EXPERIMENT NO.2

1)Quick Sort Algorithm:-

Program:-

```
#include <stdio.h>

// Function to partition the array and return the pivot index
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            // Swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    // Swap arr[i+1] and arr[high] (pivot)
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return i + 1;
}

// Function to perform QuickSort on the array
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // Partition the array and get the pivot index
        int pivotIndex = partition(arr, low, high);

        // Recursively sort the subarrays
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}

// Function to print an array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
```

```
    }
    printf("\n");
}

// Example usage
int main() {
    int arr[] = {12, 5, 7, 3, 2, 8, 4};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: ");
    printArray(arr, size);

    quickSort(arr, 0, size - 1);

    printf("Sorted array: ");
    printArray(arr, size);

    return 0;
}
```

Output:-

```
/tmp/p2rMWLGxdk.o
Original array: 12 5 7 3 2 8 4
Sorted array: 2 3 4 5 7 8 12


=== Code Execution Successful ===
```

2)Merge Sort Algorithm

Program:-

```
#include <stdio.h>

// Merge two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
```

```
    // Create temporary arrays
    int L[n1], R[n2];

    // Copy data to temporary arrays L[] and R[]
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    // Merge the temporary arrays back into arr[l..r]
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of L[], if there are any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy the remaining elements of R[], if there are any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// l is for left index and r is right index of the sub-array of arr to be sorted
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for large l and r
```

```
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

// Function to print an array
void printArray(int A[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

// Driver program to test above functions
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

Output:-

```
/tmp/4V9N6Mmdqo.o
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13


=== Code Execution Successful ===
```

3)Binary Search Algorithm

Program:-

```
#include <stdio.h>

// Function to perform binary search
int binarySearch(int arr[], int left, int right, int target) {
    while (left <= right) {
        int mid = left + (right - left) / 2;

        // Check if target is present at mid
        if (arr[mid] == target)
            return mid;

        // If target is greater, ignore left half
        if (arr[mid] < target)
            left = mid + 1;

        // If target is smaller, ignore right half
        else
            right = mid - 1;
    }

    // If target is not found, return -1
    return -1;
}

// Driver program to test above function
int main() {
    int arr[] = {2, 3, 4, 10, 40};
    int target = 10;
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    int result = binarySearch(arr, 0, arr_size - 1, target);

    if (result != -1)
        printf("Element is present at index %d\n", result);
    else
        printf("Element is not present in array\n");

    return 0;
}
```

Output:-

```
/tmp/5OVrrA5GzV.o
Element is present at index 3


=== Code Execution Successful ===
```