CL2001 Data Structures Lab	Lab 04 Linear, Binary and ; Interpolation Search, Elementary Sorting Techniques (Bubble, Selection, Insertion)

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Fall 2025

Lab Content

1. Bubble
2. Insertion
3. Selection
4. binary search
5. interpolation search
6. Linear Search

BUBBLE SORT:

Bubble Sort, the two successive strings arr[i] and arr[i+1] are exchanged whenever arr[i]> arr[i+1]. The larger values sink to the bottom and hence called sinking sort. At the end of each pass, smaller values gradually “bubble” their way upward to the top and hence called bubble sort.

Example:

```
//you need to take input from user and display the unsorted array.
//sort the array using the following steps .

for (int i = 0; i < n; i++) {
   for (int j = 0; j < n - 1; j++) {
      if (a[j] > a[j + 1]) {
         // Swap elements if they are in the wrong order
         int temp = a[j];
         a[j] = a[j + 1];
         a[j + 1] = temp;
      }
   }
}
//display your sorted array.
```

SELECTION SORT:

Key Points:

void selectionSort(int *array, int size) {

Find the smallest element in the array and exchange it with the element in the first position.

Find the second smallest element in the array and exchange it with the element in the second position.

Continue this process until done.

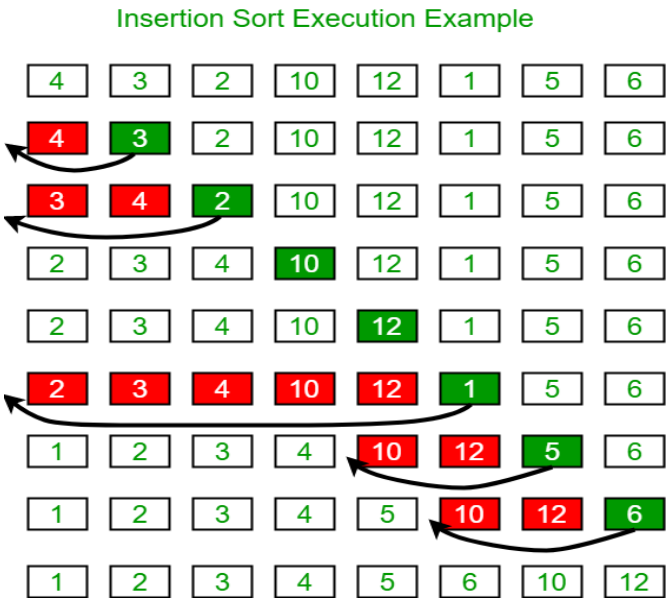
}

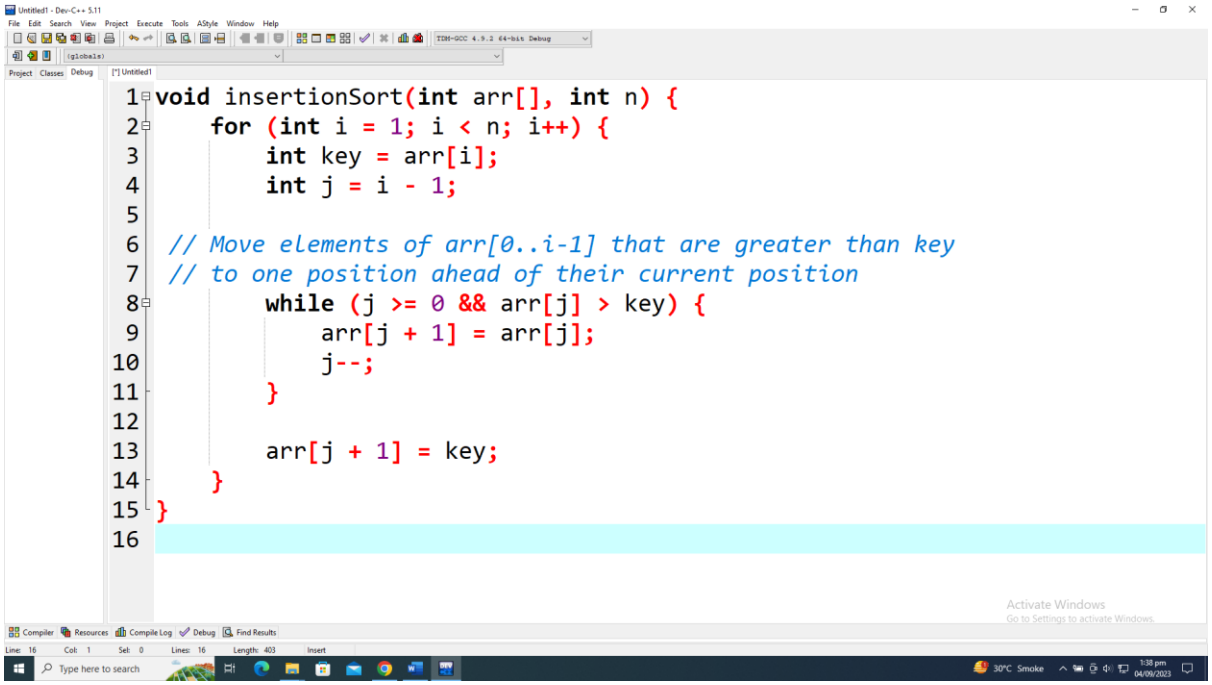
- Example: (5,10,3,5,4)

```
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int min_index = i;

        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_index]) {
                min_index = j;
            }
        }

        int temp = arr[i];
        arr[i] = arr[min_index];
        arr[min_index] = temp;
    }
}
```

INSERTION SORT:

Insertion Sort is a sorting algorithm that gradually builds a sorted sequence by repeatedly inserting unsorted elements into their appropriate positions. In each iteration, an unsorted element is taken and placed within the sorted portion of the array. This process continues until the entire array is sorted.





```
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

 // Move elements of arr[0..i-1] that are greater than key
 // to one position ahead of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = key;
    }
}
```

5,10,3,2

SEARCHING ALGORITHMS:

LINEAR SEARCH ALGORITHM: Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

```
int i;
for (i = 0; i < N; i++)
        if (arr[i] == x)
                return i;
}
```

BINARY SEARCH ALGORITHM:

Binary Search is a searching algorithm for finding an element's position in a sorted array. In this approach, the element is always searched in the middle of a portion of an array. Binary search can be implemented only on a sorted list of items. If the elements are not sorted already, we need to sort them first.

```
while (left <= right) {
   int mid = left + (right - left) / 2;
   if (arr[mid] == key) {
      return mid;
   }
   else if (arr[mid] < key) {
      left = mid + 1;
   }
   else {
      right = mid - 1;
   }
}
return -1;
```

INTERPOLATION SEARCH:

The Interpolation Search is an improvement over Binary Search for instances, where the values in a sorted array are uniformly distributed. Interpolation constructs new data points within the range of a discrete set of known data points.

```cpp
#include <iostream>

int interpolationSearch(int arr[], int size, int x) {
    int low = 0;
    int high = size - 1;

    while (low <= high && x >= arr[low] && x <= arr[high]) {
        if (low == high) {
            if (arr[low] == x) return low;
            return -1;
        }

        // Estimate the position
        int pos = low + ((x - arr[low]) * (high - low)) / (arr[high] - arr[low]);

        // Check if the estimated position has the target value
        if (arr[pos] == x) return pos;

        // If the target value is greater, ignore the left half
        if (arr[pos] < x) low = pos + 1;
        // If the target value is smaller, ignore the right half
        else high = pos - 1;
    }
    return -1;
}

int main() {
    int arr[] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
    int size = sizeof(arr) / sizeof(arr[0]);

    int x;
    std::cout << "Enter the value to search: ";
    std::cin >> x;

    int index = interpolationSearch(arr, size, x);

```

COMPARATIVE TABLE OF SORTING AND SEARCHING ALGORITHMS

Algorithm	Type	Best Case	Worst Case	Average Case	Space Complexity	Key Characteristics
Bubble Sort	Sorting	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Easy to implement, inefficient for large datasets. Values "bubble" to the correct position in each pass.
Selection Sort	Sorting	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	Simple but inefficient for large datasets. Swaps are reduced compared to Bubble Sort.
Insertion Sort	Sorting	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Efficient for small or nearly sorted arrays. Performs better than Bubble or Selection Sort on small datasets.
Shell Sort	Sorting	O(nlogn)	$O(n^2)$	O(n log n)	O(1)	Gap reduces in multiple passes, improves efficiency over Insertion Sort.
Comb Sort	Sorting	O(nlogn)	$O(n^2)$	$O(n^2)$	O(1)	Reduces the number of comparisons compared to Bubble Sort. More efficient for larger arrays.
Linear Search	Searching	O(1)	O(n)	O(n)	O(1)	Simple, but inefficient for large datasets as it doesn't take advantage of any sorting.
Binary Search	Searching	O(1)	O(log n)	O(log n)	O(1)	Efficient on sorted arrays. It divides the search space by half at each step, significantly reducing the search time.
Interpolation Search	Searching	O(1)	O(n)	O(log log n)	O(1)	Efficient when elements are uniformly distributed, but can degrade to linear search when the distribution is not uniform.

Lab 4 Exercise

Task#1 Write a Program that ask user to enter 10 elements and finds the 4 minimum elements from given array using selection Sort.

Task#2: Let arr[9] = { 20, 12, 15, 2, 10, 1, 13, 9, 5} now sort the array in such a way that maximum element must be at middle of the array and rest of array must be sorted in ascending order do this using insertion sort.

Sorted array: 1 2 5 9 20 10 12 13 15

Task#3: Given an array of strings arr[]. Sort given strings using Bubble Sort and display the sorted array.
Input: string arr[] = {"banana", "apple", "cherry", "date", "grape"};
Output: apple banana cherry date grape

Task#4: Given an unsorted array that may contain duplicates. Write a function that returns true if the array contains duplicates.

Task#5: Given an array with birth years of children born in 2022, 2023, and 2024, the task is to sort the array so that all children born in 2022 come first, followed by those born in 2023, and finally those born in 2024.
Input: {2022, 2023, 2024, 2022, 2023, 2024}
Output: {2022, 2022, 2023, 2023, 2024, 2024}
Explanation: {2022, 2022, 2023, 2023, 2024, 2024} shows that all the 2022 birth years come first, followed by the 2023 , and then all the 2024 birth years at the end.

Task#6: . Implement the array given in the heading of binary search topic, sort it and find the value corresponding to to your last two digits of the roll number
(if its not in the array add a value somewhere in between the array) and find it via binary search.

Task#7: From array mentioned below find the following names and their index using binary search as well as linear search and count the number of steps on each finding element by using both techniques and analyze which technique is getting more time:

- Aftab
- Rizwan
- Tariq

Ahmed	Ali	Basit	Karim	Rizwan	Sarwar	Tariq	Taufiq	Yasin	Zulfiqar
0	1	2	3	4	5	6	7	8	9