BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

Committed to Academic Excellence

LAB REPORT- 01 & 02

Course Title	: Algorithms Lab
Course Code	: CSE 242

Submitted By :

Name	: Md. Mursalin Hasan Nirob
ID No	: 21225103423
Intake	: 49
Section	: 10
Program	: B.Sc. Engg. in CSE

Submitted To :

Name : Faria Binte Kader

Lecturer

Department of : CSE

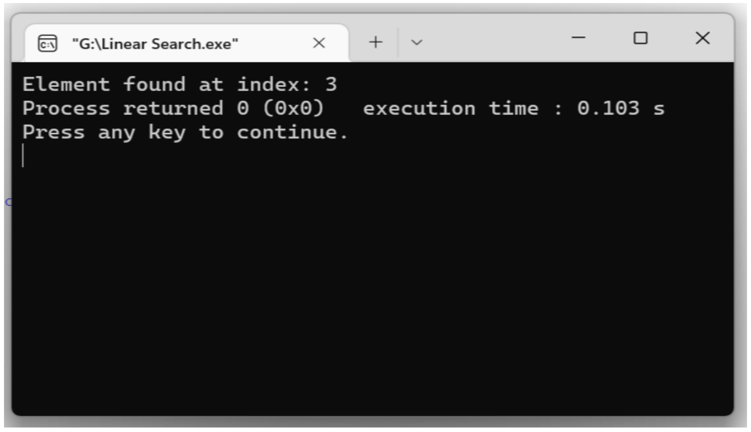
Bangladesh University of Business & Technology

Date of Submision: 25/07/2023

Task No: 01
Name of the Task : Linear Search
Code:

```
#include <iostream>
using namespace std;
int search(int arr[], int n, int x) {
  for (int i = 0; i < n; i++)
    if (arr[i] == x)
      return i;
  return -1;
}
int main() {
  int arr[] = {2, 4, 0, 1, 9};
  int x = 1;
  int n = sizeof(arr) / sizeof(arr[0]);
  int result = search(arr, n, x);
  (result == -1) ?
cout << "Element not found" : cout << "Element found at index: " << result;
}
```

Output:



```
Element found at index: 3
Process returned 0 (0x0)   execution time : 0.103 s
Press any key to continue.
```

Conclusion:
Linear search (Searching algorithm) which is used to find whether a given number is present in an array and if it is present then at what location it occurs. It is also known as sequential search. It is straightforward and works as follows: We keep on comparing each element with the element to search until it is found or the list ends.

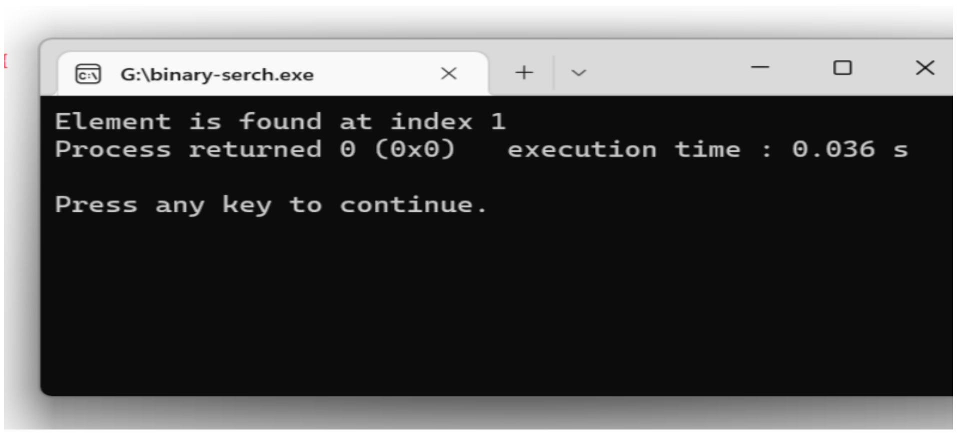
Task No: 02

Name of the Task : Binary Search

Code:

```
#include <iostream>

using namespace std;

int binarySearch(int arr[], int x, int l, int h) {

  while (l <= h) {

    int mid = l + (h - l) / 2;

    if (arr[mid] == x)

      return mid;

    if (arr[mid] < x)

      l = mid + 1;

    else

      h = mid - 1;

  }

 return -1;

}

   int main(void) {

  int arr[] = {3, 4, 5, 6, 7, 8, 9};

  int x = 4;

  int n = sizeof(arr) / sizeof(arr[0]);

  int result = binarySearch(arr, x, 0, n - 1);

  if (result == -1)

    printf("Not found");

  else

    printf("Element is found at index %d", result);

}
```

Output:



```
Element is found at index 1
Process returned 0 (0x0)   execution time : 0.036 s

Press any key to continue.
```

Conclusion:

Binary Search is a method to find the required element in a sorted array by repeatedly halving the array and searching in the half. This method is done by starting with the whole array. Then it is halved. If the required data value is greater than the element at the middle of the array, then the upper half of the array is considered. Otherwise, the lower half is considered. This is done continuously until either the required data value is obtained or the remaining array is empty.

Task No: 03

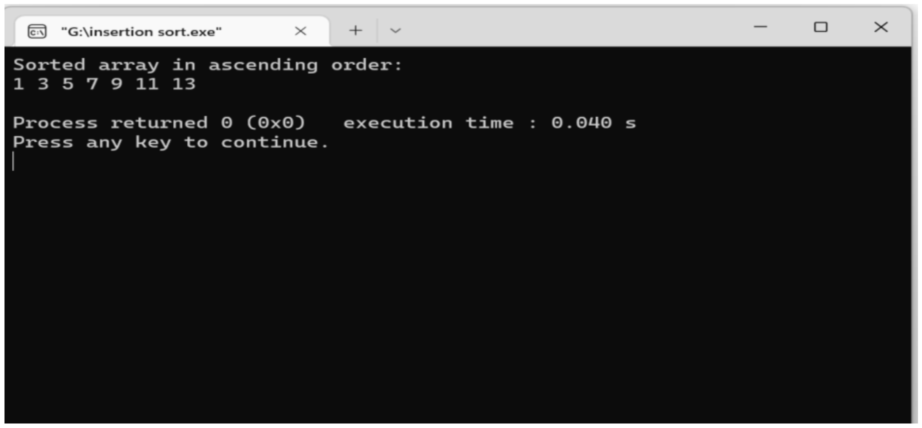
Name of the Task : Insertion Sort

Code:

```
#include <iostream>
using namespace std;
void printArray(int array[], int size) {
  for (int i = 0; i < size; i++) {
    cout << array[i] << " ";
  }
  cout << endl;
}

void insertionSort(int array[], int size) {
  for (int step = 1; step < size; step++) {
    int key = array[step];
    int j = step - 1;
    while (key < array[j] && j >= 0) {
      array[j + 1] = array[j];
      --j;
    }
    array[j + 1] = key;
  }
}
```

```
int main() {
  int data[] = {9, 5, 1, 7, 3 ,11,13};
  int size = sizeof(data) / sizeof(data[0]);
  insertionSort(data, size);
  cout << "Sorted array in ascending order:\n";
  printArray(data, size);
}
```

Output:



```
Sorted array in ascending order:
1 3 5 7 9 11 13

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

Conclusion:

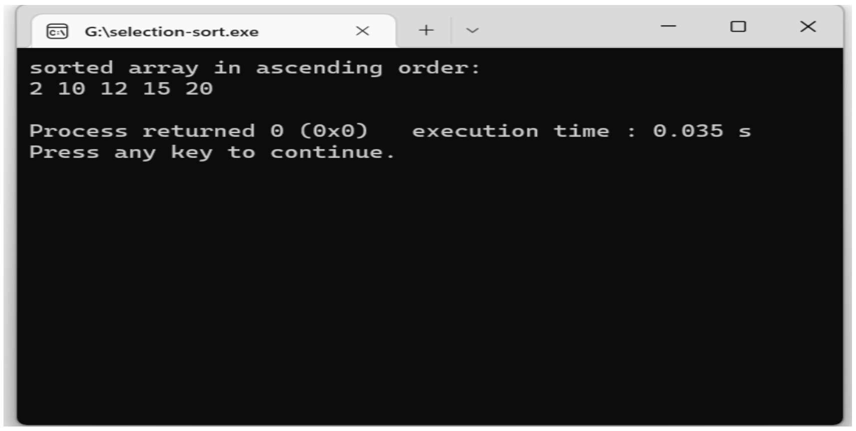
Insertion sort is a sorting algorithm that places an unsorted element of an array at its suitable place with each iteration. It works similar to how we sort the playing cards in our hands during a card game – the first card is always assumed to be sorted already. Then an unsorted card is selected, and if it is greater than the first card, we place it to its right, otherwise left. Then, we look at the next unsorted card and place it in its correct position based on its value.

Task No: 04
Name of the Task : Selection Sort

Code:

```
#include <iostream>
using namespace std;
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}
void printArray(int array[], int size) {
  for (int i = 0; i < size; i++) {
    cout << array[i] << " ";
  }
  cout << endl;
}

void selectionSort(int array[], int size) {
  for (int step = 0; step < size - 1; step++) {
    int min_idx = step;
    for (int i = step + 1; i < size; i++) {
      if (array[i] < array[min_idx])
        min_idx = i;
    }
    swap(&array[min_idx], &array[step]);
  }
}

int main() {
  int data[] = {20, 12, 10, 15, 2};
  int size = sizeof(data) / sizeof(data[0]);
  selectionSort(data, size);
  cout << "sorted array in ascending order:\n";
  printArray(data, size);
}
```

Output:



G:\selection-sort.exe

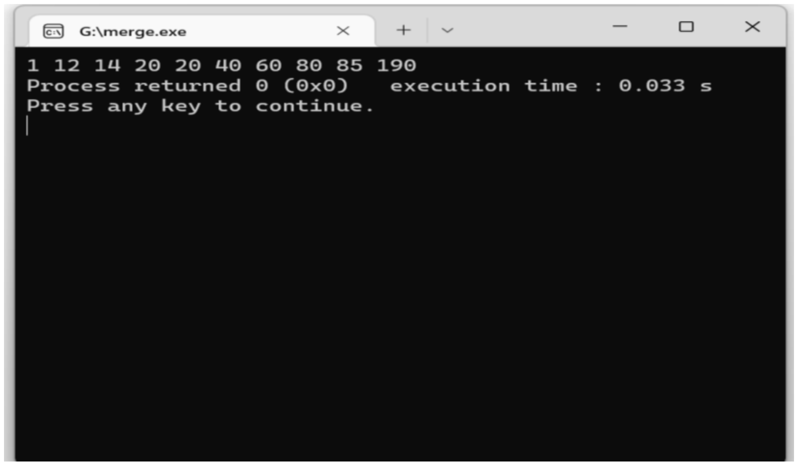
```
sorted array in ascending order:
2 10 12 15 20

Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

Conclusion:

Selection sort is a sorting algorithm that selects the smallest element from an unsorted list and sets it at the beginning of the unsorted list in each iteration. It sorts an array by repeatedly choosing the smallest member from the unsorted segment and placing it at the beginning.

Task No: 05
Name of the Task : Merge Sort
Code:
```
#include <iostream>
using namespace std;
void merge(int arr[], int l, int mid, int r)
{
    int b[r - l + 1];
    int l1 = l, l2 = mid + 1, i = 0;
    while (l1 <= mid && l2 <= r)
    {
        if (arr[l1] <= arr[l2])
            b[i++] = arr[l1++];
        else
            b[i++] = arr[l2++];
    }
    while (l1 <= mid)
    {
        b[i++] = arr[l1++];
    }

    while (l2 <= r)
    {
        b[i++] = arr[l2++];
    }

    int j = 0;
    for (i = l; i <= r; i++)
        arr[i] = b[j++];
}

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


int main()
{
    int i = 0;
    int arr[] = {14, 12, 1, 20, 20, 40, 60, 80, 85, 190};
    int size = sizeof(arr) / sizeof(arr[0]);
    int l = 0, r = size - 1;
    mergeSort(arr, l, r);
    for (i = 0; i < size; i++)
    {
        cout<<arr[i]<<" ";
    }
    return 0;
}
```

Output:



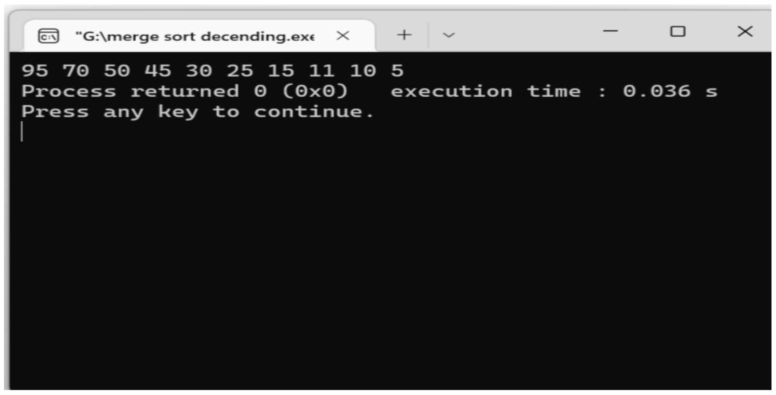
```
1 12 14 20 20 40 60 80 85 190
Process returned 0 (0x0)   execution time : 0.033 s
Press any key to continue.
```

Conclusion:

Merge sort is an efficient sorting algorithm that divides the input array into two parts of a sub-list. It sorts each sub-list and then merges them. To sort each sub-list, the algorithm divides the sub-list in two again. If this sub-list of the sub-list can be divided into two halves, then it will be. The idea of the merge sort is that it will merge all the sorted sub-lists into the fully sorted list.

Task No: 06
Name of the Task : Merge Sort Descending
Code:
```
#include <iostream>
using namespace std;
void merge(int arr[], int l, int mid, int r)
{
    int b[r - l + 1];
    int l1 = l, l2 = mid + 1, i = 0;
    while (l1 <= mid && l2 <= r)
    {
        if (arr[l1] >= arr[l2])
            b[i++] = arr[l1++];
        else
            b[i++] = arr[l2++];
    }
    while (l1 <= mid)
    {
        b[i++] = arr[l1++];
    }
    while (l2 <= r)
    {
        b[i++] = arr[l2++];
    }
    int j = 0;
    for (i = l; i <= r; i++)
        arr[i] = b[j++];
}
void mergeSort(int arr[], int l, int r)
{
    int mid;
    if (l < r)
    {
        mid = (l + r) / 2;
        mergeSort(arr, l, mid);
        mergeSort(arr, mid + 1, r);
        merge(arr, l, mid, r);
    }
}
int main()
{
    int i = 0;
    int arr[] = {15, 11, 5, 25, 30, 45, 50, 70, 95, 10};
    int size = sizeof(arr) / sizeof(arr[0]);
    int l = 0, r = size - 1;
    mergeSort(arr, l, r);
    for (i = 0; i < size; i++)
    {

cout<<arr[i]<<" ";
    }
    return 0;
}
```

Output:



"G:\merge sort decending.exe

```
95 70 50 45 30 25 15 11 10 5
Process returned 0 (0x0)   execution time : 0.036 s
Press any key to continue.
```

Conclusion:

Merge Sort in descending order is a sorting algorithm that arranges elements in a list or array in non-increasing order. In other words, the larger elements will appear before the smaller elements in the sorted output. The Merge Sort algorithm is based on the Divide and Conquer paradigm. It recursively divides the input array into two halves until each subarray contains only one element .Then, it merges the sorted subarrays back together while maintaining the descending order. The merging step is the core of the Merge Sort algorithm.