. Insert the current element into its correct position.
4. Repeat for all elements.

Example:

Consider the array: [5, 2, 9, 1, 5]

- Start with 2:
  - Compare with 5, shift 5 to the right.
  - Insert 2 at the beginning.
  - Array: [2, 5, 9, 1, 5]
- Next, 9:
  - No change needed.
  - Array: [2, 5, 9, 1, 5]
- Next, 1:
  - Compare with 9, shift 9 to the right.
  - Compare with 5, shift 5 to the right.
  - Compare with 2, shift 2 to the right.
  - Insert 1 at the beginning.
  - Array: [1, 2, 5, 9, 5]
- Next, 5:
  - Compare with 9, shift 9 to the right.
  - Insert 5.
  - Array: [1, 2, 5, 5, 9]

Final Sorted Array: [1, 2, 5, 5, 9]

---

b) Write the merge sort algorithm. Explain the algorithm with a suitable example.

Merge Sort Algorithm:

1. Divide the array into two halves.
2. Recursively sort each half.
3. Merge the sorted halves.

Example:

Consider the array: [38, 27, 43, 3, 9, 82, 10]

- Divide: [38, 27, 43, 3] and [9, 82, 10]
- Sort each half:
  - [38, 27, 43, 3] → [3, 27, 38, 43]
  - [9, 82, 10] → [9, 10, 82]
- Merge:
  - [3, 27, 38, 43] and [9, 10, 82]
  - Merge to: [3, 9, 10, 27, 38, 43, 82]

Final Sorted Array: [3, 9, 10, 27, 38, 43, 82]

2022

a) State some distinguishing features of insertion sort. Write insertion sort algorithm and trace its execution sequence with a suitable example.

Distinguishing Features of Insertion Sort:

- Simple and intuitive.
- Efficient for small datasets or nearly sorted data.
- Adaptive: performs better on nearly sorted arrays.
- Stable: maintains the relative order of equal elements.
- In-place: requires only a constant amount of additional memory.

Insertion Sort Algorithm:

1. Start from the second element.
2. Compare it with the elements before it.
3. Shift all elements greater than the current element to the right.
4. Insert the current element into its correct position.
5. Repeat for all elements.

Example Array: [12, 11, 13, 5, 6]

- Step 1: 11 is compared with 12 → shift 12 right → Insert 11
  Array: [11, 12, 13, 5, 6]
- Step 2: 13 > 12 → no shift needed
  Array: [11, 12, 13, 5, 6]
- Step 3: 5 is compared with 13, 12, and 11 → shift them right → insert 5
  Array: [5, 11, 12, 13, 6]
- Step 4: 6 compared with 13, 12, and 11 → shift them → insert 6
  Array: [5, 6, 11, 12, 13]

Final Sorted Array: [5, 6, 11, 12, 13]

b) Write KMP algorithm for string processing and trace its execution sequence with a suitable example.

KMP Algorithm (Knuth-Morris-Pratt):

- Used for efficient **pattern matching** in a string.
- Avoids rechecking characters after a mismatch using a **prefix table (LPS array)**.

Steps:

1. Preprocess the pattern to create the LPS (Longest Prefix Suffix) array.
2. Start matching the pattern from left to right with the text.
3. On mismatch, use LPS array to avoid rechecking previous characters.

Example:

Text = "ABABDABACDABABCABAB"
Pattern = "ABABCABAB"

LPS Array for Pattern:

Index 0 1 2 3 4 5 6 7 8

Value 0 0 1 2 0 1 2 3 4

Matching Process:

- Start comparing text and pattern.
- After a few shifts, match found at **index 10** in text.

Output: Pattern found at index 10.

2022–23

a) Write quick sort algorithm and use it to sort the file 2, 7, 9, 1, 11, 6, 5, 43, 12, 10

Quick Sort Algorithm:

1. Choose a **pivot**.
2. Partition array into two halves:
   - Elements less than pivot go left.
   - Elements greater go right.
3. Recursively sort both halves.

Example: Sort [2, 7, 9, 1, 11, 6, 5, 43, 12, 10]

- Choose pivot: 10
- Partition:
  Left: [2, 7, 9, 1, 6, 5]
  Right: [11, 43, 12]
- Recursively sort left:
  - Pivot: 5 → Left: [2, 1], Right: [7, 9, 6]
  - Sort [2, 1] → [1, 2]
  - Sort [7, 9, 6] → Pivot 6 → [7, 9] → Final [6, 7, 9]
  - Combine: [1, 2, 5, 6, 7, 9]
- Sort right [11, 43, 12]:
  - Pivot: 12 → Left: [11], Right: [43] → Combine: [11, 12, 43]

Final Sorted Array: [1, 2, 5, 6, 7, 9, 10, 11, 12, 43]

b) What is precondition for applying binary search algorithm? Write binary search algorithm and determine its time complexity.

Precondition:

- The array **must be sorted** in ascending or descending order.

Binary Search Algorithm:

pseudo

BinarySearch(arr, key):

  low = 0

  high = length(arr) - 1

```
while low <= high:
  mid = (low + high) / 2
  if arr[mid] == key:
      return mid
  else if arr[mid] < key:
      low = mid + 1
  else:
      high = mid - 1

return -1  // not found
```

Time Complexity:

- **Best Case:** O(1) (key found at middle)
- **Worst Case:** O(log n)
- **Average Case:** O(log n)