

LAB 13: Searching and Sorting Arrays

CLOs: CLO1, CLO4

Searching:

A search algorithm is a method of locating a specific item in a larger collection of data.

This section discusses two algorithms for searching the contents of an array.

The Linear Search:

The linear search is a very simple algorithm. Sometimes called a sequential search, it uses a loop to sequentially step through an array, starting with the first element. It compares each element with the value being searched for and stops when either the value is found or the end of the array is encountered. If the value being searched for is not in the array, the algorithm will unsuccessfully search to the end of the array

Example:

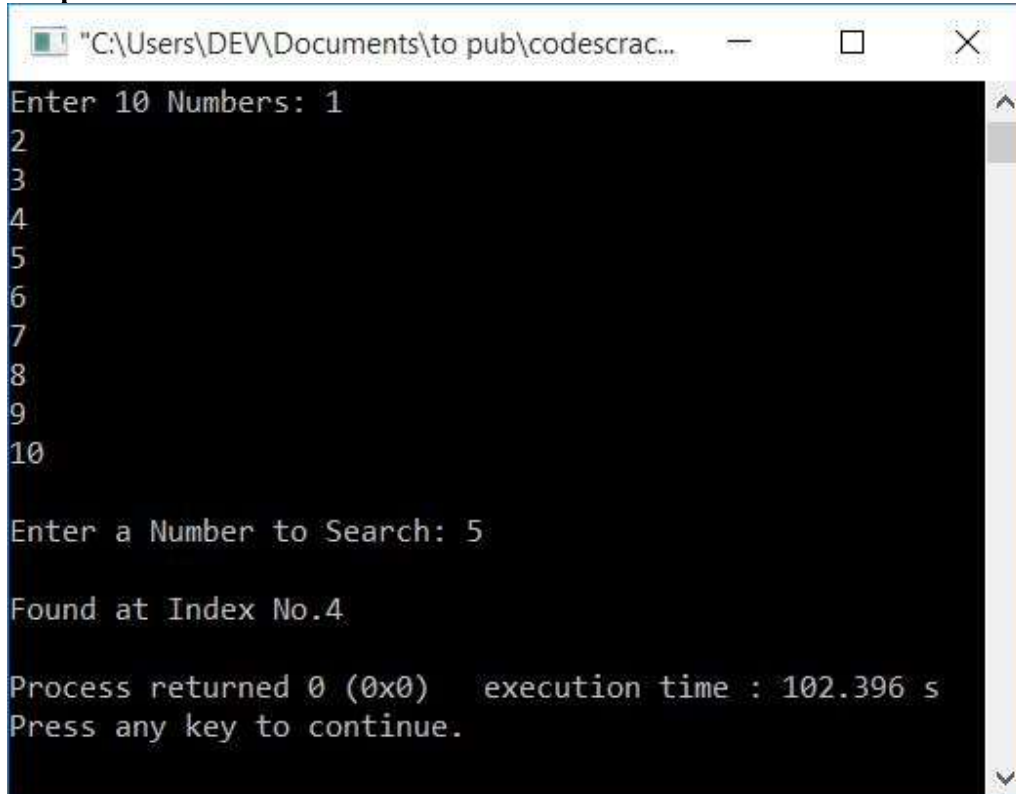
```
#include<iostream>

using namespace std;

int main()
{
    int arr[10], i, num, index;
    cout<<"Enter 10 Numbers: ";
    for(i=0; i<10; i++)
        cin>>arr[i];
    cout<<"\nEnter a Number to Search: ";
    cin>>num;
    for(i=0; i<10; i++)
    {
        if(arr[i]==num)
        {
            index = i;
            break;
        }
    }
    cout<<"\nFound at Index No."<<index;
```

```
cout<<endl;
return 0;
}
```

Output:



```
"C:\Users\DEV\Documents\to pub\codescrac...
Enter 10 Numbers: 1
2
3
4
5
6
7
8
9
10
Enter a Number to Search: 5
Found at Index No.4
Process returned 0 (0x0) execution time : 102.396 s
Press any key to continue.
```

Binary Search:

Binary Search: Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

We basically ignore half of the elements just after one comparison.

- Compare x with the middle element.
- If x matches with middle element, we return the mid index.
- Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So, we recur for right half.
- Else (x is smaller) recur for the left half.

Example:

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

```

int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r) {
        int m = l + (r - l) / 2;

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

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }

    // if we reach here, then element was
    // not present
    return -1;
}

int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = binarySearch(arr, 0, n - 1, x);
    (result == -1) ? cout << "Element is not present in array"
                  : cout << "Element is present at index " << result;
    return 0;
}

```

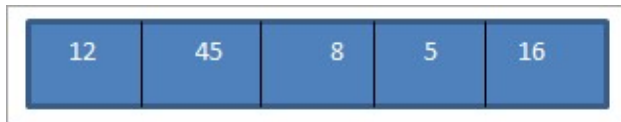
Sorting Arrays:

Sorting is a technique that is implemented to arrange the data in a specific order. Sorting is required to ensure that the data which we use is in a particular order so that we can easily retrieve the required piece of information from the pile of data.

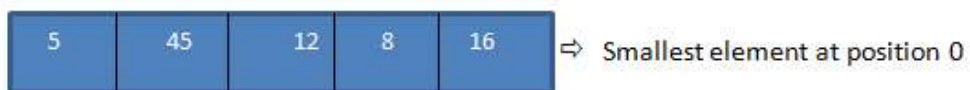
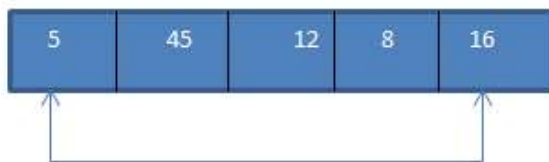
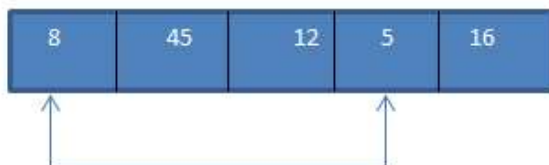
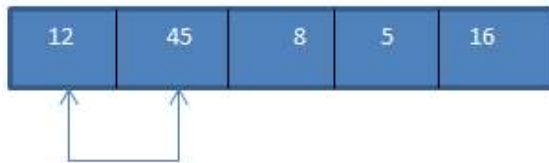
Selection Sort:

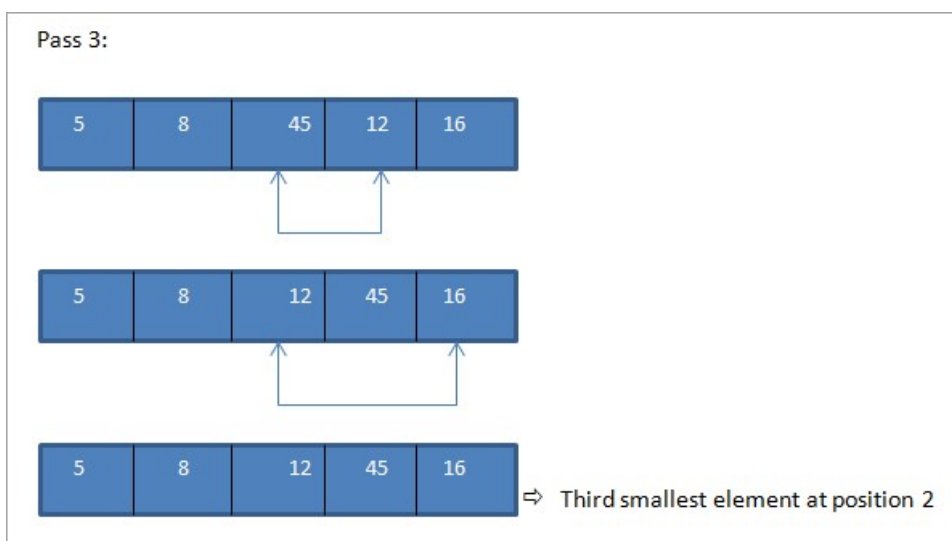
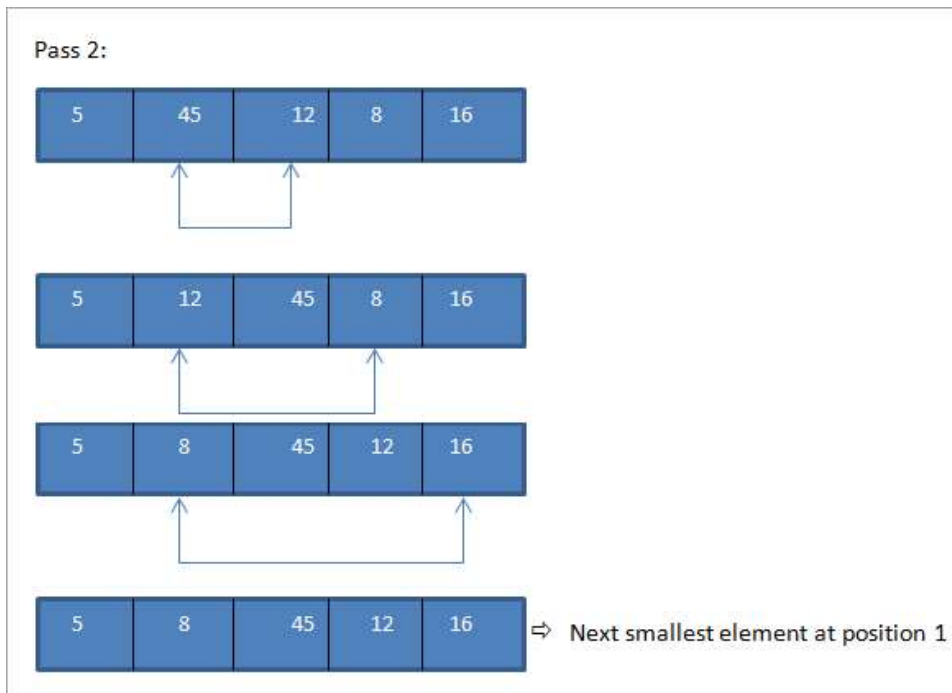
It is simple yet easy to implement technique in which we find the smallest element in the list and put it in its proper place. At each pass, the next smallest element is selected and placed in its proper position.

Let us take the same array as in the previous example and perform Selection Sort to sort this array.



Pass 1:





As shown in the above illustration, for N number of elements we take N-1 passes to completely sort the array. At the end of every pass, the smallest element in the array is placed at its proper position in the sorted array.

Example:

```
#include<iostream>
using namespace std;
int findSmallest (int[],int);
int main ()
{
    int myarray[5] = {12,45,8,15,33};
    int pos,temp;
    cout<<"\n Input list of elements to be Sorted\n";
    for(int i=0;i<5;i++)
    {
        cout<<myarray[i]<<"\t";
    }
    for(int i=0;i<5;i++)
    {
        pos = findSmallest (myarray,i);
        temp = myarray[i];
        myarray[i]=myarray[pos];
        myarray[pos] = temp;
    }
    cout<<"\n Sorted list of elements is\n";
    for(int i=0;i<5;i++)
    {
        cout<<myarray[i]<<"\t";
    }
    return 0;
}
```

```
int findSmallest(int myarray[],int i)
{
    int ele_small,position,j;
    ele_small = myarray[i];
    position = i;
    for(j=i+1;j<5;j++)
    {
        if(myarray[j]<ele_small)
        {
            ele_small = myarray[j];
            position=j;
        }
    }
    return position;
}
```

Output:

Input list of elements to be Sorted
12 45 8 15 33

Sorted list of elements is

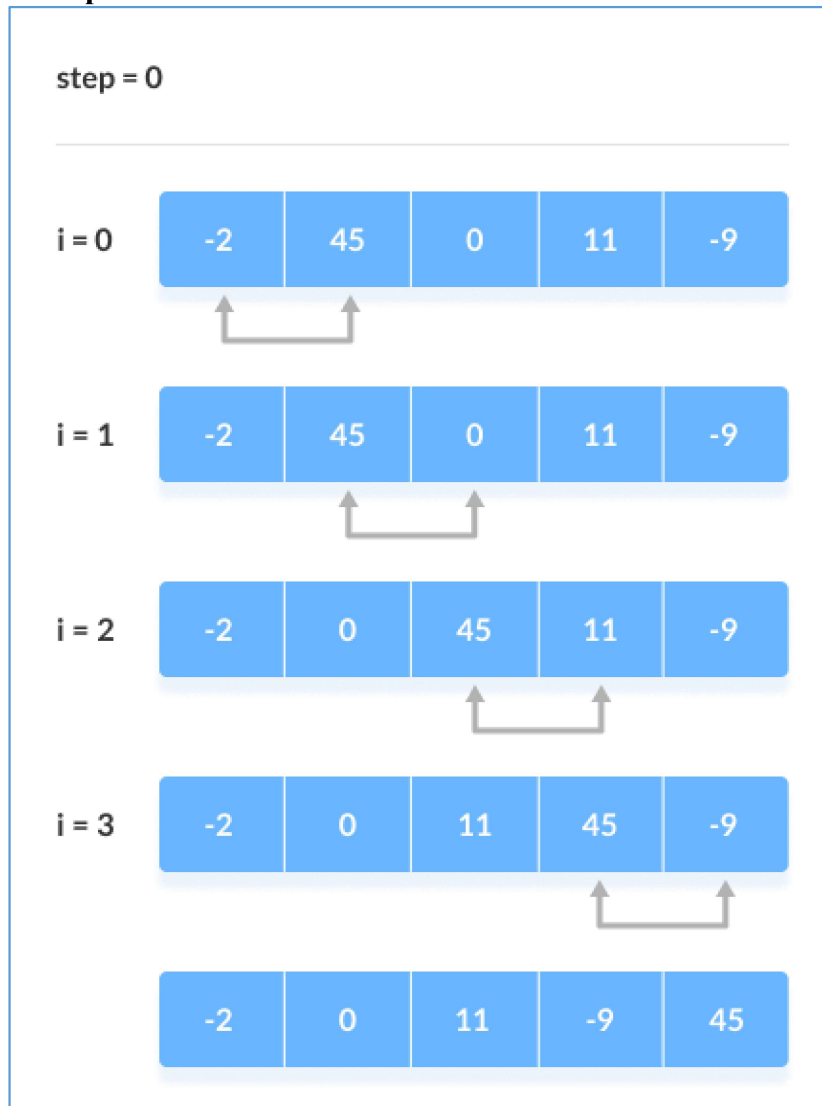
8 12 15 33 45

In selection sort, with every pass, the smallest element in the array is placed in its proper position. Hence at the end of the sorting process, we get a completely sorted array.

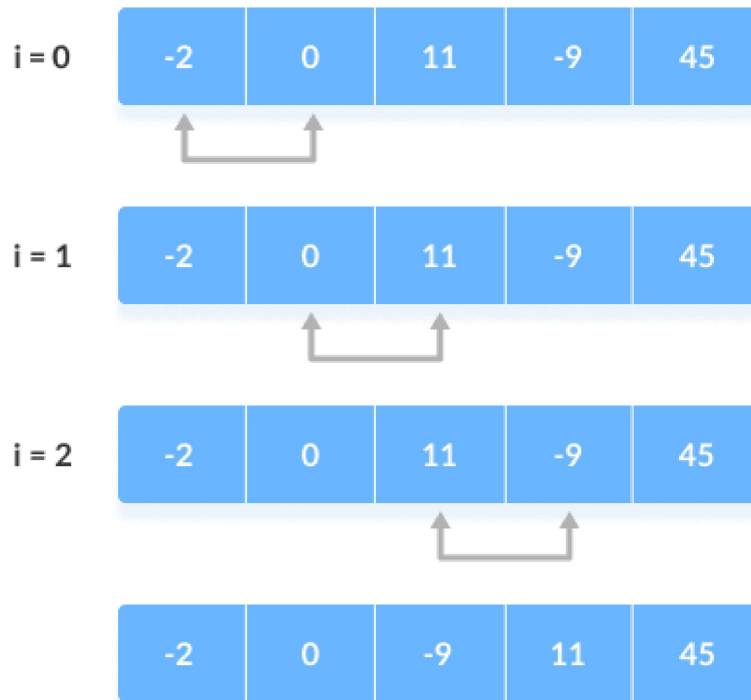
Bubble Sort:

Bubble sort is an algorithm that compares the adjacent elements and swaps their positions if they are not in the intended order. The order can be ascending or descending.

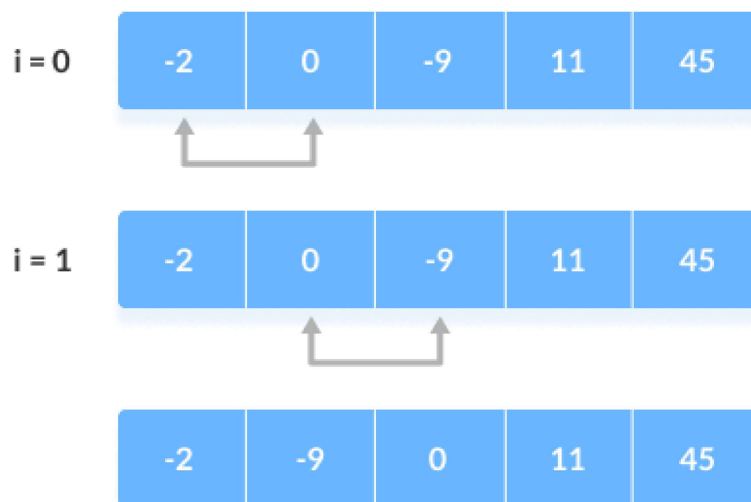
Example:



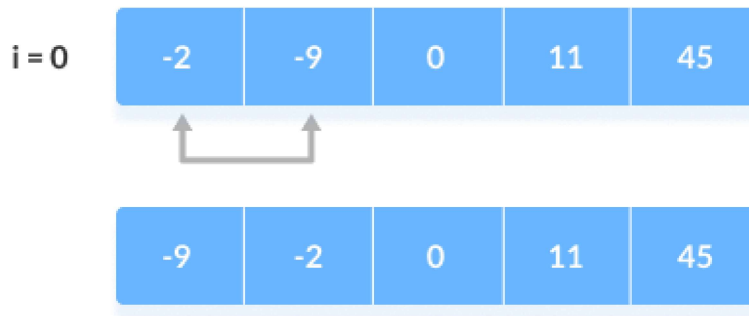
step = 1



step = 2



step = 3



Example:

```
#include <iostream>
using namespace std;
void bubbleSort(int array[], int size) {

    // run loops two times: one for walking through the array
    // and the other for comparison
    for (int step = 0; step < size - 1; ++step) {
        for (int i = 0; i < size - step - 1; ++i) {

            // To sort in descending order, change > to < in this line.
            if (array[i] > array[i + 1]) {

                // swap if greater is at the rear position
                int temp = array[i];
                array[i] = array[i + 1];
                array[i + 1] = temp;
            }
        }
    }
}

// function to print the array
void printArray(int array[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << " " << array[i];
    }
    cout << "\n";
}
```

```
int main() {  
    int data[] = {-2, 45, 0, 11, -9};  
    int size = sizeof(data) / sizeof(data[0]);  
    bubbleSort(data, size);  
    cout << "Sorted Array in Ascending Order:\n";  
    printArray(data, size);  
}
```

Lab Tasks:

Task1: Charge Account Validation

Write a program that lets the user enter a charge account number. The program should determine if the number is valid by checking for it in the following list:

5658845 4520125 7895122 8777541 8451277 1302850

8080152 4562555 5552012 5050552 7825877 1250255

1005231 6545231 3852085 7576651 7881200 4581002

The list of numbers above should be initialized in a single-dimensional array. A simple linear search should be used to locate the number entered by the user. If the user enters a number that is in the array, the program should display a message saying that the number is valid. If the user enters a number that is not in the array, the program should display a message indicating that the number is invalid.

Task 2:

A lottery ticket buyer purchases 10 tickets a week, always playing the same 10 5-digit “lucky” combinations. Write a program that initializes an array with these numbers and then lets the player enter this week’s winning 5-digit number.

The program should perform a linear search through the list of the player’s numbers and report whether or not one of the tickets is a winner this week. Here are the numbers:

13579, 26791, 26792, 33445, 55555

62483 77777 79422 85647 93121

Task 3:

Modify the program you wrote for Task 2 (Lottery Winners) so it performs a binary search instead of a linear search.

Task 4:

Modify the program you wrote for Task 1 (Charge Account Validation) so it performs a binary search to locate valid account numbers. Use the selection sort algorithm to sort the array before the binary search is performed.

Task 5:

Write a program that uses two identical arrays of at least 20 integers. It should call a function that uses the bubble sort algorithm to sort one of the arrays in ascending order. The function should keep a count of the number of exchanges it makes. The program then should call a function that uses the selection sort algorithm to sort the other array. It should also keep count of the number of exchanges it makes. Display these values on the screen.