Design and Analysis of Algorithm

Lab 2

Name: Harsh Brahmecha
PRN: 20220802003

1) Write a C code to implement Binary Search

```
#include <stdio.h>

// Binary Search function
int binarySearch(int arr[], int size, int target) {
    int left = 0;
    int right = size - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        // Check if the target is found at the middle
        if (arr[mid] == target) {
            return mid;
        }

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

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

    // Target not found in the array
    return -1;
}

int main() {
    int arr[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 12;
```

```
    int result = binarySearch(arr, size, target);

    if (result != -1) {
        printf("Element %d found at index %d.\n", target, result);
    } else {
        printf("Element %d not found in the array.\n", target);
    }

    return 0;
}
```

Output

```
Element 12 found at index 5.
```

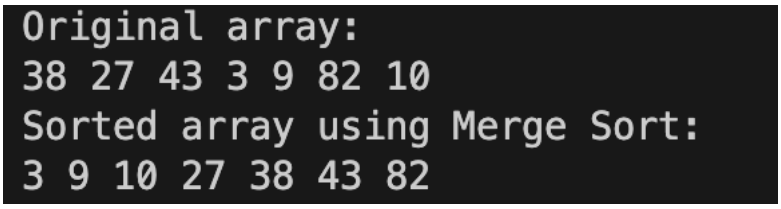
2) Write a C code to implement merge sort

```
#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;
```

```
    j = 0;
    k = l;
    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++;
    }
}

// Merge Sort function
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        // Find the middle point
        int m = l + (r - l) / 2;

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

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

int main() {
    int arr[] = {38, 27, 43, 3, 9, 82, 10};
```

```
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Original array:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Perform Merge Sort
    mergeSort(arr, 0, size - 1);

    printf("Sorted array using Merge Sort:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Output



3) Write a C code to implement quick sort

```
#include <stdio.h>

// Function to swap two elements in an array
void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```

```
// Partition the array and return the pivot index
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // Choose the rightmost element as the pivot
    int i = (low - 1);    // Initialize the index of the smaller element

    for (int j = low; j <= high - 1; j++) {
        // If the current element is smaller than or equal to the pivot
        if (arr[j] <= pivot) {
            i++; // Increment the index of the smaller element
            swap(&arr[i], &arr[j]); // Swap arr[i] and arr[j]
        }
    }

    swap(&arr[i + 1], &arr[high]); // Swap the pivot element with the element at
(i + 1)
    return (i + 1);
}

// Quick Sort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // Find pivot element such that
        // element smaller than pivot are on the left
        // and elements greater than pivot are on the right
        int pi = partition(arr, low, high);

        // Recursively sort elements before and after pivot
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    int arr[] = {98, 27, 43, 2, 29, 82, 10};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Original array:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Perform Quick Sort
    quickSort(arr, 0, size - 1);
```

```
    printf("Sorted array using Quick Sort:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Output

```
Original array:
98 27 43 2 29 82 10
Sorted array using Quick Sort:
2 10 27 29 43 82 98
```