Adv Algo Practicals

Question 1 : Write a program to sort the elements of an array using Randomized Quick Sort (the program should report the number of comparisons)

Solution :

- **Time Complexity:**
  - Best/Average Case: O(n log n) – Due to balanced partitioning with random pivot selection.
  - Worst Case: $O(n^2)$ – Occurs when the partitioning is highly unbalanced.
- **Space Complexity:** O(log n) – Recursion depth for the average case, as Quick Sort is in-place.

Output :

```
Initial array: 15 3 45 10 29 8 19 42
Partitioning with pivot: 10 at position: 7
Pivot finalized at position: 2
Partitioning with pivot: 3 at position: 1
Pivot finalized at position: 0
Partitioning with pivot: 29 at position: 7
Pivot finalized at position: 5
Partitioning with pivot: 15 at position: 4
Pivot finalized at position: 3
Partitioning with pivot: 42 at position: 7
Pivot finalized at position: 6

Sorted array: 3 8 10 15 19 29 42 45
Total comparisons made: 14
```

Question 2 : Write a program to find the ith smallest element of an array using Randomized Select.

Solution :

Time Complexity:
Average Case: O(n) – The randomized pivot selection ensures that on average, the array is divided evenly.
Worst Case: $O(n^2)$ – Occurs when the pivot selection repeatedly results in highly unbalanced partitions.

Space Complexity: O(1) – The algorithm operates in-place with no additional data structures, but recursive calls are replaced by an iterative loop to maintain O(1) space.

Output :

```
Initial array: 15 3 45 10 29 8 19 42
Enter the value of i (to find the ith smallest element): 3
Pivot chosen: 15
Pivot chosen: 3
Pivot chosen: 8
Pivot chosen: 10
The 3th smallest element is: 10
```

Question 3 : Write a program to determine the minimum spanning tree of a graph using Kruskal's algorithm

Solution :

Overall Time Complexity: O(E * log(E) + E * alpha(V)) which simplifies to O(E * log(E)) in most cases since alpha(V) is nearly constant.

Overall Space Complexity: O(V + E).

Output :

```
Edges in the Minimum Spanning Tree:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
```

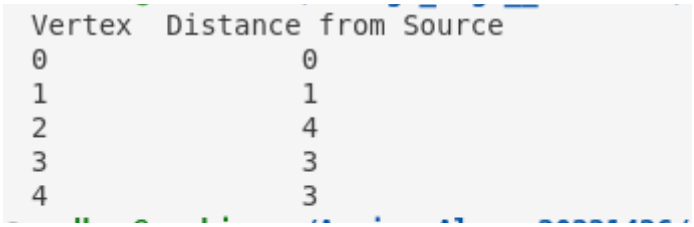
Question 4 : Write a program to implement the Bellman-Ford algorithm to find the shortest paths from a given source node to all other nodes in a graph

Solution :

Time Complexity Analysis:
Overall Time Complexity: O(V * E).

Space Complexity Analysis:
Overall Space Complexity: O(V + E).

Output :



Question 5 : Write a program to implement a B-Tree

Solution :

Time Complexity Analysis:
1. Insertion: O(t * log_t(N)) where t is the degree of the tree, and N is the number of nodes. This complexity arises from splitting the nodes and traversing the tree.
2. Search: O(log_t(N)) where t is the degree of the tree, and N is the number of nodes.
3. Traversal: O(N) where N is the total number of keys in the tree.

Space Complexity Analysis:
1. The space complexity is O(N) for storing keys and child pointers, where N is the number of keys.

Output :

```
Traversal of the constructed B-Tree is:
 5 6 7 10 12 17 20 30

Searching for element: 6
Present

Searching for element: 15
Not Present
```

Question 6 : Write a program to implement the Tree Data structure, which supports the following operations:
a. Insert
b. Search

Solution :

Time Complexity Analysis:
1. Insertion: O(h), where h is the height of the tree. In the worst case (unbalanced tree), h = O(n).
2. Search: O(h), where h is the height of the tree. In the worst case, h = O(n).
3. Deletion: O(h), where h is the height of the tree. In the worst case, h = O(n).
4. Traversal (in-order): O(n), where n is the number of nodes in the tree.

Space Complexity Analysis:
1. Space complexity is O(n) for storing all nodes in the tree.

Output :

```
Initial tree (in-order traversal):
20 30 40 50 60 70 80

Searching for elements:
Element 40 found in the tree.
Element 90 not found in the tree.

Deleting nodes:
Deleted key: 20
30 40 50 60 70 80
Deleted key: 30
40 50 60 70 80
Deleted key: 50
40 60 70 80
```

Question 7 : Write a program to search a pattern in a given text using the KMP algorithm

Solution :

Time Complexity Analysis:
1. Preprocessing the Pattern (LPS Array): O(m), where m is the length of the pattern.
2. Searching the Pattern in Text: O(n), where n is the length of the text.

Overall Time Complexity: O(n + m).

Space Complexity Analysis:
1. Space Complexity for LPS Array: O(m).
2. Overall Space Complexity: O(m).

Output :

```
Text: ababcabcabababd
Pattern: ababd
Pattern found at index 10
```

Question 8 : Write a program to implement a Suffix tree

Solution :

Time Complexity Analysis:
1. Building the Suffix Tree: O(n),
2. **Space Complexity:** O(n * |Σ|), where |Σ| is the alphabet size. The space complexity is determined by the number of nodes and edges in the suffix tree.

Output :