Hands-on Activity 7.1	
Sorting Algorithms Pt1	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 09 - 18 - 25
Section: CPE21S4	**Date Submitted:** 09 - 18 - 25
Name(s): Santiago, David Owen A.	**Instructor:** Engr. Jimlord Quejado

6. Output

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

Code:

Void function for generating random values:

```
void randomNumberArray(int arr[], size_t arrSize) {
    for (int i = 0; i < maxSize; i++) {
        arr[i] = rand() % 100;
    }
}
```

Print function in sorting.h header:

```
template<typename T>
void printArray(T arr[], size_t arrSize) {
    for (int i = 0; i < arrSize; i++) {
        std::cout << arr[i] << " ";
    }std::cout << '\n';
}
```

Main driver function:

```
#include <iostream>
#include <cstdlib>
#include "sorting.h"

const int maxSize = 100;

void line() {
    std::cout <<
"------------------------------------------------------------\n";
}
void randomNumberArray(int arr[], size_t arrSize) {
    for (int i = 0; i < maxSize; i++) {
        arr[i] = rand() % 100;
    }
}
int main() {

    int data[maxSize];
```

```
    std::cout << "Array of numbers from 0-100: \n";
    randomNumberArray(data, maxSize);
    printArray(data, maxSize);

    //line();

    //bubbleSort(data, maxSize);
    //printArray(data, maxSize);

    return 0;
}
```

Output:

```
"D:\College\Year 2 CPE\Y2 Sem 1\CPE 010\Midterm\HOA 7.1\cmake-build-debug\HOA_7_1.exe"
Array of numbers from 0-100:
41 67 34 0 69 24 78 58 62 64 5 45 81 27 61 91 95 42 27 36 91 4 2 53 92 82 21 16 18 95 47 26 71 38 69 12 67 99 35 94 3 11
 22 33 73 64 41 11 53 68 47 44 62 57 37 59 23 41 29 78 16 35 90 42 88 6 40 42 64 48 46 5 90 29 70 50 6 1 93 48 29 23 84
54 56 40 66 76 31 8 44 39 26 23 37 38 18 82 29 41

Process finished with exit code 0
```

Because I didn't use a specific time seed, I was able to create one set of random numbers from 0-100. This line arr[i] = rand() % 100; ensures that the numbers are strictly between 0 -100. As expected, since I used the rand() function, these numbers are unsorted and randomized when entered in an array.

Table 7-2. Bubble Sort Technique

Code:

Header File:
Swap function that takes the reference address of the two variables to be swapped:

```
template<typename T>
void swap(T &a, T &b) {
    T temp = a;
    a = b;
    b = temp;
}
```

Implementation of bubble sort:

```
template<typename T>
void bubbleSort(T arr[], size_t arrSize) {
    std::cout << "Bubble Sort: \n";
```

```
    // To keep track of the number of swaps
    int swaps = 0;
    // To keep track of comparisons
    int comparisons = 0;
    // Step 1: For i = 0 to N - 1 repeat step 2
    // Traversal Loop
    for (int i = 0; i < arrSize - 1; i++) {
        // Step 2: For j = i + 1 to N - i repeat
        // Comparison loop
        for (int j = i + 1; j < arrSize; j++) {
            // Incrementing comparisons to keep track
            comparisons++;
            // Reverse arrow for descending order
            if (arr[i] > arr[j]) {
                swap(arr[j], arr[i]);
                // Increments only if swapped
                swaps++;
            }
        }
    }
    std::cout << "Swaps: " << swaps << '\n';
    std::cout << "Comparisons: " << comparisons << '\n';
}
```

Main:

```
#include <iostream>
#include <cstdlib>
#include "sorting.h"

const int maxSize = 100;

void line() {
    std::cout <<
"-------------------------------------------------------\n";
}
void randomNumberArray(int arr[], size_t arrSize) {
    for (int i = 0; i < maxSize; i++) {
        arr[i] = rand() % 100;
    }
}

int main() {

    int data[maxSize];

    std::cout << "Array of numbers from 0-100: \n";
    randomNumberArray(data, maxSize);
    printArray(data, maxSize);

    line();
```

```
    bubbleSort(data, maxSize);
    printArray(data, maxSize);

    return 0;
}
```

Output:

```
"D:\College\Year 2 CPE\Y2 Sem 1\CPE 010\Midterm\HOA 7.1\cmake-build-debug\HOA_7_1.exe"
Array of numbers from 0-100:
Array: 41 67 34 0 69 24 78 58 62 64 5 45 81 27 61 91 95 42 27 36 91 4 2 53 92 82 21 16 18 95 47 26 71 38 69 12 67 99 35
94 3 11 22 33 73 64 41 11 53 68 47 44 62 57 37 59 23 41 29 78 16 35 90 42 88 6 40 42 64 48 46 5 90 29 70 50 6 1 93 48 29
 23 84 54 56 40 66 76 31 8 44 39 26 23 37 38 13 82 29 41
-----------------------------------------------------
Bubble Sort:
Swaps: 1955
Comparisons: 4950
Array: 0 1 2 3 4 5 5 6 6 8 11 11 12 16 16 18 18 21 22 23 23 23 24 26 26 27 27 29 29 29 29 31 33 34 35 35 36 37 37 38 38
39 40 40 41 41 41 41 42 42 42 44 44 45 46 47 47 48 48 50 53 53 54 56 57 58 59 61 62 62 64 64 64 66 67 67 68 69 69 70 71
73 76 78 78 81 82 82 84 88 90 90 91 91 92 93 94 95 95 99

Process finished with exit code 0
```

The bubble sorting algorithm has a noticeably higher amount of swapping due to the loop going back to the first element over and over again. The outer loop starts at the beginning, then the inner loop compares one element to all the elements, brings it to the front, ends, then the process begins again per iteration. This is inefficient and memory costly.

Table 7-3. Selection Sort Algorithm

Code:

Header File:

```
template<typename T>
int routineSmallest(T A[], int K, const int arrSize, int
&comparisons) {
    int position, j;
    // Step 1: [initialize] set smallestElem = A[K]
    T smallestElem = A[K];
    // Step 2:  [initialize] set POS = K
    position = K;
```

```
    // Step 3: for J = K + 1 to N - 1 repeat
    for (int j = K + 1; j < arrSize; j++) {
        comparisons++;
        if (A[j] < smallestElem) {
            smallestElem = A[j];
            position = j;
        }
    }
    // Step 4: Return pos
    return position;

}
void selectionSort(int arr[], const int N) {
    int swaps = 0;
    int comparisons = 0;
    int POS;
    std::cout << "Selection Sort: \n";

    // Step 1: Repeat Steps 2 and 3 for K = 1 to N-1
    for (int i = 0; i < N; i++) {
        // Step 2: Call routine smallest(A, K, N, POS,
added comparisons to track)
        POS = routineSmallest(arr, i, N, comparisons);
        // Step 3: Swap A[K] with A[POS]
        if (i != POS) {
            swap(arr[i], arr[POS]);
            swaps++;
        }
    }
    std::cout << "Swaps: " << swaps << '\n';
    std::cout << "Comparisons: " << comparisons << '\n';
}
```

Main File:

```
#include <iostream>
#include <cstdlib>
#include "sorting.h"

const int maxSize = 100;
```

```
void line() {
   std::cout <<
"-------------------------------------------------------
-\n";
}
void randomNumberArray(int arr[], size_t arrSize) {
   for (int i = 0; i < maxSize; i++) {
       arr[i] = rand() % 100;
   }
}
int main() {

   int data[maxSize];

   std::cout << "Array of numbers from 0-100: \n";
   randomNumberArray(data, maxSize);
   printArray(data, maxSize);

   line();

   selectionSort(data, maxSize);
   printArray(data, maxSize);

   return 0;
}
```

Output:

```
"D:\College\Year 2 CPE\Y2 Sem 1\CPE 010\Midterm\HOA 7.1\cmake-build-debug\HOA_7_1.exe"
Array of numbers from 0-100:
Array: 41 67 34 0 69 24 78 58 62 64 5 45 81 27 61 91 95 42 27 36 91 4 2 53 92 82 21 16 18 95 47 26 71 38 69 12 67 99 35
94 3 11 22 33 73 64 41 11 53 68 47 44 62 57 37 59 23 41 29 78 16 35 90 42 88 6 40 42 64 48 46 5 90 29 70 50 6 1 93 48 29
 23 84 54 56 40 66 76 31 8 44 39 26 23 37 38 18 82 29 41
-----------------------------------------------------
Selection Sort:
Swaps: 94
Comparisons: 4950
Array: 0 1 2 3 4 5 5 6 6 8 11 11 12 16 16 18 18 21 22 23 23 23 24 26 26 27 27 29 29 29 29 31 33 34 35 35 36 37 37 38 38
39 40 40 41 41 41 41 42 42 42 44 44 45 46 47 47 48 48 50 53 53 54 56 57 58 59 61 62 62 64 64 64 66 67 67 68 69 69 70 71
73 76 78 78 81 82 82 84 88 90 90 91 91 92 93 94 95 95 99

Process finished with exit code 0
```

The sorting algorithm has much less swapping done which means it is more memory efficient and faster than bubble sorting. Since I kept my swapping and comparisons counter, I was able to keep track of each swap and comparison which shows how its much more efficient. Still, its disadvantage is that it can't immediately detect a sorted array.

Table 7-4. Insertion Sort Algorithm

Code:

Header File:

```
template<typename T>
void insertionSort(T arr[], const int N) {
   std::cout << "Insertion Sort: \n";
   int K = 1, J;
   T temp;
   int comparisons = 0;
   int moves = 0;

   // Step 1: Repeat steps 2-5 for K = 1 to N - 1
   while (K < N) {
       // Step 2: set temp = A[K]
       temp = arr[K];
       // Step 3: set J = K - 1 to start comparison on the element before K
       J = K - 1;

       // Step 4: Repeat while temp <= A[J]
       while (J >= 0 && temp < arr[J] ) {
           comparisons++;
           // Set A[J + 1] = A[J]
           arr[J + 1] = arr[J];
           moves++;
           // Set J = J - 1
           J--;
       }
       if (J >= 0) comparisons++; // includes last comparison
       // Step 5: set A[J + 1] = temp
       arr[J + 1] = temp;
```

```
        moves++;
        K++;
    }
    std::cout << "Moves: " << moves << '\n';
    std::cout << "Comparisons: " << comparisons << '\n';
}
```

Main File:

```
#include <iostream>
#include <cstdlib>
#include "sorting.h"

const int maxSize = 100;

void line() {
    std::cout << "------------------------------------------------------------\n";
}
void randomNumberArray(int arr[], size_t arrSize) {
    for (int i = 0; i < maxSize; i++) {
        arr[i] = rand() % 100;
    }
}
int main() {

    int data[maxSize];

    std::cout << "Array of numbers from 0-100: \n";
    randomNumberArray(data, maxSize);
    printArray(data, maxSize);

    line();

    // bubbleSort(data, maxSize);
    // printArray(data, maxSize);

    //selectionSort(data, maxSize);
    //printArray(data, maxSize);

    insertionSort(data, maxSize);
    printArray(data, maxSize);

    return 0;
}
```

Output:

```
"D:\College\Year 2 CPE\Y2 Sem 1\CPE 010\Midterm\HOA 7.1\cmake-build-debug\HOA_7_1.exe"
Array of numbers from 0-100:
Array: 41 67 34 0 69 24 78 58 62 64 5 45 81 27 61 91 95 42 27 36 91 4 2 53 92 82 21 16 18 95 47 26 71 38 69 12 67 99 35
94 3 11 22 33 73 64 41 11 53 68 47 44 62 57 37 59 23 41 29 78 16 35 90 42 88 6 40 42 64 48 46 5 90 29 70 50 6 1 93 48 29
 23 84 54 56 40 66 76 31 8 44 39 26 23 37 38 18 82 29 41
--------------------------------------------------------
Insertion Sort:
Moves: 2708
Comparisons: 2706
Array: 0 1 2 3 4 5 5 6 6 8 11 11 12 16 16 18 18 21 22 23 23 23 24 26 26 27 27 29 29 29 29 31 33 34 35 35 36 37 37 38 38
39 40 40 41 41 41 41 42 42 42 44 44 45 46 47 47 48 48 50 53 53 54 56 57 58 59 61 62 62 64 64 64 66 67 67 68 69 69 70 71
73 76 78 78 81 82 82 84 88 90 90 91 91 92 93 94 95 95 99

Process finished with exit code 0
```

7. Supplementary Activity

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

Candidate 1	Bo Dalton Capistrano
Candidate 2	Cornelius Raymon Agustín
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.

Deliverables:

- Pseudocode of Algorithm

- Screenshot of Algorithm Code
- Output Testing

Pseudocode

```
DEFINE constant maxSize = 100

FUNCTION randomNumberArray(array, size)
   FOR i FROM 0 TO size - 1
      array[i] = RANDOM INTEGER BETWEEN 1 AND 5
   END FOR
END FUNCTION

FUNCTION line()
   PRINT horizontal divider
END FUNCTION

MAIN PROGRAM

   DECLARE array data[maxSize]

   PRINT "Candidates:"
   INITIALIZE array c[5] WITH:
      Candidate("Bo Dalton Capistrano", 0)
      Candidate("Cornelius Raymon Agustin", 0)
      Candidate("Deja Jayla Banaga", 0)
      Candidate("Lalla Brielle Yabut", 0)
      Candidate("Franklin Relano Castro", 0)

   CALL line()

   PRINT "Votes:"
   CALL randomNumberArray(data, maxSize)
   CALL printArray(data, maxSize)
   CALL line()

   PRINT "Sorting Votes:"
   CALL selectionSort(data, maxSize)
   CALL printArray(data, maxSize)
   CALL line()

   PRINT "Counting Votes..."
   CALL countVotes(c, data, 5, maxSize)
   PRINT "Successfully counted..."

   FOR each candidate IN c
      DISPLAY candidate name and vote count
   END FOR
```

```
    CALL line()

    CALL sortCandidatesByVotes(c, 5)
    CALL line()

    PRINT "Candidates from most votes to least:"
    FOR each candidate IN c
        DISPLAY candidate name and vote count
    END FOR

    CALL line()

    PRINT "The winner is:"
    DISPLAY c[0] (first candidate in sorted list)
    CALL line()

END MAIN
```

Headers:

I created a new candidate header file to create candidates:

```
#ifndef CANDIDATES_H
#define CANDIDATES_H
#include <iostream>
#include "searching.h"
#include "sorting.h"

class Candidates {
public:
    std::string name;
    int votes;

    // Default Constructor
    Candidates() {
        name = "";
        votes = 0;
        std::cout << "Candidate created\n";
    }
    // Constructor with parameter
    Candidates(const std::string &n, const int v) {
        name = n;
        votes = v;
        std::cout << "Candidate " << name << " created\n";
    }
```

```
    //Copy Constructor
    Candidates(const Candidates& orig) {
        name = orig.name;
        votes = orig.votes;
        std::cout << "Copy candidate " << name << " created\n";
    }
    // Destructor
    ~Candidates() {
        std::cout << "Candidate " << name << " deleted\n";
    }


};
inline void sortCandidatesByVotes(Candidates arr[], int size) {
    for (int i = 0; i < size - 1; ++i) {
        int maxIndex = i;
        for (int j = i + 1; j < size; ++j) {
            if (arr[j].votes > arr[maxIndex].votes) {
                maxIndex = j;
            }
        }
        if (maxIndex != i) {
            Candidates temp = arr[i];
            arr[i] = arr[maxIndex];
            arr[maxIndex] = temp;
        }
    }
}
inline void countVotes(Candidates candidates[], int votes[], int numCandidates,
int voteArraySize) {
    for (int i = 0; i < numCandidates; ++i) {
        candidates[i].votes = countOccurrencesArray(votes, voteArraySize, i + 1);
    }
}
inline void cDisplay(const Candidates &candidate)  {
    std::cout << "Candidate " << candidate.name << " has " << candidate.votes << "
votes.\n";
}

#endif //CANDIDATES_H
```

Searching.h for searching and counting votes

```
//
// Created by David Owen A. Santiago on 16/09/2025.
//
#ifndef SEARCHING_H
#define SEARCHING_H

#include <iostream>
#include "nodes.h"

// Linear search function template
template <typename T>
bool arrayLinearSearch(const T data[], int size, T item) {
    for (int i = 0; i < size; ++i) {
```

```
        if (data[i] == item) {
            std::cout << "Searching is successful.\nFound at index: " << i << "\n";
            return true;
        }
    }
    std::cout << "Searching is not successful.\n";
    return false;
}


// Added for linked list
template <typename T>
bool linkedLinearSearch(Node<T> *head, T item ) {
    int index = 0;
    Node<T> *temp = head;
    while (temp != nullptr) {
        if (temp->data == item) {
            std::cout << "Searching is successful.\nFound at node: " << index <<
"\n";
            return true;
        }
        temp = temp->next;
        ++index;
    }

    std::cout << "Searching is not successful.\n";
    return false;
}

template <typename T>
bool arrayBinarySearch(const T arr[], int size, T item) {
    int low = 0;
    int up = size - 1;

    while (low <= up) {
        int mid = (low + up) / 2;

        if (arr[mid] == item) {
            std::cout << "Search element is found at index: " << mid << "\n";
            return true;
        } else if (item < arr[mid]) {
            up = mid - 1;
        } else {
            low = mid + 1;
        }
    }

    std::cout << "Search element is not found.\n";
    return false;
}

template <typename T>
Node<T>* getMiddle(Node<T>* start, Node<T>* end) {
    if (start == nullptr) return nullptr;
    Node<T>* slow = start;
    Node<T>* fast = start;
    while (fast != end && fast->next != end) {
```

```
        fast = fast->next->next;
        slow = slow->next;
    }
    return slow;
}

template <typename T>
Node<T>* linkedBinarySearch(Node<T>* head, T key) {
    Node<T>* start = head;
    Node<T>* end = nullptr;

    while (start != end) {
        Node<T>* mid = getMiddle(start, end);

        if (mid == nullptr) return nullptr;

        if (mid->data == key) {
            std::cout << "Search successful. Found: " << mid->data << "\n";
            return mid;
        } else if (key < mid->data) {
            end = mid;
        } else {
            start = mid->next;
        }
    }

    std::cout << "Search unsuccessful. Element not found.\n";
    return nullptr;
}


template <typename T>
int arrayLinearSearchComparisons(const T data[], int size, T item) {
    int comparisons = 0;
    for (int i = 0; i < size; ++i) {
        comparisons++;
        if (data[i] == item) break;
    }
    return comparisons;
}

template <typename T>
int countOccurrencesArray(const T data[], int size, T item) {
    int count = 0;
    for (int i = 0; i < size; ++i)
        if (data[i] == item) count++;
    return count;
}

template <typename T>
int linkedLinearSearchComparisons(Node<T>* head, T item) {
    int comparisons = 0;
    Node<T>* temp = head;
    while (temp != nullptr) {
        comparisons++;
        if (temp->data == item) break;
        temp = temp->next;
```

```
    }
    return comparisons;
}

template <typename T>
int countOccurrencesLinkedList(Node<T>* head, T item) {
    int count = 0;
    Node<T>* temp = head;
    while (temp) {
        if (temp->data == item) count++;
        temp = temp->next;
    }
    return count;
}

template <typename T>
int recursiveBinarySearch(const T arr[], T low, T up, T key) {
    if (low > up) return -1;

    int mid = (low + up) / 2;

    if (arr[mid] == key)
        return mid;
    else if (key < arr[mid])
        return recursiveBinarySearch(arr, low, mid - 1, key);

    else
        return recursiveBinarySearch(arr, mid + 1, up, key);
}




#endif // SEARCHING_H
```

sorting.h for sorting the votes

```
//
// Created by oms2v on 18/09/2025.
//

#ifndef SORTING_H
#define SORTING_H
#include <iostream>
#include <cstdlib>

//swapping
template<typename T>
void swap(T &a, T &b) {
    T temp = a;
    a = b;
    b = temp;
}

//printing
```

```cpp
template<typename T>
void printArray(T arr[], size_t arrSize) {
   std::cout << "Array: ";
   for (int i = 0; i < arrSize; i++) {
       std::cout << arr[i] << " ";
   }std::cout << '\n';
}

//bubble
template<typename T>
void bubbleSort(T arr[], size_t arrSize) {
   std::cout << "Bubble Sort: \n";
   // To keep track of the number of swaps
   int swaps = 0;
   // To keep track of comparisons
   int comparisons = 0;
   // Step 1: For i = 0 to N - 1 repeat step 2
   // Traversal Loop
   for (int i = 0; i < arrSize - 1; i++) {
       // Step 2: For j = i + 1 to N - i repeat
       // Comparison loop
       for (int j = i + 1; j < arrSize; j++) {
           // Incrementing comparisons to keep track
           comparisons++;
           // Reverse arrow for descending order
           if (arr[i] > arr[j]) {
               swap(arr[j], arr[i]);
               // Increments only if swapped
               swaps++;
           }
       }
   }
   std::cout << "Swaps: " << swaps << '\n';
   std::cout << "Comparisons: " << comparisons << '\n';
}

//selection
template<typename T>
int routineSmallest(T A[], int K, const int arrSize, int &comparisons) {
   int position, j;
   // Step 1: [initialize] set smallestElem = A[K]
   T smallestElem = A[K];
   // Step 2:  [initialize] set POS = K
   position = K;
   // Step 3: for J = K + 1 to N - 1 repeat
   for (int j = K + 1; j < arrSize; j++) {
       comparisons++;
       if (A[j] < smallestElem) {
           smallestElem = A[j];
           position = j;
       }
   }
   // Step 4: Return pos
   return position;

}
template<typename T>
```

```cpp
void selectionSort(T arr[], const int N) {
   int swaps = 0;
   int comparisons = 0;
   int POS;
   std::cout << "Selection Sort: \n";

   // Step 1: Repeat Steps 2 and 3 for K = 1 to N-1
   for (int i = 0; i < N; i++) {
       // Step 2: Call routine smallest(A, K, N, POS, added comparisons to track)
       POS = routineSmallest(arr, i, N, comparisons);
       // Step 3: Swap A[K] with A[POS]
       if (i != POS) {
           swap(arr[i], arr[POS]);
           swaps++;
       }
   }
   std::cout << "Swaps: " << swaps << '\n';
   std::cout << "Comparisons: " << comparisons << '\n';
}

//insertion
template<typename T>
void insertionSort(T arr[], const int N) {
   std::cout << "Insertion Sort: \n";
   int K = 1, J;
   T temp;
   int comparisons = 0;
   int moves = 0;

   // Step 1: Repeat steps 2-5 for K = 1 to N - 1
   while (K < N) {
       // Step 2: set temp = A[K]
       temp = arr[K];
       // Step 3: set J = K - 1 to start comparison on the element before K
       J = K - 1;

       // Step 4: Repeat while temp <= A[J]
       while (J >= 0 && temp < arr[J] ) {
           comparisons++;
           // Set A[J + 1] = A[J]
           arr[J + 1] = arr[J];
           moves++;
           // Set J = J - 1
           J--;
       }
       if (J >= 0) comparisons++; // includes last comparison
       // Step 5: set A[J + 1] = temp
       arr[J + 1] = temp;
       moves++;
       K++;
   }
   std::cout << "Moves: " << moves << '\n';
   std::cout << "Comparisons: " << comparisons << '\n';
}

#endif //SORTING_H
```

Main file

```cpp
//
// Created by oms2v on 18/09/2025.
//
#include <iostream>
#include <cstdlib>
#include "sorting.h"
#include "searching.h"
#include "candidates.h"


const int maxSize = 100;

void randomNumberArray(int arr[], size_t arrSize) {
    for (int i = 0; i < arrSize; i++) {
        arr[i] = (rand() % 5) + 1;
    }
}

void line() {
    std::cout << "-----------------------------------------\n";
}
int main() {

    int data[maxSize];

    std::cout << "Candidates: \n";

    Candidates c[] = {
        Candidates("Bo Dalton Capistrano", 0),
        Candidates("Cornelius Raymon Agustin", 0),
        Candidates("Deja Jayla Banaga", 0),
        Candidates("Lalla Brielle Yabut",0),
        Candidates("Franklin Relano Castro", 0),
    };

    line();

    std::cout << "Votes: \n";
    randomNumberArray(data, maxSize);
    printArray(data, maxSize);
    line();

    std::cout << "Sorting Votes: \n";
    selectionSort(data, maxSize);
    printArray(data, maxSize);
    line();

    std::cout << "Counting Votes...\n";
    countVotes(c, data, 5, maxSize);
    std::cout << "Successfully counted...\n";

    for (int i = 0; i < 5; i++) {
        cDisplay(c[i]);
    }
```

```
    line();
    sortCandidatesByVotes(c, 5);
    line();

    std::cout << "Candidates from most votes to least: \n";
    for (int i = 0; i < 5; i++) {
        cDisplay(c[i]);
    }

    line();
    std::cout << "The winner is: \n";
    cDisplay(c[0]);
    line();

    return 0;
}
```

Output:

```
Candidates:
Candidate Bo Dalton Capistrano created
Candidate Cornelius Raymon Agustin created
Candidate Deja Jayla Banaga created
Candidate Lalla Brielle Yabut created
Candidate Franklin Relano Castro created
-----------------------------------------
Votes:
Array: 2 3 5 1 5 5 4 4 3 5 1 1 2 3 2 2 1 3 3 2 2 5 3 4 3 3 2 2 4 1 3 2 2 4 5 3 3 5 1 5 4 2 3 4 4 5 2 2 4 4 3 5 3 3 3 5 4
 2 5 4 2 1 1 3 4 2 1 3 5 4 2 1 1 5 1 1 2 2 4 4 5 4 5 5 2 1 2 2 2 4 5 5 2 4 3 4 4 3 5 2
-----------------------------------------
Sorting Votes:
Selection Sort:
Swaps: 69
Comparisons: 4950
Array: 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 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
-----------------------------------------
Counting Votes...
Successfully counted...
Candidate Bo Dalton Capistrano has 14 votes.
Candidate Cornelius Raymon Agustin has 25 votes.
Candidate Deja Jayla Banaga has 20 votes.
Candidate Lalla Brielle Yabut has 21 votes.
Candidate Franklin Relano Castro has 20 votes.
-----------------------------------------
Copy candidate Bo Dalton Capistrano created
Candidate Bo Dalton Capistrano deleted
Copy candidate Bo Dalton Capistrano created
Candidate Bo Dalton Capistrano deleted
Copy candidate Bo Dalton Capistrano created
Candidate Bo Dalton Capistrano deleted
-----------------------------------------
Candidates from most votes to least:
Candidate Cornelius Raymon Agustin has 25 votes.
Candidate Lalla Brielle Yabut has 21 votes.
Candidate Deja Jayla Banaga has 20 votes.
Candidate Franklin Relano Castro has 20 votes.
Candidate Bo Dalton Capistrano has 14 votes.
-----------------------------------------
The winner is:
Candidate Cornelius Raymon Agustin has 25 votes.
-----------------------------------------
Candidate Bo Dalton Capistrano deleted
Candidate Franklin Relano Castro deleted
Candidate Deja Jayla Banaga deleted
Candidate Lalla Brielle Yabut deleted
Candidate Cornelius Raymon Agustin deleted

Process finished with exit code 0
```

Explanation

I first created a header for the candidates to make creation easier and more intuitive. The header file has a class Candidates with corresponding constructors and printing functions. There I included the candidate name and number of votes. Then, I created the candidates as an array of candidates to make sorting of votes easier. Afterwards, in the main driver function, I included all of my headers: candidates.h, searching.h, and sorting.h. I began by creating a random number generator function that creates 100 numbers from 1 - 5. Then, in main, I initialized the candidates using the parametized constructor and set their starting votes to 0. Afterwards, I generated the random votes, sorted it using increasing order, and I used the selection sort because it requires less swapping and can be generally faster especially since we're dealing with a large array of numbers. After sorting them, I called my countVotes function:

```
inline void countVotes(Candidates candidates[], int votes[], int numCandidates,
int voteArraySize) {
    for (int i = 0; i < numCandidates; ++i) {
        candidates[i].votes = countOccurrencesArray(votes, voteArraySize, i + 1);
    }
}
```

To not only count all of the 1's, 2's, and so on in the vote array, but to also assign it immediately to the corresponding candidate which has an index since I created it as an array. I passed the candidates array, the votes array, then the sizes of both arrays. I implemented my countOccurencesArray method in the searching.h header to make things easier since it counts all instances of a target value in the given array. This makes sorting the votes of each candidate easier.

Then I called my sortCandidatesByVotes function in the candidates class to sort the candidates array using a selection sort logic:

```
inline void sortCandidatesByVotes(Candidates arr[], int size) {
    for (int i = 0; i < size - 1; ++i) {
        int maxIndex = i;
        for (int j = i + 1; j < size; ++j) {
            if (arr[j].votes > arr[maxIndex].votes) {
                maxIndex = j;
            }
        }
        if (maxIndex != i) {
            Candidates temp = arr[i];
            arr[i] = arr[maxIndex];
            arr[maxIndex] = temp;
        }
    }
}
```

This arranges my candidates from most votes to least so that I can immediately declare the winner as the first element at index 0:

```
std::cout << "The winner is: \n";
cDisplay(c[0]);
```

```
The winner is:
Candidate Cornelius Raymon Agustin has 25 votes.
-----------------------------------------
```

8. Conclusion

In conclusion, I explored the advantages and disadvantages of the three different types of algorithms. Bubble, the more beginner friendly algorithm, is generally slower and less memory efficient than the others. Selection is by far the most efficient from my testing as it does the least amount of swapping but I also know that it cannot immediately recognize an already sorted array. Lastly, insertion sorting is by far the most confusing sorting algorithm due to the constant storing of temporary variables before swapping and inserting. I understood the logic of selection sorting the most which is why I implemented it in the supplementary activity.

Additionally, I was able to implement my selection and sorting algorithms into one project which showed me the efficiency of these algorithms. I was able to practice and recognize what algorithm is the best to use for any situation. Sorting and searching algorithms go hand-in-hand especially in binary searching which requires a sorted array to work. By learning how to implement both, I can create a great program that is memory efficient, effective and quick, and user friendly.

9. Assessment Rubric