Assignment-2

Section-A

1. What is Disjoint Set? What are its applications?

- A Disjoint Set (Union-Find) is a data structure that keeps track of elements partitioned into disjoint (non-overlapping) sets.
- Applications:
  - Kruskal's algorithm for minimum Spanning Tree
  - Network Connectivity
  - Image Processing (Connected Components)
  - Cycle detection in graphs.

2. What is a Red-Black Tree? What are its Properties?

- A Red Black Tree is a balanced binary Search tree where each node has a color (red or black) ensuring balance through Properties:
  - Each node is red or black.
  - Root is always black.
  - No two red nodes are adjacent.
  - Every path from a node to its descendant NULL nodes has the same numbers of black nodes

3. What is divide and conquer paradigm for Problem Solving.

- Divide and conquer involves:
  - Divide: Break the problem into subproblems
  - Conquer: Solve the subproblem recursively.
  - Combine: Merge solutions to get the final result.

4 - What is AVL Tree?

- An AVL Tree is a self-balancing binary search tree where the difference between heights of left and right subtrees cannot be more than 1 for any node.

Section-B

1. Algorithm:

```
void selectionSort (int arr[], int n){
    for (int i = 0; i<n-1; i++){
        int min_idx = i;
        for (int j = i+1; j<n; j++){
            if (arr[j] < arr[min_idx])
                min_idx = j;
        }
        Swap (arr[min_idx], arr[i]);
    }
}
```

Time Complexity:

- Best case: $O(n^2)$
- Average case: $O(n^2)$
- Worst case: $O(n^2)$

2. ~~Write an~~

Algorithm:

```
Void BFS (int start){
    Queue q;
    visited [start] = true;
    q.enqueue (start);
    While (!q.isEmpty ()){
        int node = q.dequeue ();
        for (each neighbour of node){
            if (! visited [neighbour]){
                visited [neighbour] = true;
```

```
            q.enqueue(neighbor);
        }
    }
}
```

BFS traversal for given graph starting at A:

- Order: A → B → C → D → F → E

Section-C

1. Heap: A complete binary tree where every parent node is greater (max-Heap) or smaller (min-Heap) than its children.

- MAX-HEAPIFY:

```
void maxHeapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;
    if (left < n && arr[left] > arr[largest])
        largest = left;
    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest != i) {
        swap(arr[i], arr[largest]);
        maxHeapify(arr, n, largest);
    }
}
```

- Heap Sort steps on A:

After building max heap: [84, 22, 19, 10, 5, 17, 6, 3]

Sorted array after Heap Sort:

[3, 5, 6, 10, 17, 19, 22, 84]

- Time Complexity:
  - O(n log n)

2. Algorithm:

```
void quickSort (int arr[], int low, int high){
    if (low < high ){
        int pi = Partition (arr, low, high);
        quickSort (arr, low, pi - 1);
        quickSort (arr, pi + 1, high);
    }
}
```

Partition

```
int Partition (int arr[], int low, int high) {
    int Pivot = arr[high];
    int i = low - 1;
    for (int j = low; j <= high - 1; j++)
    {
        if (arr[j] < Pivot ) {
            i++;
            swap (arr [i], arr [j]);
        }
    }
    swap (arr [i + 1], arr [high]);
    return i + 1;
}
```

Time Complexity:

- Best case: $O(n \log n)$
- Average case: $O(n \log n)$
- Worst case: $O(n^2)$

- Quick Sort steps for A:

Sorted array : [3, 5, 6, 10, 17, 19, 22, 84]