PRINCE KUMAR
CSE- C
Admission No. : 2023b0101177

EXPERIMENT NO. 1

AIM: To implement Linear Search

THEORY:

Linear Search (also called Sequential Search) is the simplest searching algorithm. It checks each element of the array one by one until the required element (called key or target) is found, or the end of the array is reached.

CODE:

```cpp
#include <iostream>
using namespace std;

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

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key;
    cin >> key;

    int result = linearSearch(arr, n, key);

    if (result != -1)
        cout << "Element found at index " << result << endl;
    else
        cout << "Element not found" << endl;

    return 0;
}
```

OUTPUT:

```
Enter any 10 Numbers: 1
2
3
4
5
6
7
8
9
10

Enter a Number to Search: 7

Found at Index No.6
```

PRINCE KUMAR
CSE- C
Admission No. : 2023b0101177

EXPERIMENT NO. 2

AIM: To implement Binary Search

THEORY:

Binary Search is an efficient search algorithm used to find the position of a target element in a sorted array.
It works on the principle of divide and conquer by repeatedly halving the search interval.

CODE:

```cpp
#include <iostream>
using namespace std;

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

int main() {
    int arr[] = {10, 20, 30, 40, 50, 60};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key;
    cin >> key;
    int result = binarySearch(arr, n, key);
    if (result != -1)
        cout << "Element found at index " << result << endl;
    else
        cout << "Element not found" << endl;
    return 0;
}
```

OUTPUT:

```
Enter 10 elements (in ascending order): 1
2
3
4
5
6
7
8
9
10

Enter element to be search: 7

The number, 7 found at Position 7
```

PRINCE KUMAR
CSE- C
Admission No. : 2023b0101177

EXPERIMENT NO. 3

AIM: To implement the Bubble Sort

THEORY:

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping adjacent elements if they are in the wrong order.
The largest element “bubbles up” to the end in each pass.

CODE:

```cpp
#include <iostream>
using namespace std;

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
    bubbleSort(arr, n);
    printArray(arr, n);
    return 0;
```

OUTPUT:

```
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90
Enter element to search: 25
The number 25 found at position 4
```

PRINCE KUMAR
CSE- C
Admission No. : 2023b0101177

EXPERIMENT NO. 4

AIM: To implement Selection Sort

THEORY:

Selection Sort is a simple comparison-based sorting algorithm.
It repeatedly selects the smallest (or largest) element from the unsorted part of the array and places it in the correct position in the sorted part.

CODE:

```cpp
#include <iostream>
using namespace std;

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    int arr[] = {5, 3, 4, 1, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    insertionSort(arr, n);
    printArray(arr, n);
    return 0;
}
```

OUTPUT:

```
Enter size for Array: 5
Enter 5 array elements: 54
21
8
18
3

Now the Array after sorting is:
3 8 18 21 54
```

PRINCE KUMAR
CSE- C
Admission No. : 2023b0101177

EXPERIMENT NO. 5

AIM: To implement Insertion Sort

THEORY:

Insertion Sort is a simple comparison-based sorting algorithm. It builds the final sorted array one element at a time by picking an element and placing it in its correct position among the already sorted part.

CODE:

```
#include <iostream>
using namespace std;

void insertionSort(int arr[], int n) {
    for(int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while(j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int arr[] = {5, 3, 8, 2, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    insertionSort(arr, n);
    for(int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

OUTPUT:

```
Enter Array Size: 5
Enter 5 Array Elements: 28
16
5
11
0

Sorted Array:
0 5 11 16 28
```

PRINCE KUMAR
CSE- C
Admission No. : 2023b0101177

EXPERIMENT NO. 6

AIM: To implement Merge Sort

THEORY:

- Merge Sort is a Divide and Conquer sorting algorithm.
- It repeatedly divides the array into two halves, sorts each half, and then merges the sorted halves to produce the final sorted array.

2. Working Principle

1. **Divide**: Split array into two halves (until each subarray has one element).
2. **Conquer**: Sort the subarrays recursively.
3. **Combine**: Merge the sorted subarrays into a single sorted array.

CODE:

```
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];

    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    int 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++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++; k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++; k++;
    }
}
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}
```

OUTPUT:

```
Enter number of elements in the Array: 5
Enter 5 elements:
15 25 96 85 74
The Sorted List is:
15 25 74 85 96
```

PRINCE KUMAR
CSE- C
Admission No. : 2023b0101177

EXPERIMENT NO. 7

AIM: To implement Quick Sort

THEORY:

Quick Sort is a divide-and-conquer sorting algorithm that works by:

1. Choosing a pivot element from the array.
2. Partitioning the array into two parts:
   - Left side → elements smaller than pivot.
   - Right side → elements greater than pivot.
3. Recursively

CODE:

```cpp
#include <bits/stdc++.h>
using namespace std;

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], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    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 n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    printArray(arr, n);
    quickSort(arr, 0, n - 1);
    cout << "Sorted array: ";
    printArray(arr, n);
    return 0;
}
```

OUTPUT:

```
How many elements are you going to enter?: 6
Enter 6 elements: 36 8 5 96 7 5
Order of sorted elements:
5 5 7 8 36 96
```

PRINCE KUMAR
CSE- C
Admission No. : 2023b0101177

EXPERIMENT NO. 8

AIM: To implement Heap Sort

THEORY:

- Heap Sort is a comparison-based sorting algorithm.
- It uses a binary heap data structure (usually a max-heap) to sort elements.
- It is an in-place algorithm (does not need extra memory like Merge Sort).

CODE:

```cpp
#include <iostream>
using namespace std;

void heapify(int arr[], int n, int i) {
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;

    if(l < n && arr[l] > arr[largest]) largest = l;
    if(r < n && arr[r] > arr[largest]) largest = r;

    if(largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    for(int i = n/2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for(int i = n-1; i >= 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

int main() {
    int arr[] = {5, 3, 8, 2, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    heapSort(arr, n);
    for(int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

OUTPUT:

```
How many elements are you going to enter?:
5
Enter 5 Elements: 96 8 45 7 58
Order of sorted elements:
7 8 45 58 96
```