

Activity No. 7			
Sorting Algorithms			
Course Code: CPE010	Program: Computer Engineering		
Course Title: Data Structures and Algorithms	Date Performed: 10/16/2024		
Section: CPE21S4	Date Submitted:10/18/2024		
Name(s): Danica T. Guariño	Instructor: Ma'am Rizette Sayo		
6. Output			
<table><tr><td>Code + Console Screenshot</td><td><pre>#include <iostream> #include <cstdlib> #include <ctime> using namespace std; int main() { const int size = 100; int arr[size]; std::srand(std::time(0)); // Seed for random number generation for (int i = 0; i < size; ++i) { arr[i] = std::rand() % 1000 + 1; // Random values between 1 and 1000 } // Display the unsorted array cout << "Unsorted Array:\n"; for (int i = 0; i < size; ++i) { cout << arr[i] << " "; } cout << std::endl; return 0; }</pre></td></tr></table>		Code + Console Screenshot	<pre>#include <iostream> #include <cstdlib> #include <ctime> using namespace std; int main() { const int size = 100; int arr[size]; std::srand(std::time(0)); // Seed for random number generation for (int i = 0; i < size; ++i) { arr[i] = std::rand() % 1000 + 1; // Random values between 1 and 1000 } // Display the unsorted array cout << "Unsorted Array:\n"; for (int i = 0; i < size; ++i) { cout << arr[i] << " "; } cout << std::endl; return 0; }</pre>
Code + Console Screenshot	<pre>#include <iostream> #include <cstdlib> #include <ctime> using namespace std; int main() { const int size = 100; int arr[size]; std::srand(std::time(0)); // Seed for random number generation for (int i = 0; i < size; ++i) { arr[i] = std::rand() % 1000 + 1; // Random values between 1 and 1000 } // Display the unsorted array cout << "Unsorted Array:\n"; for (int i = 0; i < size; ++i) { cout << arr[i] << " "; } cout << std::endl; return 0; }</pre>		


	 <pre> main.cpp 1 #include <iostream> 2 #include <cstdlib> 3 #include <ctime> 4 using namespace std; 5 6 int main() { 7 const int size = 100; 8 int arr[size]; 9 10 std::srand(std::time(0)); 11 for (int i = 0; i < size; ++i) { 12 arr[i] = std::rand() % 1000 + 1; 13 } 14 15 cout << "Unsorted Array:\n"; 16 for (int i = 0; i < size; ++i) { 17 cout << arr[i] << " "; 18 } 19 cout << std::endl; 20 21 return 0; 22 } 23 24 </pre> <p>Unsorted Array:</p> <p>261 852 279 514 924 182 876 109 72 803 653 54 566 618 941 296 362 354 36 36 972 438 589 363 182 134 137 53 498 308 839 110 159 469 975 434 2 203 542 73 5 547 478 570 516 418 217 878 123 252 913 94 41 853 457 222 986 945 627 836 252 817 945 410 637 920 196 638 122 737 62 478 283 539 47 799 308 615 28 431 218 940 524 258 792 332 832 130 276 810 965 879 626 261 641 262 180 836 251 653</p>
Observations	In this program, the array elements contains 100 random values that are not yet sorted.

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

Code + Console Screenshot	<pre> bubble.h #ifndef BUBBLESORT_H #define BUBBLESORT_H #include <algorithm> template <typename T> void bubble(T arr[], size_t arrSize) { for (size_t i = 0; i < arrSize - 1; i++) { for (size_t j = 0; j < arrSize - i - 1; j++) { if (arr[j] > arr[j + 1]) { std::swap(arr[j], arr[j + 1]); } } } } #endif // BUBBLESORT_H </pre>
---------------------------	---

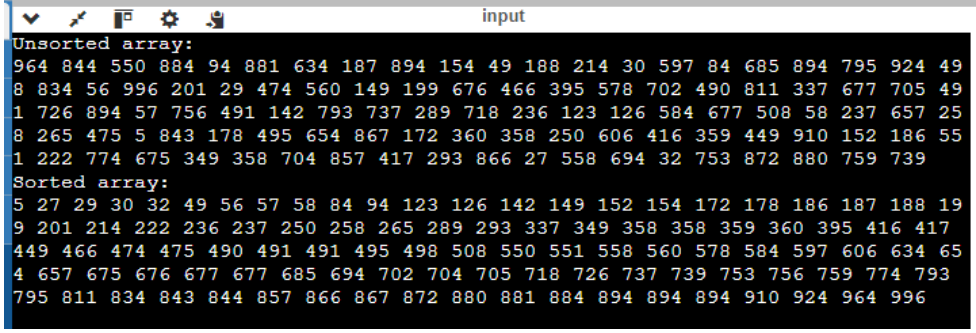
	 <pre> input Unsorted array: 964 844 550 884 94 881 634 187 894 154 49 188 214 30 597 84 685 894 795 924 49 8 834 56 996 201 29 474 560 149 199 676 466 395 578 702 490 811 337 677 705 49 1 726 894 57 756 491 142 793 737 289 718 236 123 126 584 677 508 58 237 657 25 8 265 475 5 843 178 495 654 867 172 360 358 250 606 416 359 449 910 152 186 55 1 222 774 675 349 358 704 857 417 293 866 27 558 694 32 753 872 880 759 739 Sorted array: 5 27 29 30 32 49 56 57 58 84 94 123 126 142 149 152 154 172 178 186 187 188 19 9 201 214 222 236 237 250 258 265 289 293 337 349 358 358 359 360 395 416 417 449 466 474 475 490 491 491 495 498 508 550 551 558 560 578 584 597 606 634 65 4 657 675 676 677 677 685 694 702 704 705 718 726 737 739 753 756 759 774 793 795 811 834 843 844 857 866 867 872 880 881 884 894 894 894 910 924 964 996 </pre>
Observations	<p>bubble.h function is used to sort an array, it's function is to compare the elements that has similar side or point. It will swap them if it is not in the right order.</p>

Table 7-2. Bubble Sort Technique

Code + Console Screenshot	<pre> selection.h #ifndef SELECTIONSORT_H #define SELECTIONSORT_H // Find the position of the smallest element in the unsorted portion template <typename T> int Routine_Smallest(T arr[], int start, int size) { int pos = start; for (int j = start + 1; j < size; j++) { if (arr[j] < arr[pos]) { pos = j; } } return pos; } // Selection sort implementation template <typename T> void selectionSort(T arr[], const int size) { for (int i = 0; i < size - 1; i++) { int pos = Routine_Smallest(arr, i, size); if (pos != i) { std::swap(arr[i], arr[pos]); // Swap the smallest found with the current position } } } #endif // SELECTIONSORT_H </pre>
---------------------------	---

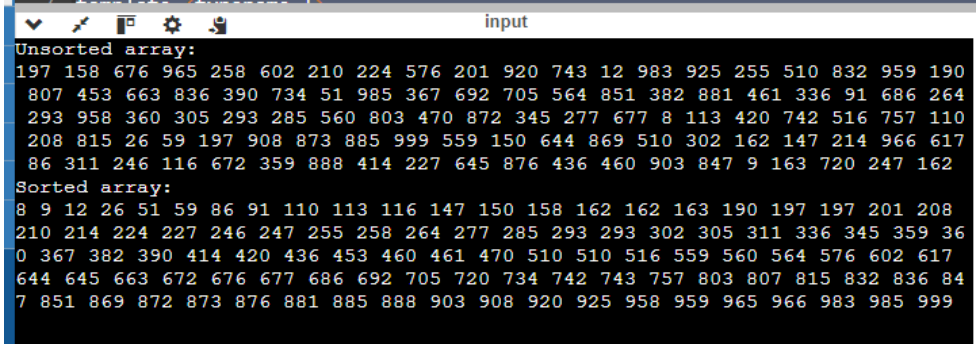
	
Observations	This function sorts the array of random numbers. From smallest element to the first unsorted element.

Table 7-3. Selection Sort Algorithm

Code + Console Screenshot	<pre> insertion.h #ifndef INSERTIONSORT_H #define INSERTIONSORT_H // Insertion sort function template 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 greater than temp to one position ahead while (J >= 0 && temp < arr[J]) { arr[J + 1] = arr[J]; J--; } // Place temp in the correct position arr[J + 1] = temp; } } #endif // INSERTIONSORT_H </pre>
---------------------------	---

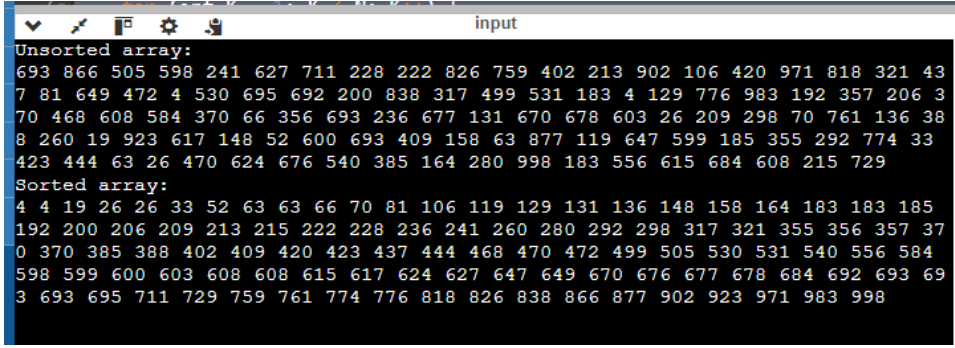
	 <pre> input Unsorted array: 693 866 505 598 241 627 711 228 222 826 759 402 213 902 106 420 971 818 321 43 7 81 649 472 4 530 695 692 200 838 317 499 531 183 4 129 776 983 192 357 206 3 70 468 608 584 370 66 356 693 236 677 131 670 678 603 26 209 298 70 761 136 38 8 260 19 923 617 148 52 600 693 409 158 63 877 119 647 599 185 355 292 774 33 423 444 63 26 470 624 676 540 385 164 280 998 183 556 615 684 608 215 729 Sorted array: 4 4 19 26 26 33 52 63 63 66 70 81 106 119 129 131 136 148 158 164 183 183 185 192 200 206 209 213 215 222 228 236 241 260 280 292 298 317 321 355 356 357 37 0 370 385 388 402 409 420 423 437 444 468 470 472 499 505 530 531 540 556 584 598 599 600 603 608 608 615 617 624 627 647 649 670 676 677 678 684 692 693 69 3 693 695 711 729 759 761 774 776 818 826 838 866 877 902 923 971 983 998 </pre>
Observations	Using insertion.h, an element from the unsorted array was selected and placed into its correct position within the sorted array.

Table 7-4. Insertion Sort Algorithm

7. Supplementary Activity

ILO B: Solve given data sorting problems using appropriate basic sorting algorithms

Candidate 1	Bo Dalton Capistrano
Candidate 2	Cornelius Raymon Agustin
Candidate 3	Deja Jayla Bañaga
Candidate 4	Lalla Brielle Yabut
Candidate 5	Franklin Relano Castro

List of Candidates

Problem: Generate an array $A[0...100]$ of unsorted elements, wherein the values in the array are indicative of a vote to a candidate. This means that the values in your array must only range from 1 to 5. Using sorting and searching techniques, develop an algorithm that will count the votes and indicate the winning candidate.

NOTE: The sorting techniques you have the option of using in this activity can be either bubble, selection, or insertion sort.

Justify why you chose to use this sorting algorithm.

I used bubble sort since it's simple to implement and understand, which makes it a good choice for small datasets. It is much easier to use and fast to understand the output of the program created.

• Pseudocode of Algorithm

Initialize SIZE to 100

Function generateVotes(arr, size):

Seed random number generator with current time

```
For i from 0 to size-1:  
    arr[i] = random number between 1 and 5
```

```
Function bubbleSort(arr, size):  
    For i from 0 to size-2:  
        For j from 0 to size-i-2:  
            If arr[j] > arr[j+1]:  
                Swap arr[j] and arr[j+1]
```

```
Function countVotes(arr, size, counts, numCandidates):  
    For i from 0 to size-1:  
        counts[arr[i] - 1]++
```

```
Function findWinner(counts, numCandidates):  
    maxVotes = counts[0]  
    winner = 1  
    For i from 1 to numCandidates-1:  
        If counts[i] > maxVotes:  
            maxVotes = counts[i]  
            winner = i + 1  
    Return winner
```

```
Main function:  
    Declare votes array of size SIZE  
    Call generateVotes(votes, SIZE)  
    Print "Unsorted Votes: " followed by votes array  
    Call bubbleSort(votes, SIZE)  
    Print "Sorted Votes: " followed by votes array  
    Declare voteCounts array of size 5 and initialize to 0  
    Call countVotes(votes, SIZE, voteCounts, 5)  
    Print "Vote Counts: " followed by counts for each candidate  
    Call findWinner(voteCounts, 5)  
    Print the winning candidate
```

• Screenshot of Algorithm Code

```
C/C++  
  
#include <iostream>  
#include <cstdlib>  
#include <ctime>  
  
const int SIZE = 100;  
  
void generateVotes(int arr[], int size) {  
    srand(time(0));  
    for (int i = 0; i < size; ++i) {  
        arr[i] = rand() % 5 + 1;  
    }  
}
```

```

    }
}

void bubbleSort(int arr[], int size) {
    for (int i = 0; i < size - 1; ++i) {
        for (int j = 0; j < size - i - 1; ++j) {
            if (arr[j] > arr[j + 1]) {
                std::swap(arr[j], arr[j + 1]);
            }
        }
    }
}

void countVotes(int arr[], int size, int counts[], int numCandidates) {
    for (int i = 0; i < size; ++i) {
        counts[arr[i] - 1]++;
    }
}

int findWinner(int counts[], int numCandidates) {
    int maxVotes = counts[0];
    int winner = 1;
    for (int i = 1; i < numCandidates; ++i) {
        if (counts[i] > maxVotes) {
            maxVotes = counts[i];
            winner = i + 1;
        }
    }
    return winner;
}

int main() {
    int votes[SIZE];
    generateVotes(votes, SIZE);

    std::cout << "Unsorted Votes: ";
    for (int i = 0; i < SIZE; ++i) {
        std::cout << votes[i] << " ";
    }
    std::cout << std::endl;

    bubbleSort(votes, SIZE);

    std::cout << "Sorted Votes: ";
    for (int i = 0; i < SIZE; ++i) {
        std::cout << votes[i] << " ";
    }
    std::cout << std::endl;

    int voteCounts[5] = {0};
    countVotes(votes, SIZE, voteCounts, 5);

    std::cout << "Vote Counts: ";
    for (int i = 0; i < 5; ++i) {
        std::cout << "Candidate " << i + 1 << ": " << voteCounts[i] << " votes" <<
std::endl;
    }
}

```

- **Output Testing**

Output Console Showing Sorted Array	Manual Count	Count Result of Algorithm
<pre>Unsorted Votes: 3 5 2 3 1 2 2 3 4 2 4 2 5 1 2 2 4 2 1 5 5 3 4 5 4 5 2 5 3 2 3 4 3 5 5 2 3 2 5 1 2 4 5 3 5 3 3 1 3 4 5 5 2 1 4 5 2 2 4 5 4 3 3 3 4 4 4 1 1 4 3 2 2 2 1 1 2 5 3 1 1 4 5 2 1 1 3 3 2 3 4 2 1 3 5 4 2 3 2 Sorted Votes: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 4 4 4 4 4 4 4 4 5</pre>	<p>Vote Counts</p> <p>Candidate 1: 15 votes</p> <p>Candidate 2: 25 votes</p> <p>Candidate 3: 22 votes</p> <p>Candidate 4: 18 votes</p> <p>Candidate 5: 20 votes</p>	<p>The winning candidate is Candidate 2</p>

Yes, it is effective since it is much easier to understand the program since I used bubble sort and it was sorted properly.

In conclusion, using sorting techniques in C++ provide a variety of approaches for effectively managing various data sets. While simpler algorithms, such as Bubble Sort, are simple to develop and understand, they may be inefficient for large datasets due to current complexity. Choosing the appropriate sorting algorithm is determined by the application's specific needs, such as dataset size, complexity, and necessary stability. To summarize, knowing a variety of sorting algorithms enables optimized and efficient data processing, making it an essential skill in C++ programming. Having to know how to sort out any unsorted elements is such a great tactic that I can use for my future codings.

9. Assessment Rubric