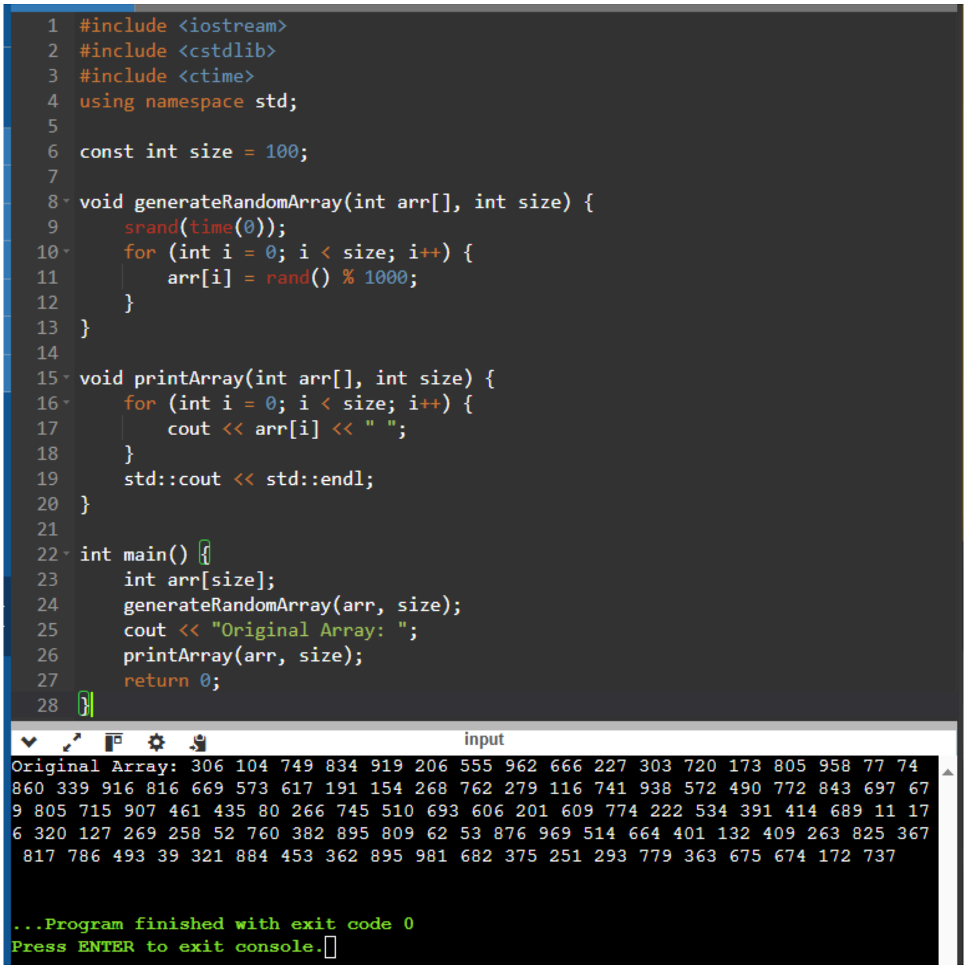
Activity No. 7

Hands-on Activity 7.1 Sorting Algorithms

Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** October 16, 2024
Section: CPE21S4	**Date Submitted:** October 17, 2024
Name(s): Don Eleazar T. Fernandez	**Instructor:** Maria Rizette Sayo

6. Output

Code + Console Screenshot



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

const int size = 100;

void generateRandomArray(int arr[], int size) {
    srand(time(0));
    for (int i = 0; i < size; i++) {
        arr[i] = rand() % 1000;
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    int arr[size];
    generateRandomArray(arr, size);
    cout << "Original Array: ";
    printArray(arr, size);
    return 0;
}
```

```
Original Array: 306 104 749 834 919 206 555 962 666 227 303 720 173 805 958 77 74
860 339 916 816 669 573 617 191 154 268 762 279 116 741 938 572 490 772 843 697 67
9 805 715 907 461 435 80 266 745 510 693 606 201 609 774 222 534 391 414 689 11 17
6 320 127 269 258 52 760 382 895 809 62 53 876 969 514 664 401 132 409 263 825 367
 817 786 493 39 321 884 453 362 895 981 682 375 251 293 779 363 675 674 172 737


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

Observations

The program generates an array with 100 random digits, each ranging from 0 to 999. It first initializes a random number based on the current time to ensure different values each time the program runs. And so, it prints the entire array to the console.

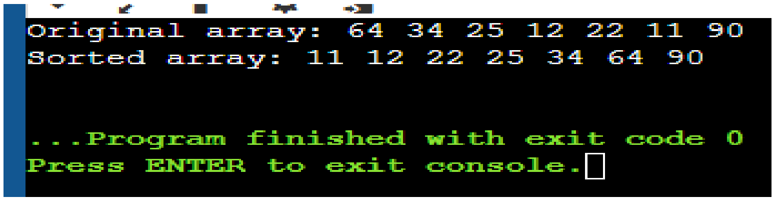
Table 7 - 1. Array of Values for Sort Algorithm Testing

Code + Console Screenshot

```
#include <iostream>
#include <algorithm>
using namespace std;

void printArray(int arr[], size_t arrSize) {
    for (int i = 0; i < arrSize; i++) {
        cout << arr[i] << " ";
```

<table>
<tr>
<td></td>
<td>

```
    }
    cout << endl;
}

template <typename T>
void bubbleSort(T arr[], size_t arrSize) {
    for (int i = 0; i < arrSize - 1; i++) {
        bool swapped = false;
        for (int j = i + 1; j < arrSize; j++) {
            if (arr[i] > arr[j]) {
                swap(arr[i], arr[j]);
                swapped = true;
            }
        }
        if (!swapped) {
            break;
        }
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    size_t arrSize = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    printArray(arr, arrSize);
    bubbleSort(arr, arrSize);
    cout << "Sorted array: ";
    printArray(arr, arrSize);
    return 0;
}
```



```
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90


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

</td>
</tr>
<tr>
<td>Observations</td>
<td>The program utilizes a bubble sort algorithm to sort the array. It starts by printing the original array, then sorts it in ascending order using nested loops that repeatedly swap adjacent elements if they are out of order. After, it prints the sorted array to the console.</td>
</tr>
</table>

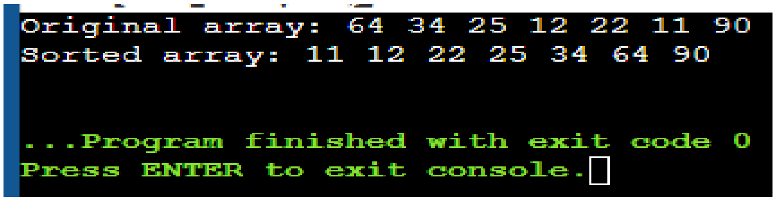
Table 7 - 2. Bubble Sort Technique

<table>
<tr>
<td>Code + Console Screenshot</td>
<td>

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

void printArray(int arr[], size_t arrSize) {
    for (int i = 0; i < arrSize; i++) {
        cout << arr[i] << " ";
    }
```

</td>
</tr>
</table>

```
    cout << endl;
}

template <typename T>
int Routine_Smallest(T A[], int K, const int arrSize) {
    int position, j;
    T smallestElem = A[K];
    position = K;
    for (int J = K + 1; J < arrSize; J++) {
        if (A[J] < smallestElem) {
            smallestElem = A[J];
            position = J;
        }
    }
    return position;
}

template <typename T>
void selectionSort(T arr[], const int N) {
    int POS, temp, pass = 0;
    for (int i = 0; i < N; i++) {
        POS = Routine_Smallest(arr, i, N);
        temp = arr[i];
        arr[i] = arr[POS];
        arr[POS] = temp;
        pass++;
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    size_t arrSize = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    printArray(arr, arrSize);
    selectionSort(arr, arrSize);
    cout << "Sorted array: ";
    printArray(arr, arrSize);
    return 0;
}
```



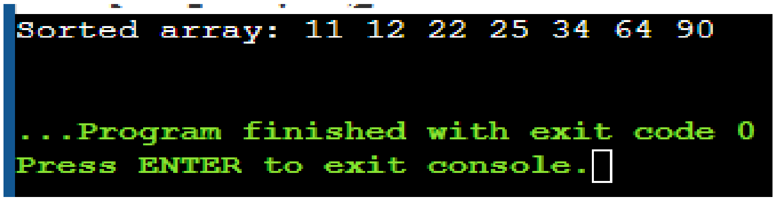
```
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90


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

Observations	The program utilized the selection sort algorithm to sort the array. First, it prints the original unsorted array, then it sorts the array in ascending order. Lastly, it prints the sorted array to the console.

Table 7 - 3. Selection Sort Algorithm

Code + Console Screenshot

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

template <typename T>
void insertionSort(T arr[], const int N) {
    int K = 0, J, temp;
    while (K < N) {
        temp = arr[K];
        J = K - 1;
        while (temp <= arr[J] && J >= 0) {
            arr[J + 1] = arr[J];
            J--;
        }
        arr[J + 1] = temp;
        K++;
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int N = sizeof(arr) / sizeof(arr[0]);
    insertionSort(arr, N);
    cout << "Sorted array: ";
    for (int i = 0; i < N; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```



```
Sorted array: 11 12 22 25 34 64 90


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

Observations

The program utilizes the Insertion Sort algorithm to sort an array in ascending order. After, the program prints the sorted array to the console.

Table 7 - 4. Insertion Sort Algorithm

7. Supplementary Activity

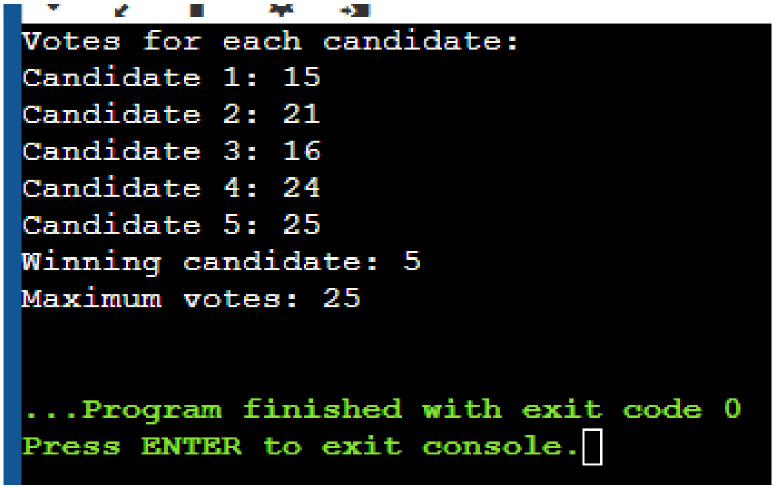
Pseudocode of Algorithm

```
Procedure VoteCounter()
  Initialize array A[0...100] with random votes (1-5)
  Sort array A using Insertion Sort
  Initialize variables to store the count of votes for each candidate
  Initialize variable to store the winning candidate
  Initialize variable to store the maximum votes
  For each element in the sorted array
    If the element is equal to 1
      Increment the count of votes for candidate 1
    Else if the element is equal to 2
      Increment the count of votes for candidate 2
    Else if the element is equal to 3
      Increment the count of votes for candidate 3
    Else if the element is equal to 4
      Increment the count of votes for candidate 4
    Else if the element is equal to 5
      Increment the count of votes for candidate 5

  Find the maximum votes and the corresponding winning candidate

  Print the count of votes for each candidate
  Print the winning candidate and the maximum votes

End Procedure
```

Screenshot of Algorithm Code

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

void insertionSort(int arr[], int n) {
  int i, key, j;
  for (i = 1; i < n; i++) {
    key = arr[i];
    j = i - 1;
    while (j >= 0 && arr[j] > key) {
      arr[j + 1] = arr[j];
      j = j - 1;
    }
    arr[j + 1] = key;
  }
}

void voteCounter(int arr[], int n) {
  int votes[5] = {0};
  int maxVotes = 0;
  int winningCandidate;
  for (int i = 0; i < n; i++) {
    if (arr[i] == 1) votes[0]++;
    else if (arr[i] == 2) votes[1]++;
    else if (arr[i] == 3) votes[2]++;
    else if (arr[i] == 4) votes[3]++;
    else if (arr[i] == 5) votes[4]++;
  }
  for (int i = 0; i < 5; i++) {
    if (votes[i] > maxVotes) {
      maxVotes = votes[i];
      winningCandidate = i + 1;
    }
  }
  cout << "Votes for each candidate:" << endl;
  for (int i = 0; i < 5; i++) {
    cout << "Candidate " << i + 1 << ": " << votes[i] << endl;
  }
  cout << "Winning candidate: " << winningCandidate << endl;
  cout << "Maximum votes: " << maxVotes << endl;
}

int main() {
  srand(time(0));
  int arr[101];
  for (int i = 0; i < 101; i++) {
    arr[i] = rand() % 5 + 1;
  }
  insertionSort(arr, 101);
  voteCounter(arr, 101);
  return 0;
}
```

Output



```
Votes for each candidate:
Candidate 1: 15
Candidate 2: 21
Candidate 3: 16
Candidate 4: 24
Candidate 5: 25
Winning candidate: 5
Maximum votes: 25


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

Output Console Showing Sorted Array	Manual Count	Count Result of Algorithm
Sorted Array: 1 2 2 3 3 3 4 4 4 4 5…	Candidate 1: 14 Candidate 2: 18 Candidate 3: 14 Candidate 4: 26 Candidate 5: 29	winner is candidate 5 with maximum votes: 29

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

- The program is effective, as it successfully counts the votes and identifies the winning candidate. Through iteration of the array of votes, it updates the vote count for each candidate, and then identifies the candidate with the maximum votes.

8. Conclusion

In conclusion, this laboratory activity provided clear instructions on implementing bubble, insertion, and selection sort. Bubble sort is simple but slow, selection sort is easy to understand but inefficient for large datasets, and insertion sort works best for small sorted arrays. The program done in the supplementary activity effectively applied all the mentioned sort. The areas of improvement that can be done are in the linked list, as I still am confused with the structure.

9. Assessment Rubric

I affirm that I will not give or receive any unauthorized help on this activity/exam and that all work will be my own.