Clean Code

**Q. Write a program for various sorting and searching algorithms.**

* Program should be developed based on clean code principles and document the code.
* Create a flowchart explaining program flow
* Create Algorithms for program

**Ans**: **Sorting Algorithms –**

1. **Bubble Sort** – It works by repeatedly swapping the adjacent elements if they are in wrong order.

**Flowchart** –

**Algorithm** –

1. Start the algorithm.
2. Define a function bubble\_sort that takes an array “arr” as input.
3. Set n to the length of the array arr.
4. Initialize a for loop with index I, which iterates in range of 0 to n-1.
5. Initialize a nested for loop with index j, which iterates in range of 0 to n-i-1.
6. Compare adjacent elements arr[j] and arr[j+1].
7. If arr[j] is greater than arr[j+1], swap the elements.
8. End the nested loop.
9. End the outer loop.
10. Output the sorted array arr.
11. End the algorithm.

Code -

def bubble\_sort(arr):

    n = len(arr)

    # Traverse through all array elements

    for i in range(n):

        # Last i elements are already in place

        for j in range(0, n-i-1):

            # Compare adjacent elements and swap them if they are in the wrong order

            if arr[j] > arr[j+1]:

                arr[j], arr[j+1] = arr[j+1], arr[j]

arr = [40, 80, 25, 12]

bubble\_sort(arr)

print("After bubble sort : ", arr)

Output – After bubble sort : [12, 25, 40, 80]

1. **Insertion Sort –** It works by iteratively building a sorted portion of the array, while placing each unsorted element in its correct position within sorted position.

**Flowchart** –

**Algorithm** –

1. Start the algorithm.
2. Define a function insertion\_sort that takes an array “arr” as input.
3. Set n to the length of the array arr.
4. Initialize a for loop with index i, which iterates in range of 1 to n-1.
5. Set key to arr[i].
6. Set j to i-1.
7. While j is greater than or equal to 0 and key is less than arr[j], do steps 8 to 9.
8. Assign arr[j+1] the value of arr[j].
9. Decrement j by 1.
10. End the while loop.
11. Assign arr[j+1] the value of key.
12. End the for loop.
13. Output the sorted array arr.
14. End the algorithm.

**Code** –

def insertion\_sort(arr):

    n = len(arr)

    # Loop through the array, starting from the second element

    for i in range(1, n):

        key = arr[i]

        j = i-1

        while j >= 0 and key < arr[j]:

            # Shift the value at index j to index j+1 in the array

            arr[j+1] = arr[j]

            j -= 1

        arr[j+1] = key

arr = [20, 50, 10, 30]

insertion\_sort(arr)

print("After insertion sort : ", arr)

**Output** – After insertion sort : [10, 20, 30, 50]

1. **Selection Sort -** It sorts an array by repeatedly finding the minimum element from unsorted part and putting it in the beginning of the sorted part.

**Flowchart** –

**Algorithm** –

1. Start the algorithm.
2. Define a function selection\_sort that takes an array “arr” as input.
3. Set n to the length of the array “arr”.
4. Initialize a for loop with index i, which iterates in range of 0 to n-1.
5. Set min\_index to i.
6. Initialize a for loop with index j, which iterates in range of i+1 to n-1.
7. If arr[j] is less than arr[min\_index], set min\_index to j.
8. Swap the values of arr[i] and arr[min\_index].
9. End the inner for loop.
10. End the outer for loop.
11. Output the sorted array arr.
12. End the function.
13. Call the function selection\_sort with the input array arr.
14. Print the sorted array arr.
15. End the algorithm.

**Code** –

def selection\_sort(arr):

    n = len(arr)

    for i in range(n-1):

        # Find the minimum element in the remaining unsorted array

        min\_index = i

        for j in range(i+1, n):

            if arr[j] < arr[min\_index]:

                min\_index = j

        # Swap the found minimum element with the first element

        arr[i], arr[min\_index] = arr[min\_index], arr[i]

arr = [50, 20, 30, 10]

selection\_sort(arr)

print("After selection sort :", arr)

**Output** - After selection sort : [10, 20, 30, 50]

1. **Merge** **Sort** – Merge sort is a popular sorting algorithm that follows the divide-and-conquer approach. It divides the input array into smaller subarrays, sorts them individually, and then merges them back together to produce a sorted output array. Here's how the merge sort algorithm works:

Divide: The input array is recursively divided into two halves until each subarray contains only one element or is empty.

Conquer: Once the array is divided into its smallest parts, the merging process begins. It compares the elements of the divided subarrays and merges them into larger sorted subarrays. This process continues until all subarrays are merged back into a single sorted array.

Merge: In the merging step, two sorted subarrays are combined into a single sorted subarray. The algorithm compares the elements from both subarrays and adds the smaller element to the output array. This process continues until all elements from both subarrays are merged.

Flowchart –

**Algorithm** –

1. Start the algorithm.

2. Define a function selection\_sort that takes an array "arr" as input.

3. Set n to the length of the array "arr".

4. Initialize a for loop with index i, which iterates in the range of 0 to n-1.

a. Set min\_index to i.

b. Initialize a for loop with index j, which iterates in the range of i+1 to n-1.

i. If arr[j] is less than arr[min\_index], set min\_index to j.

c. Swap the values of arr[i] and arr[min\_index].

5. End the inner for loop.

6. End the outer for loop.

7. Output the sorted array arr.

8. End the function.

9. Call the function selection\_sort with the input array arr.

10. Print the sorted array arr.

11. End the algorithm.

**Code** –

def merge\_sort(arr):

    if len(arr) <= 1:

        return arr

    mid = len(arr) // 2

    left = arr[:mid]

    right = arr[mid:]

    left = merge\_sort(left)

    right = merge\_sort(right)

    return merge(left, right)

def merge(left, right):

    result = []

    i = j = 0

    while i < len(left) and j < len(right):

        if left[i] <= right[j]:

            result.append(left[i])

            i += 1

        else:

            result.append(right[j])

            j += 1

    result.extend(left[i:])

    result.extend(right[j:])

    return result

arr = [4, 2, 9, 1, 6, 5, 3, 8, 7]

sorted\_arr = merge\_sort(arr)

print(sorted\_arr)