

Activity No. 7.1	
Sorting Algorithms	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: 10/16/24
Section: CPE21S4	Date Submitted: 10/17/24
Name(s): SANTOS, ANDREI R.	Instructor: Professor Maria Rizette Sayo

6. Output

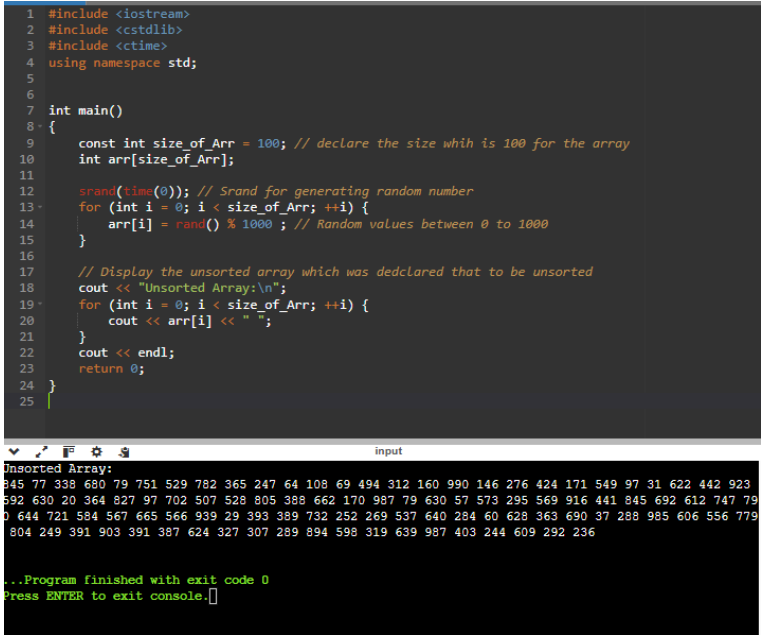
Code + Console Screenshot

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
    const int size_of_Arr = 100; // declare the size which is 100 for the array
    int arr[size_of_Arr];

    srand(time(0)); // Srand for generating random number
    for (int i = 0; i < size_of_Arr; ++i) {
        arr[i] = rand() % 1000 ; // Random values between 0 to 1000
    }

    // Display the unsorted array which was declared that to be unsorted
    cout << "Unsorted Array:\n";
    for (int i = 0; i < size_of_Arr; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```



Observations	This code runs to create a hundred random values of numbers between 0 to 1000 in the element of the array which is unsorted.
--------------	--

Table 7-1. Array of Values for Sort Algorithm Testing

Code + Console Screenshot	<pre> #ifndef BUBBLESORT_H #define BUBBLESORT_H void bubbleSort(int arr[], int arrSize); #endif // BUBBLESORT_H #include <iostream> #include <cstdlib> #include <ctime> #include "BubbleSort.h" using namespace std; // Implementation of bubbleSort void bubbleSort(int arr[], int arrSize) { // Outer loop for each element for (int i = 0; i < arrSize - 1; ++i) { // Inner loop for comparing adjacent elements for (int j = 0; j < arrSize - i - 1; ++j) { // If the current element is greater than the next, swap them if (arr[j] > arr[j + 1]) { swap(arr[j], arr[j + 1]); } } } } int main() { const int size_of_Arr = 100; int arr[size_of_Arr]; // Seed the random number generator srand(time(0)); // Fill the array with random numbers for (int i = 0; i < size_of_Arr; ++i) { arr[i] = rand() % 1000; } // Display the unsorted array cout << "This is the unsorted Array:\n"; for (int i = 0; i < size_of_Arr; ++i) { cout << arr[i] << " "; } cout << endl; // Call bubble sort to sort the array </pre>
---------------------------	--

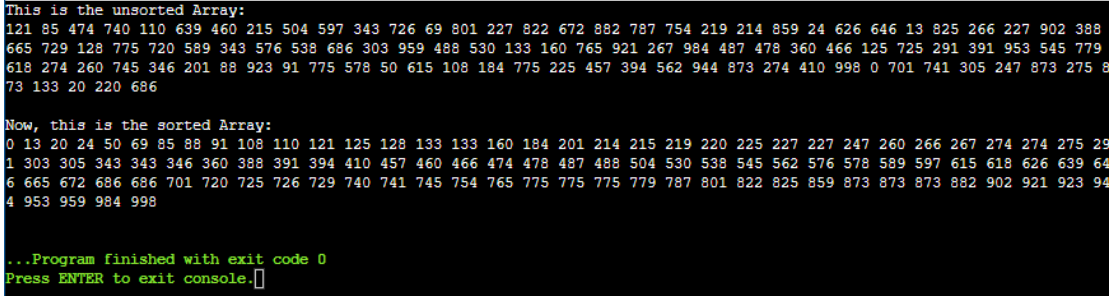
	<pre> bubbleSort(arr, size_of_Arr); // Display the sorted array cout << "\nNow, this is the sorted Array:\n"; for (int i = 0; i < size_of_Arr; ++i) { cout << arr[i] << " "; } cout << endl; return 0; } </pre>  <pre> This is the unsorted Array: 121 85 474 740 110 639 460 215 504 597 343 726 69 801 227 822 672 882 787 754 219 214 859 24 626 646 13 825 266 227 902 388 665 729 128 775 720 589 343 576 538 686 303 959 488 530 133 160 765 921 267 984 487 478 360 466 125 725 291 391 953 545 779 618 274 260 745 346 201 88 923 91 775 578 50 615 108 184 775 225 457 394 562 944 873 274 410 998 0 701 741 305 247 873 275 8 73 133 20 220 686 Now, this is the sorted Array: 0 13 20 24 50 69 85 88 91 108 110 121 125 128 133 133 160 184 201 214 215 219 220 225 227 227 247 260 266 267 274 274 275 29 1 303 305 343 343 346 360 388 391 394 410 457 460 466 474 478 487 488 504 530 538 545 562 576 578 589 597 615 618 626 639 64 6 665 672 686 686 701 720 725 726 729 740 741 745 754 765 775 775 779 787 801 822 825 859 873 873 873 882 902 921 923 94 4 953 959 984 998 ...Program finished with exit code 0 Press ENTER to exit console. </pre>
Observations	<p>The code uses the Bubble Sort algorithm to sort an array by repeatedly swapping the elements if they are in the wrong order. It creates an array of random numbers and then uses Bubble Sort to arrange them. After sorting, it prints the sorted array as we can see on the row for the sorted array.</p>

Table 7-2. Bubble Sort Technique

Code + Console Screenshot	<pre> #ifndef SELECTIONSORT_H #define SELECTIONSORT_H template <typename T> int Small_elmnt(T A[], int K, const int arrSize); template <typename T> void selc_sort(T arr[], const int N); #endif // SELECTIONSORT_H #include <iostream> #include <cstdlib> #include <ctime> #include "selectionsort.h" </pre>
---------------------------	---

```

using namespace std;

// Finding the position of the smallest element in the array
template <typename T>
int Small_elmnt(T A[], int K, const int arrSize) {
    int position = K;
    for (int j = K + 1; j < arrSize; j++) {
        if (A[j] < A[position]) {
            position = j;
        }
    }
    return position;
}

// Implementation of selection sort
template <typename T>
void selc_sort(T arr[], const int N) {
    for (int i = 0; i < N - 1; i++) {
        int POS = Small_elmnt(arr, i, N);
        swap(arr[i], arr[POS]);
    }
}

int main() {
    const int size_of_Arr = 100;
    int arr[size_of_Arr];

    // Seed the random number generator
    srand(time(0));
    // Fill the array with random numbers
    for (int i = 0; i < size_of_Arr; ++i) {
        arr[i] = rand() % 1000;
    }

    // Display the unsorted array
    cout << "This is the unsorted Array:\n";
    for (int i = 0; i < size_of_Arr; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    // Call selection sort
    selc_sort(arr, size_of_Arr);

    // Display the sorted array
    cout << "Now, this is the sorted Array:\n";
    for (int i = 0; i < size_of_Arr; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

```

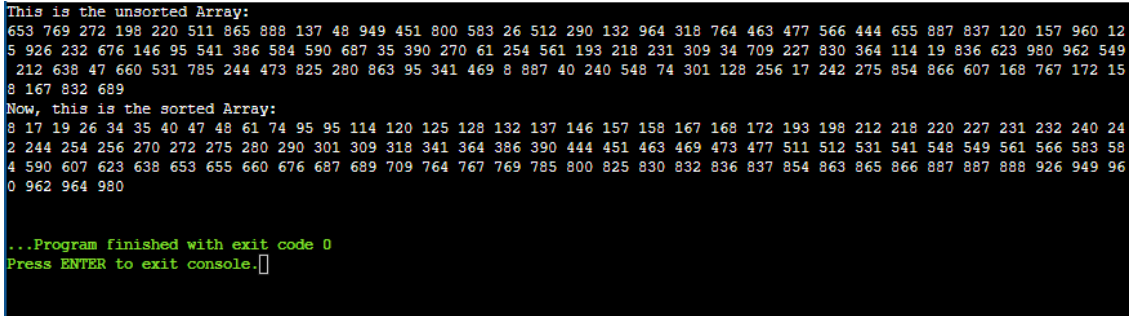
	<pre> return 0; } </pre>  <p>This is the unsorted Array: 653 769 272 198 220 511 865 888 137 48 949 451 800 583 26 512 290 132 964 318 764 463 477 566 444 655 887 837 120 157 960 12 5 926 232 676 146 95 541 386 584 590 687 35 390 270 61 254 561 193 218 231 309 34 709 227 830 364 114 19 836 623 980 962 549 212 638 47 660 531 785 244 473 825 280 863 95 341 469 8 887 40 240 548 74 301 128 256 17 242 275 854 866 607 168 767 172 15 8 167 832 689 Now, this is the sorted Array: 8 17 19 26 34 35 40 47 48 61 74 95 95 114 120 125 128 132 137 146 157 158 167 168 172 193 198 212 218 220 227 231 232 240 24 2 244 254 256 270 272 275 280 290 301 309 318 341 364 386 390 444 451 463 469 473 477 511 512 531 541 548 549 561 566 583 58 4 590 607 623 638 653 655 660 676 687 689 709 764 767 769 785 800 825 830 832 836 837 854 863 865 866 887 887 888 926 949 96 0 962 964 980 ...Program finished with exit code 0 Press ENTER to exit console.[]</p>
Observations	<p>The code runs the Selection Sort algorithm, where it repeatedly finds the smallest value in the unsorted part of the array and swaps it with the first unsorted element. This ensures that the smallest elements are progressively moved to the start of the array, resulting in a sorted sequence.</p>

Table 7-3. Selection Sort Algorithm

Code + Console Screenshot	<pre> #ifndef INSERTIONSORT_H #define INSERTIONSORT_H template <typename T> void insertionSort(T arr[], const int N) { for (int K = 1; K < N; K++) { T temp = arr[K]; int J = K - 1; // Move elements that are greater than temp while (J >= 0 && arr[J] > temp) { arr[J + 1] = arr[J]; J--; } // Place temp in the correct position arr[J + 1] = temp; } } #endif // INSERTIONSORT_H #include <iostream> </pre>
---------------------------	--

```

#include <cstdlib>
#include <ctime>
#include "insertionsort.h" // Include the header here

using namespace std;

int main() {
    const int size_of_Arr = 100;
    int arr[size_of_Arr];

    // Seed the random number generator
    srand(time(0));
    // Fill the array with random numbers
    for (int i = 0; i < size_of_Arr; ++i) {
        arr[i] = rand() % 1000;
    }

    // Display the unsorted array
    cout << "This is the unsorted Array:\n";
    for (int i = 0; i < size_of_Arr; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    // Call insertion sort
    insertionSort(arr, size_of_Arr);

    // Display the sorted array
    cout << "Now, this is the sorted Array:\n";
    for (int i = 0; i < size_of_Arr; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}

```

```

This is the unsorted Array:
291 490 805 283 905 198 895 814 614 732 493 847 420 372 916 792 519 196 108 310 868 284 145 695 343 646 844 977 432 99 451 7
23 590 256 6 495 807 253 309 421 986 154 621 758 526 889 902 397 437 10 707 305 647 852 0 990 499 196 319 283 648 771 7 238
379 365 85 538 619 394 960 957 548 933 67 426 822 969 823 259 332 882 916 979 87 269 321 938 817 993 221 465 116 228 703 495
594 788 34 565
Now, this is the sorted Array:
0 6 7 10 34 67 85 87 99 108 116 145 154 196 196 198 221 228 238 253 256 259 269 283 283 284 291 305 309 310 319 321 332 343
365 372 379 394 397 420 421 426 432 437 451 465 490 493 495 495 499 519 526 538 548 565 590 594 614 619 621 646 647 648 695
703 707 723 732 758 771 788 792 805 807 814 817 822 823 844 847 852 868 882 889 895 902 905 916 916 933 938 957 960 969 977
979 986 990 993

...Program finished with exit code 0
Press ENTER to exit console.

```

Observations	The insertion sort algorithm sorts an array by taking each element and placing it into its correct position within the already sorted section of the array. It starts with the second element, compares it with the previous elements, and shifts right to the higher or larger elements. The current element is then inserted into its correct position. This process is repeated until the entire array is sorted in ascending order or turned into a sorted array.
--------------	---

Table 7-4. Insertion Sort Algorithm

7. Supplementary Activity

Deliverables:

• Pseudocode of Algorithm

START

INCLUDE necessary libraries

iostream, cstdlib, ctime

Bubb_sr.h (for bubble sort function)

INITIALIZE main function

SEED random number generator with current time

DEFINE constant size for votes array (size = 100)

DECLARE votes array of size 100

DECLARE voteCount array of size 6 initialized to 0 (index 0 unused)

DECLARE array of random congratulatory strings: // **(I just added this code for this activity)**

randomStrings = ["Nice One!", "Good Job!", "Excellent!", "Great!"]

LOOP from 0 to size - 1

GENERATE random vote (1 to 5) and ASSIGN to votes[i]

OUTPUT "This is the raw and unsorted count of votes:"

LOOP through votes array and PRINT each vote

LOOP from 0 to size - 1

INCREMENT voteCount[votes[i]] to count votes for each candidate

CALL bubbleSort function to sort votes array

OUTPUT "Now, this is the sorted count of votes:"

LOOP through votes array and PRINT each sorted vote

OUTPUT "The number of votes for Candidates 1 to 5:"

LOOP from 1 to 5

PRINT "Candidate [candidate]: [voteCount[candidate]] votes"

INITIALIZE winner variable to 1 (candidate 1)

LOOP from 2 to 5

IF voteCount[candidate] > voteCount[winner] THEN

SET winner to candidate

GENERATE random index (0 to 3) for congratulatory string // (I just added this code for this activity)

OUTPUT "Congratulations to the winner!"

PRINT "Candidate [winner] gaining with [voteCount[winner]] votes! [randomStrings[randomIndex]]"

END

• Screenshot of Algorithm Code

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  #include "Bubb_sr.h"
5
6  using namespace std;
7
8  int main()
9  {
10     srand(time(0));
11
12     const int size = 100;
13     int votes[size];
14     int voteCount[6] = {0};
15
16     // Array of random congratulatory strings ( I just wanted to add this for the winner.)
17     const char* randomStrings[] = {"Nice One!", "Good Job!", "Excellent!", "Great!"};
18
19     // Generate random votes
20     for (int i = 0; i < size; i++)
21     {
22         votes[i] = rand() % 5 + 1; // Random numbers from 1 to 5 because we have 5 candidates
23     }
```



```

25 // Output the unsorted count of votes
26 cout << "This is the raw and unsorted count of votes:\n";
27 for (int i = 0; i < size; i++)
28 {
29     cout << votes[i] << " ";
30 }
31 cout << endl;
32
33 // Count the votes for each from the 5 candidates
34 for (int i = 0; i < size; i++)
35 {
36     voteCount[votes[i]]++;
37 }
38
39 // Sort the votes using bubble sort
40 bubbleSort(votes, size);
41
42 // Output the sorted count of votes
43 cout << "\nNow, this is the sorted count of votes:\n";
44 for (int i = 0; i < size; i++)
45 {
46     cout << votes[i] << " ";
47 }
48 cout << endl;

```

```

49
50 // Output the vote counts for the candidates 1 to 5, showing their individual votes
51 cout << "\nThe number of votes for Candidates 1 to 5:\n";
52 for (int candidate = 1; candidate <= 5; candidate++)
53 {
54     cout << "Candidate " << candidate << ": " << voteCount[candidate] << " votes\n";
55 }
56
57 // This is where we find the highest count of the votes from the 5 candidates
58 int winner = 1; // Start with candidate 1 as the winner
59 for (int candidate = 2; candidate <= 5; candidate++)
60 {
61     if (voteCount[candidate] > voteCount[winner])
62     {
63         winner = candidate;
64     }
65 }
66
67 // Select a random congratulatory string (I just add this for the winner, for winning)
68 int randomIndex = rand() % 4; // Random index from 0 to 3 because I have 4 choices of random strings
69
70 // Output the winning candidate
71 cout << "\nCongratulations to the winner!\n";
72 cout << "Candidate " << winner << " gaining with " << voteCount[winner] << " votes! " << randomStrings[randomIndex];
73 return 0;
74 }
75

```

```

1  #ifndef BUBBLESORT_H
2  #define BUBBLESORT_H
3
4  void bubbleSort(int arr[], int size) {
5      // Bubble sort algorithm
6      for (int i = 0; i < size - 1; i++) {
7          for (int j = 0; j < size - i - 1; j++) {
8              if (arr[j] > arr[j + 1]) {
9                  // Swap the elements
10                 int temp = arr[j];
11                 arr[j] = arr[j + 1];
12                 arr[j + 1] = temp;
13             }
14         }
15     }
16 }
17
18 #endif // BUBBLESORT_H

```

• Output Testing

<p>Output Console Showing Sorted Array</p>	<p>First Try:</p> <pre> Now, this is the sorted count of votes: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 4 5 </pre> <p>Second Try:</p> <pre> Now, this is the sorted count of votes: 1 2 3 4 4 4 4 4 4 4 4 5 </pre>
<p>Manual Count</p>	<p>First Try:</p> <pre> The number of votes for Candidates 1 to 5: Candidate 1: 14 votes Candidate 2: 18 votes Candidate 3: 23 votes Candidate 4: 23 votes Candidate 5: 22 votes </pre>

	<p>Second Try:</p> <pre>The number of votes for Candidates 1 to 5: Candidate 1: 20 votes Candidate 2: 28 votes Candidate 3: 21 votes Candidate 4: 18 votes Candidate 5: 13 votes</pre>
Count Result of Algorithm	<p>First Try:</p> <pre>Congratulations to the winner! Candidate 3 gaining with 23 votes! Excellent!</pre> <p>Second Try:</p> <pre>Congratulations to the winner! Candidate 2 gaining with 28 votes! Good Job!</pre>

Question: Was your developed vote counting algorithm effective? Why or why not?

The developed vote counting algorithm is effective for counting votes and finding the winner in small datasets. I used bubble sort, which works because the array is small and has only a few possible values of candidates 1 to 5. However, bubble sort is not very efficient for larger datasets, as it can slow down its run time if we have more datasets.

8. Conclusion

I learned a lesson about sorting algorithms, like bubble sort. But I also learned a few sorting techniques such as selection sort, and insertion sort, to help organize data. From the procedures, the output must be the same but different style of sorting techniques. Now in supplementary, it is generated random votes and counted them to find the winner. The process showed me that while bubble sort works well for small datasets, selection sort and insertion sort can also be effective, but they are still slow for larger ones. Overall, I did well in this activity, but I need to learn more about faster sorting methods for better performance.

9. Assessment Rubric

