CL2001 Data Structures Lab	Lab 04 Sorting Techniques (Bubble, Selection, Insertion, Radix sort)

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Fall 2025

Lab Content

1. Bubble
2. Insertion
3. Selection
4. Radix Sort

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:

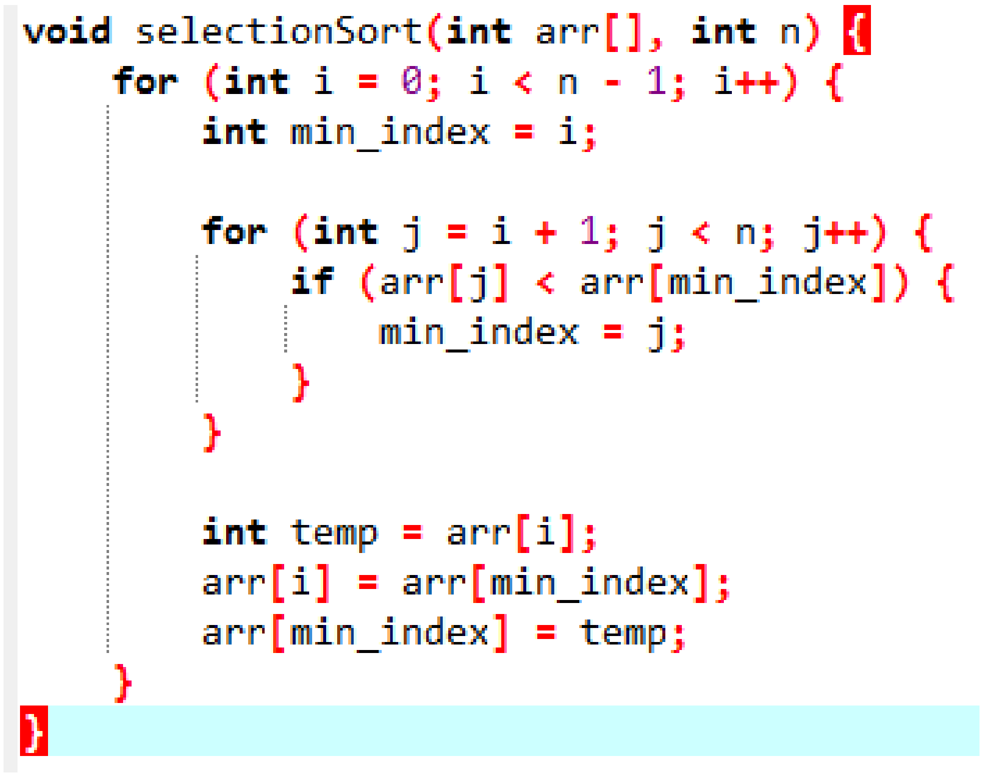
```
//you need to take input from user and display the unsorted array.
//sort the array using the following steps .

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)

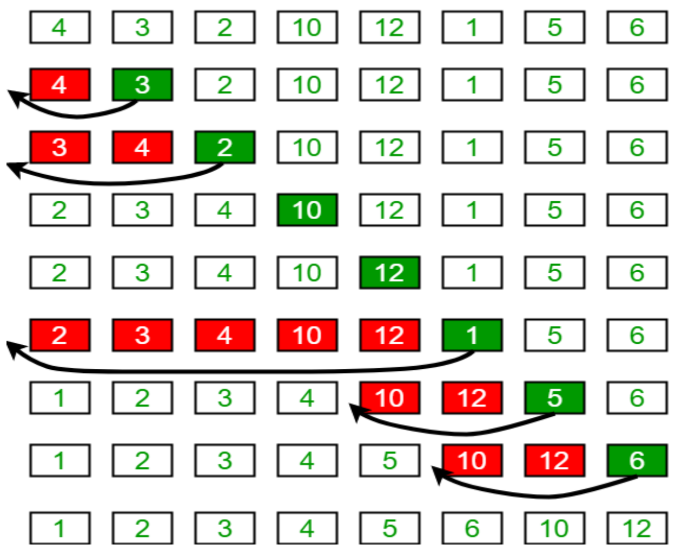


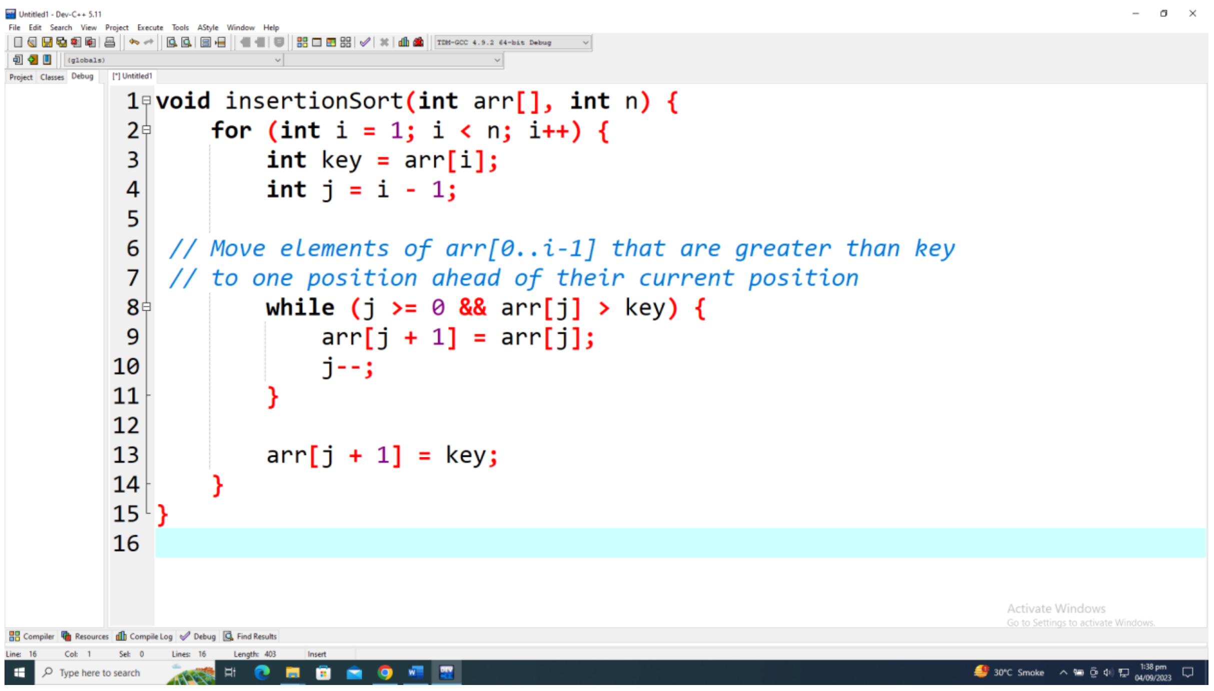
```
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int min_index = i;

        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.

Insertion Sort Execution Example




Untitled1 - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Debug
(globals)
Project Classes Debug
[*] Untitled1
Compiler Resources Compile Log Debug Find Results
Line: 16 Col: 1 Sel: 0 Lines: 16 Length: 403 Insert
Type here to search
30°C Smoke
1:38 pm 04/09/2023

```
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

Radix Sort (Advance Sorting) :

```
RadixSort(arr[], n):
    maxNum = maximum element in arr
    exp = 1   // 1, 10, 100, ... (digit position)
    while (maxNum / exp > 0):
        CountingSortByDigit(arr, n, exp)
        exp = exp * 10

CountingSortByDigit(arr[], n, exp):
    output[n]
    count[10] = {0}

    // Count occurrences of each digit
    for i = 0 to n-1:
        digit = (arr[i] / exp) % 10
        count[digit]++

    // Make count cumulative
    for i = 1 to 9:
        count[i] += count[i-1]

    // Build output array (stable)
    for i = n-1 downto 0:
        digit = (arr[i] / exp) % 10
        output[count[digit] - 1] = arr[i]
        count[digit]--

    // Copy back to arr[]
    for i = 0 to n-1:
        arr[i] = output[i]
```

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.
Radix Sort	Sorting	$O(n \cdot d)$	$O(n \cdot d)$	$O(n \cdot d)$	$O(n + b)$	Efficient for large datasets with integers or strings of limited length. Performs better than comparison-based algorithms when the key range is not excessively large.

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:
A university admissions office has received thousands of student applications, each identified by a unique 6-digit application ID (e.g., 452913, 100234, 987654).

The office needs to sort all application IDs in ascending order to generate a merit list quickly. Since the IDs are integers with a fixed number of digits, Write a program to sort the application IDs using appropriate and efficient sorting algorithm.