Aim ↬ To perform basic operations on Search and Sorting.

Objectives ↬

Applications of Divide and Conquer

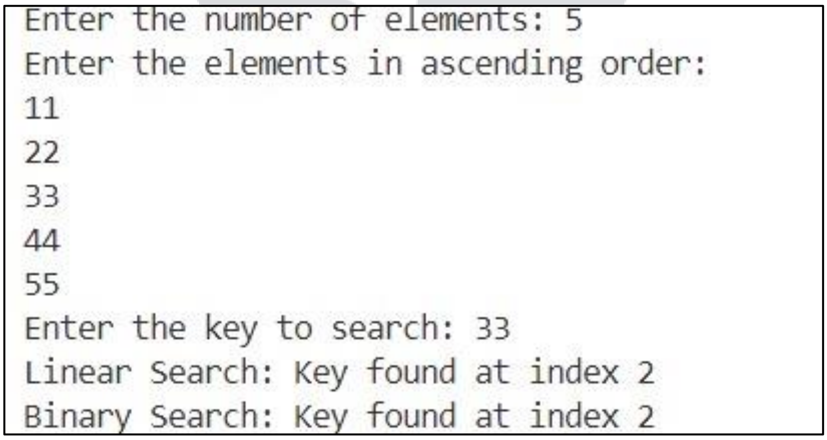
i. Write a program to compare linear search and binary search.
ii. Write a program to implement Merge Sort Algorithm.
iii. Write a program to implement Quick Sort algorithm.

Software Required ↬ Visual Studio Code

Code 1 ↬

```
#include <stdio.h>

int linearSearch(int arr[], int size, int key) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) return i;
    }
    return -1;
}

int binarySearch(int arr[], int size, int key) {
    int low = 0, high = size - 1, mid;
    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] == key) return mid;
        if (arr[mid] < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}

int main() {
    int size, key, result;

    printf("Enter the number of elements: ");
    scanf("%d", &size);

    int arr[size];
    printf("Enter the elements in ascending order:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Enter the key to search: ");
    scanf("%d", &key);

    result = linearSearch(arr, size, key);
    if (result != -1) printf("Linear Search: Key found at index %d\n", result);
    else printf("Linear Search: Key not found\n");

    result = binarySearch(arr, size, key);
    if (result != -1) printf("Binary Search: Key found at index %d\n", result);
    else printf("Binary Search: Key not found\n");

    return 0;
}
```

Output ↬

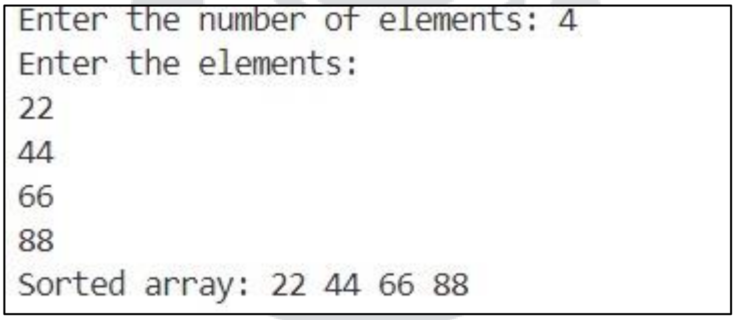


```
Enter the number of elements: 5
Enter the elements in ascending order:
11
22
33
44
55
Enter the key to search: 33
Linear Search: Key found at index 2
Binary Search: Key found at index 2
```

Code 2 ↬

```
#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

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

int main() {
    int size;

    printf("Enter the number of elements: ");
    scanf("%d", &size);
    int arr[size];

    printf("Enter the elements:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    mergeSort(arr, 0, size - 1);

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

    return 0;
}
```

Output ↬

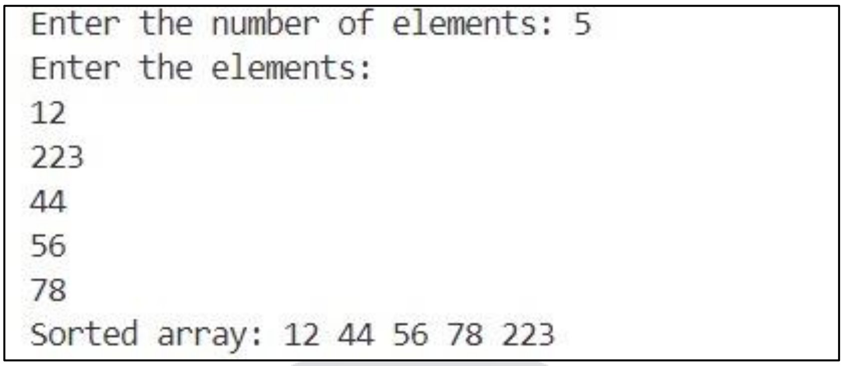


```
Enter the number of elements: 4
Enter the elements:
22
44
66
88
Sorted array: 22 44 66 88
```

Code 3 ↬

```
#include <stdio.h>

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++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return i + 1;
}

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);
    }
}

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

int main() {
    int size;

    printf("Enter the number of elements: ");
    scanf("%d", &size);

    int arr[size];

    printf("Enter the elements:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    quickSort(arr, 0, size - 1);
    printf("Sorted array: ");
    printArray(arr, size);

    return 0;
}
```

Output ↬



```
Enter the number of elements: 5
Enter the elements:
12
223
44
56
78
Sorted array: 12 44 56 78 223
```

Result ↬

The programs effectively implemented and compared:

- **Search Algorithms**: Linear and Binary Search.
- **Sorting Algorithms**: Merge Sort and Quick Sort.

Conclusion ↬

The experiment demonstrated the successful implementation of search and sorting algorithms, providing insights into their performance and efficiency.

Precautions ↬

- Ensure input data is valid and sorted correctly for binary search.
- Manage memory allocation and deallocation effectively.
- Handle edge cases, such as empty arrays or invalid indices.