**Data Structures Lab**

***Session 4***

**Course:** Data Structures (CS218) **Semester:** Fall 2024

**Instructor:** Shafique Rehman  **T.A:** N/A

**Note:**

* + - * Lab manual cover following below elementary sorting and searching algorithms

**{Bubble, insertion, selection, shell sort, comb sort, binary search, interpolation search**

**, Linear Search}**

* Maintain discipline during the lab.
* Just raise hand if you have any problem.
* Completing all tasks of each lab is compulsory.
* Get your lab checked at the end of the session.

**Bubble Sort:**

1. 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) {

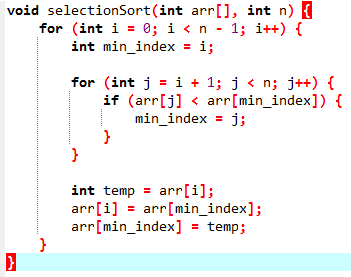
Find the smallest element in the array and exchange it with the element in the first position.

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

Continue this process until done.

}

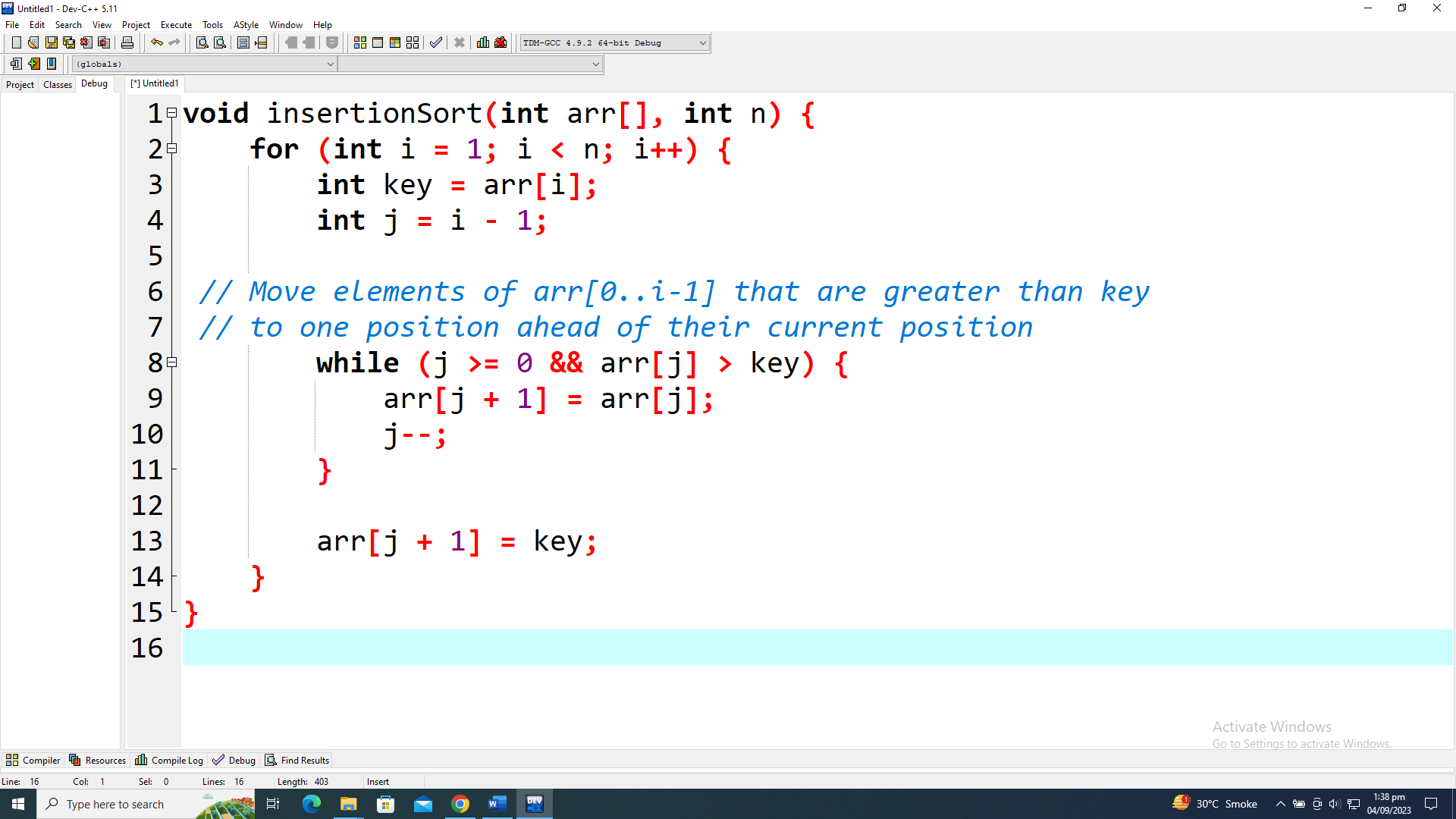
Example: (5,10,3,5,4)



**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.





**5,10,3,2**

**Shell Sort :**

shellSort(array, size)

for interval i <- size/2n down to 1

for each interval "i" in array

sort all the elements at interval "i"

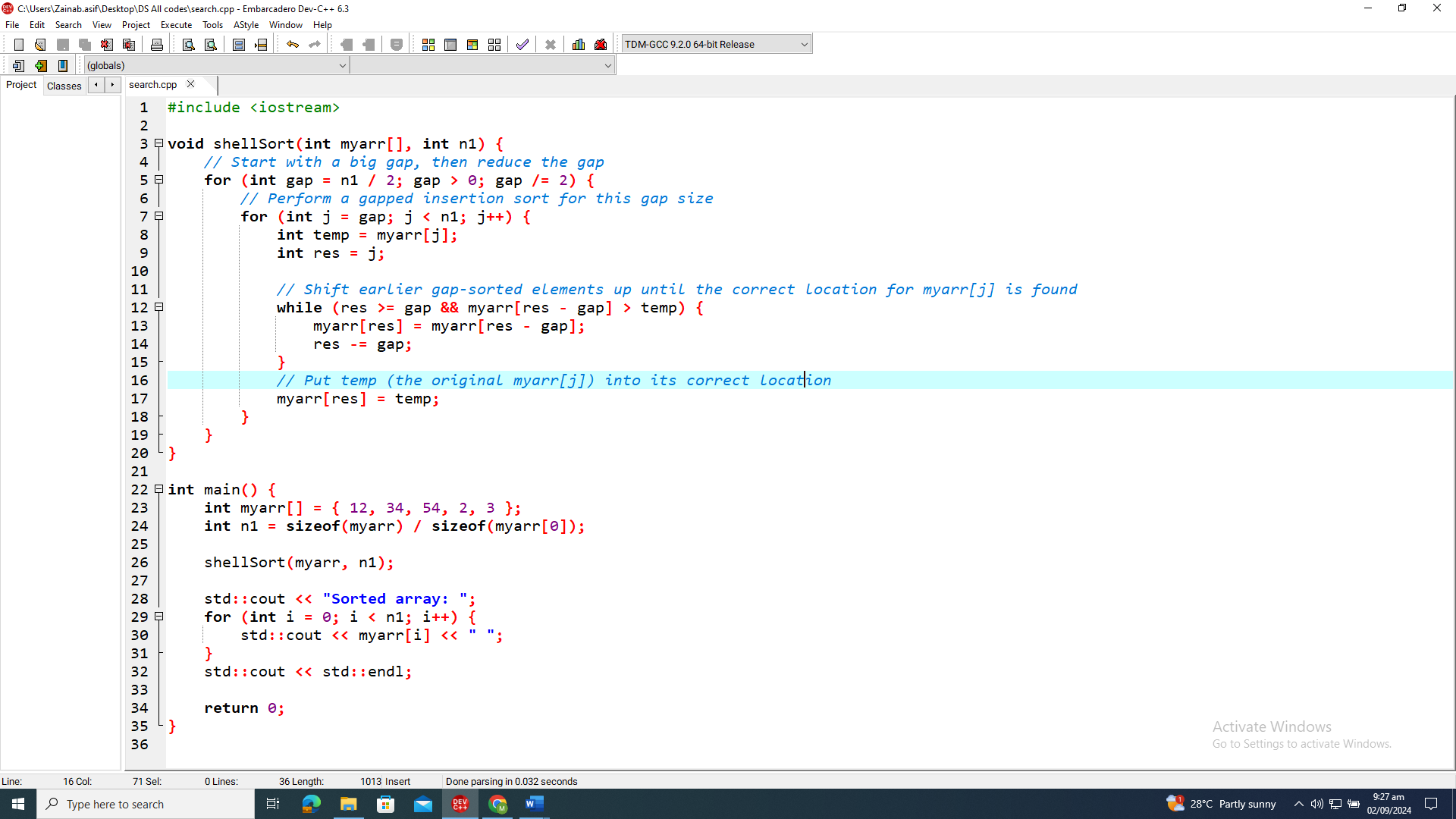
end shellSort

**Problem Description**

1. Shell sort is an improvement over insertion sort.  
2. It compares the element separated by a gap of several positions.  
3. A data element is sorted with multiple passes and with each pass gap value reduces.

**Problem Solution**

1. Assign gap value as half the length of the array.  
2. Compare element present at a difference of gap value.  
3. Sort them and reduce the gap value to half and repeat.  
4. Display the result.  
5. Exit.



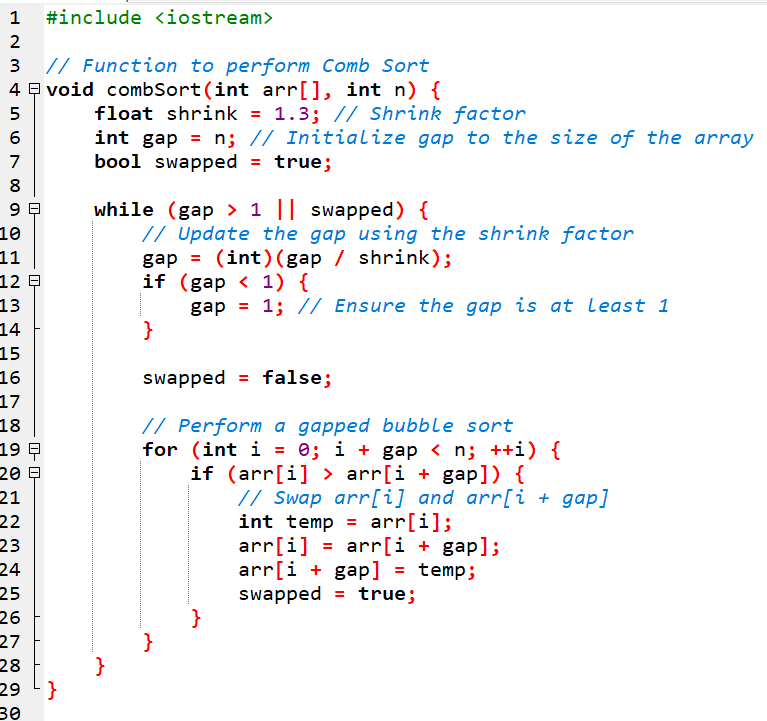
Key Points:

1.Take input of data.  
2. Create and call function name as ShellSort() contains two arguments.(‘arr’ the array of data and ‘n’ the number of values).  
3. Implement Sorting algorithm using nested for loop.  
4. The first loop will run on ‘gap’ Which decides the gap value to compare two elements.  
5. The second loop will run on ‘j’ from j to n.  
6. The third loop will run on ‘res’ and sort the element having “gap” as a gap between their index.  
7. Switch the values if arr[res] < arr[res-gap].

8. Return to main and display the result.

**Comb Sort:**

Comb Sort is an efficient sorting algorithm designed to improve upon Bubble Sort by reducing the number of comparisons and swaps required. It works by initially sorting elements that are far apart and gradually reducing the gap between elements being compared. The core idea of Comb Sort is to use a "gap" that decreases over time, which allows the algorithm to move elements into their correct positions more quickly compared to traditional sorting methods. As the gap decreases, the algorithm performs a final pass with a gap of 1, similar to Bubble Sort, to ensure the entire array is sorted. This method helps in reducing the time complexity compared to simple Bubble Sort, especially for larger arrays.



**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;

}

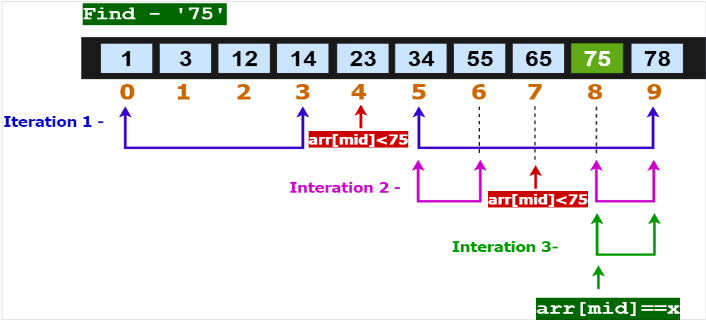
else {

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.

// The idea of formula is to return higher value of pos

// when element to be searched is closer to arr[hi]. And

// smaller value when closer to arr[lo]

arr[] ==> Array where elements need to be searched

x ==> Element to be searched

lo ==> Starting index in arr[]

hi ==> Ending index in arr[]



