Activity No. 7

SORTING ALGORITHMS: BUBBLE, SELECTION, AND INSERTION SORT

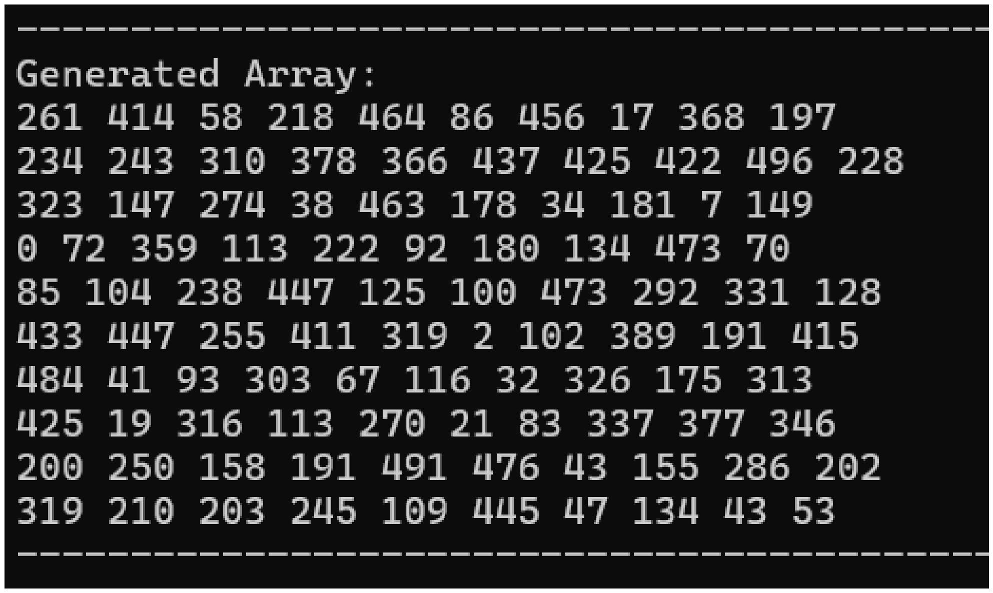
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 10/16/24
Section: CPE21S4	**Date Submitted:** 10/18/24
Name(s): Magistrado, Aira Pauleen M.	**Instructor:** Maria Rizette Sayo

6. Output

Code + Console Screenshot

C/C++

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

const int max_size = 100;

void printArray(const int arr[], int size);

int main() {
    int dataset[max_size];
    srand(time(0));  // Seed random number generator

    // Generate random numbers for the dataset
    for (int i = 0; i < max_size; ++i) {
        dataset[i] = rand() % 500;  // Random numbers between
0 and 499
    }

    std::cout <<
"-------------------------------------------------------------
-----------------------------------------------------------"
<< std::endl;
    std::cout << "Generated Array: " << std::endl;
    printArray(dataset, max_size);
    std::cout <<
"-------------------------------------------------------------
-----------------------------------------------------------"
<< std::endl;
    return 0;
}

void printArray(const int arr[], int size) {
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
        if ((i + 1) % 10 == 0) {
            std::cout << std::endl;
        }
    }
}
```

	
Observations	The code generates a random array of 100 integers and prints it to the console

```
--------------------------------------------
Generated Array:
261 414 58 218 464 86 456 17 368 197
234 243 310 378 366 437 425 422 496 228
323 147 274 38 463 178 34 181 7 149
0 72 359 113 222 92 180 134 473 70
85 104 238 447 125 100 473 292 331 128
433 447 255 411 319 2 102 389 191 415
484 41 93 303 67 116 32 326 175 313
425 19 316 113 270 21 83 337 377 346
200 250 158 191 491 476 43 155 286 202
319 210 203 245 109 445 47 134 43 53
--------------------------------------------
```

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

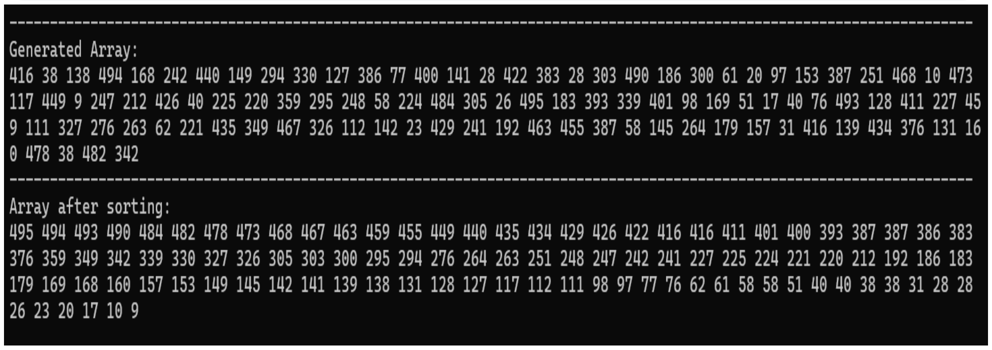
Code + Console Screenshot	

```cpp
C/C++
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "sorting_algorithms.h"

const int max_size = 100;

void printArray(const int arr[], int size);

int main() {
    int dataset[max_size];
    srand(time(0));  // Seed random number generator

    // Generate random numbers for the dataset
    for (int i = 0; i < max_size; ++i) {
        dataset[i] = rand() % 500;
    }

    std::cout <<
"-------------------------------------------------------------
-----------------------------------------------------------"
<< std::endl;
    std::cout << "Generated Array: " << std::endl;
    printArray(dataset, max_size);

    std::cout <<
"-------------------------------------------------------------
-----------------------------------------------------------"
<< std::endl;
```

```
    bubbleSort(dataset, max_size);

    std::cout << "Array after sorting: " << std::endl;
    printArray(dataset, max_size);

    return 0;
}

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

C/C++

```cpp
#include <cstddef>
#include <algorithm>

// Bubble sort algorithm template
template <typename T>
void bubbleSort(T arr[], size_t arrSize) {
    for (size_t i = 0; i < arrSize - 1; ++i) {
        for (size_t j = i + 1; j < arrSize; ++j) {
            if (arr[j] > arr[i]) {
                std::swap(arr[j], arr[i]);
            }
        }
    }
}
```



```
--------------------------------------------------------------------------------------------------------------------
Generated Array:
416 38 138 494 168 242 440 149 294 330 127 386 77 400 141 28 422 383 28 303 490 186 300 61 20 97 153 387 251 468 10 473
117 449 9 247 212 426 40 225 220 359 295 248 58 224 484 305 26 495 183 393 339 401 98 169 51 17 40 76 493 128 411 227 45
9 111 327 276 263 62 221 435 349 467 326 112 142 23 429 241 192 463 455 387 58 145 264 179 157 31 416 139 434 376 131 16
0 478 38 482 342
--------------------------------------------------------------------------------------------------------------------
Array after sorting:
495 494 493 490 484 482 478 473 468 467 463 459 455 449 440 435 434 429 426 422 416 416 411 401 400 393 387 387 386 383
376 359 349 342 339 330 327 326 305 303 300 295 294 276 264 263 251 248 247 242 241 227 225 224 221 220 212 192 186 183
179 169 168 160 157 153 149 145 142 141 139 138 131 128 127 117 112 111 98 97 77 76 62 61 58 58 51 40 40 38 38 31 28 28
26 23 20 17 10 9
```

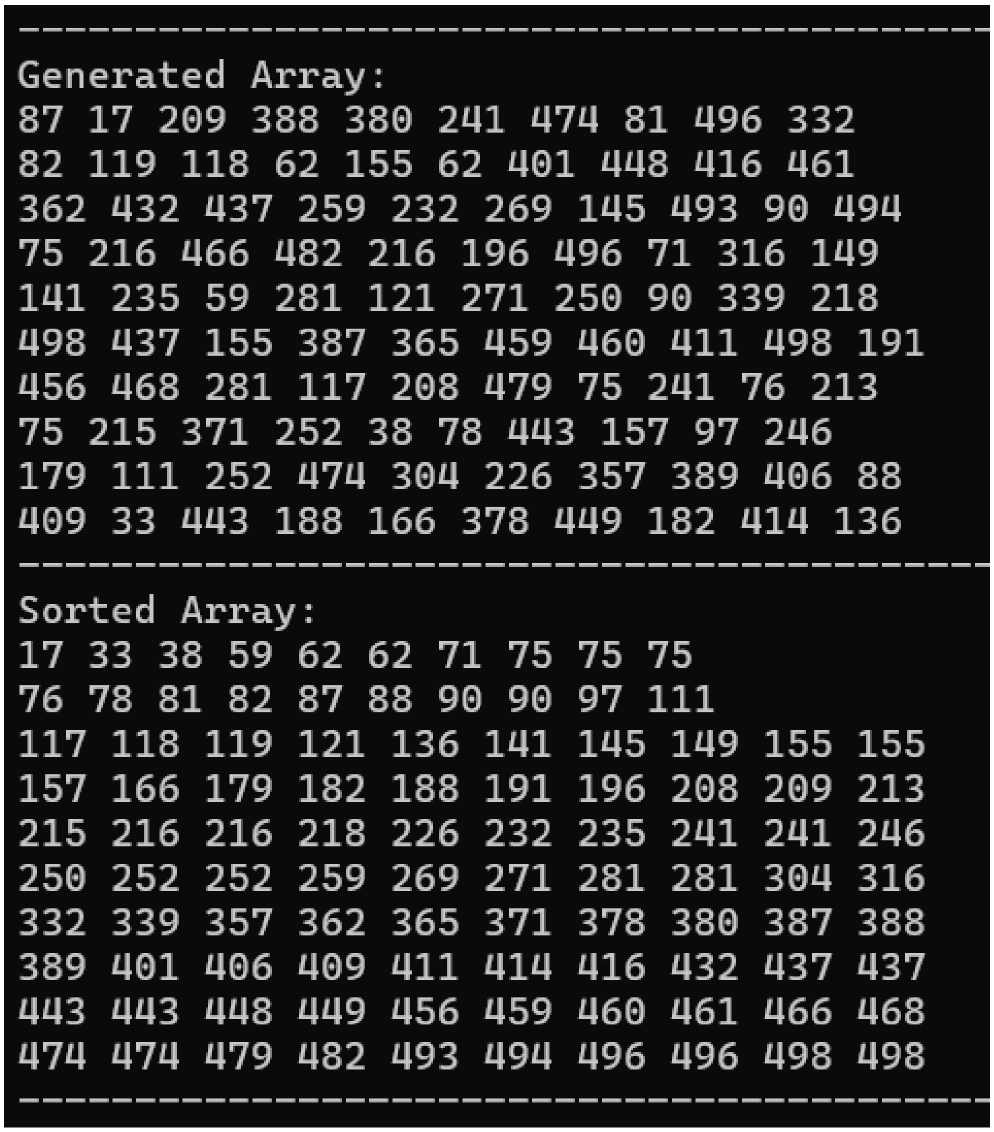
Observations	The program that implements the bubble sort algorithm. The program first generates an array of 100 random integers, then sorts it using the bubble sort algorithm, and finally prints the sorted array to the console.

Table 7-2. Bubble Sort Technique

Code + Console Screenshot	

C/C++

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

const int max_size = 100;

void printArray(const int arr[], int size);

// Function to find the position of the smallest element
template <typename T>
int Routine_Smallest(T A[], int K, const int arrSize) {
    int position = K;
    T smallestElem = A[K];

    for (int J = K + 1; J < arrSize; J++) {
        if (A[J] < smallestElem) {
            smallestElem = A[J];
            position = J;
        }
    }
    return position;  // Return the position of the smallest
element
}

// Selection Sort Algorithm
template <typename T>
void selectionSort(T arr[], const int N) {
    int POS, temp;

    for (int i = 0; i < N; i++) {
        POS = Routine_Smallest(arr, i, N);
        temp = arr[i];
        arr[i] = arr[POS];
        arr[POS] = temp;
    }
}

int main() {
    int dataset[max_size];
    srand(time(0));  // Seed random number generator

    // Generate random numbers for the dataset
    for (int i = 0; i < max_size; ++i) {
        dataset[i] = rand() % 500;  // Random numbers between
0 and 499
    }

    std::cout <<
"------------------------------------------------------------
-----------------------------------------------------------"
<< std::endl;
    std::cout << "Generated Array: " << std::endl;
    printArray(dataset, max_size);

    // Sort the array using selection sort
    selectionSort(dataset, max_size);

    std::cout <<
"------------------------------------------------------------
-----------------------------------------------------------"
<< std::endl;
    std::cout << "Sorted Array: " << std::endl;
    printArray(dataset, max_size);

    return 0;
}

void printArray(const int arr[], int size) {
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
        if ((i + 1) % 10 == 0) {  // Print newline after
every 10 elements
            std::cout << std::endl;
        }
    }
}
```



```
------------------------------------------
Generated Array:
87 17 209 388 380 241 474 81 496 332
82 119 118 62 155 62 401 448 416 461
362 432 437 259 232 269 145 493 90 494
75 216 466 482 216 196 496 71 316 149
141 235 59 281 121 271 250 90 339 218
498 437 155 387 365 459 460 411 498 191
456 468 281 117 208 479 75 241 76 213
75 215 371 252 38 78 443 157 97 246
179 111 252 474 304 226 357 389 406 88
409 33 443 188 166 378 449 182 414 136
------------------------------------------
Sorted Array:
17 33 38 59 62 62 71 75 75 75
76 78 81 82 87 88 90 90 97 111
117 118 119 121 136 141 145 149 155 155
157 166 179 182 188 191 196 208 209 213
215 216 216 218 226 232 235 241 241 246
250 252 252 259 269 271 281 281 304 316
332 339 357 362 365 371 378 380 387 388
389 401 406 409 411 414 416 432 437 437
443 443 448 449 456 459 460 461 466 468
474 474 479 482 493 494 496 496 498 498
------------------------------------------
```

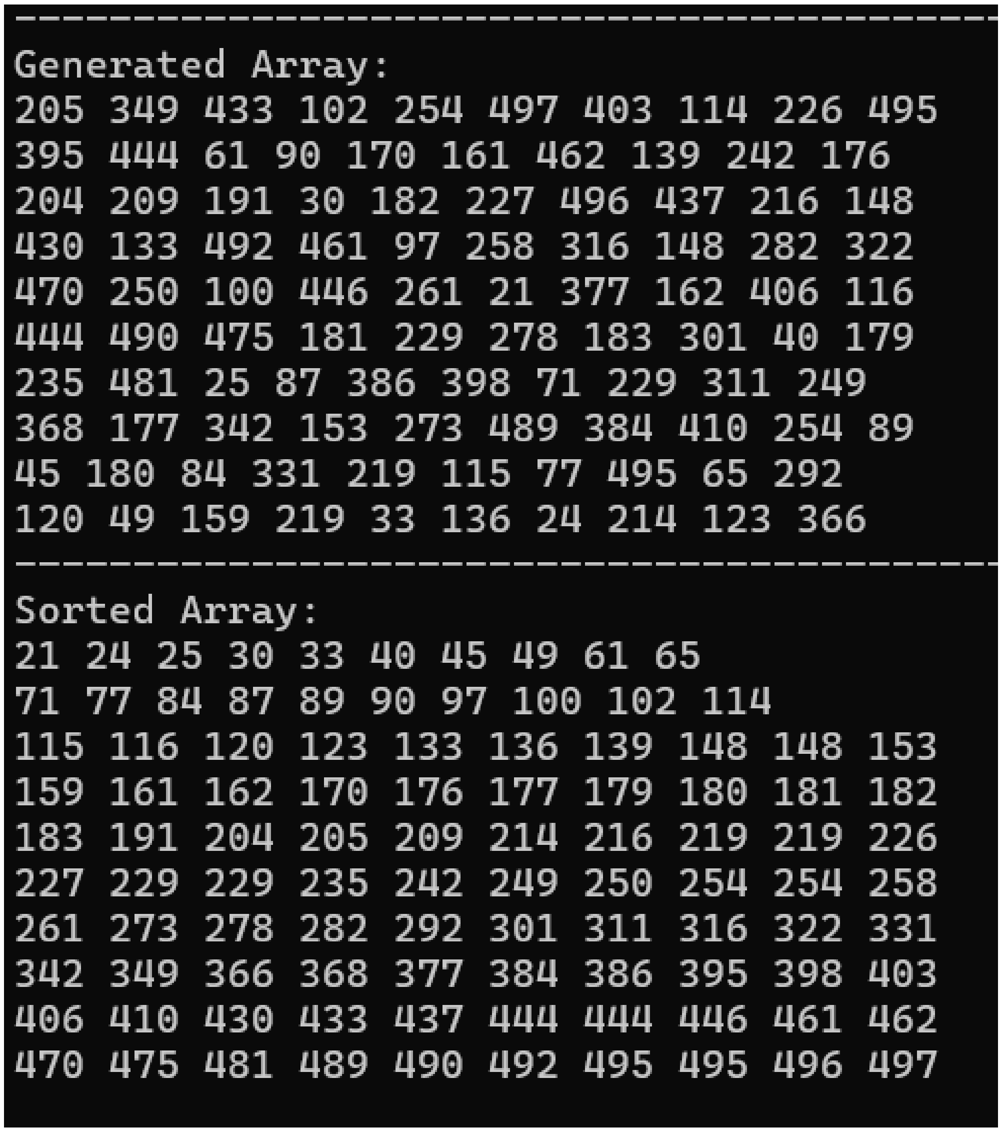
Observations	The program that implements a Selection Sort Algorithm. The program first generates an array of 100 random integers, then sorts it using the Selection sort algorithm. The selection sort works by repeatedly finding the smallest element in the unsorted portion of the array and swapping it with the element at the beginning.

Table 7-3. Selection Sort Algorithm

Code + Console Screenshot

```
C/C++
#include <iostream>
#include <cstdlib>
#include <ctime>

const int max_size = 100;

void printArray(const int arr[], int size);
```

```
// Insertion Sort Algorithm
template <typename T>
void insertionSort(T arr[], const int N) {
    int K = 1, J;
    T temp;

    // Step 1: Repeat Steps 2 to 5 for K = 1 to N-1
    while (K < N) {
        temp = arr[K];      // Step 2: set temp = A[K]
        J = K - 1;          // Step 3: set J = K − 1

        // Step 4: Repeat while J >= 0 and temp < A[J]
        while (J >= 0 && temp < arr[J]) {
            arr[J + 1] = arr[J]; // set A[J + 1] = A[J]
            J--;                 // set J = J − 1
        }
        arr[J + 1] = temp;      // Step 5: set A[J + 1] = temp
        K++;                    // Increment K for the next
iteration
    }
    // Step 6: exit
}

int main() {
    int dataset[max_size];
    srand(time(0));  // Seed random number generator

    // Generate random numbers for the dataset
    for (int i = 0; i < max_size; ++i) {
        dataset[i] = rand() % 500;  // Random numbers between
0 and 499
    }

    std::cout <<
"-------------------------------------------------------------
----------------------------------------------------------" <<
std::endl;
    std::cout << "Generated Array: " << std::endl;
    printArray(dataset, max_size);

    // Sort the array using insertion sort
    insertionSort(dataset, max_size);

    std::cout <<
"-------------------------------------------------------------
----------------------------------------------------------" <<
std::endl;
    std::cout << "Sorted Array: " << std::endl;
    printArray(dataset, max_size);

    return 0;
}

void printArray(const int arr[], int size) {
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
        if ((i + 1) % 10 == 0) {  // Print newline after every
10 elements
            std::cout << std::endl;
        }
    }
}
```



```
---------------------------------------------
Generated Array:
205 349 433 102 254 497 403 114 226 495
395 444 61 90 170 161 462 139 242 176
204 209 191 30 182 227 496 437 216 148
430 133 492 461 97 258 316 148 282 322
470 250 100 446 261 21 377 162 406 116
444 490 475 181 229 278 183 301 40 179
235 481 25 87 386 398 71 229 311 249
368 177 342 153 273 489 384 410 254 89
45 180 84 331 219 115 77 495 65 292
120 49 159 219 33 136 24 214 123 366
---------------------------------------------
Sorted Array:
21 24 25 30 33 40 45 49 61 65
71 77 84 87 89 90 97 100 102 114
115 116 120 123 133 136 139 148 148 153
159 161 162 170 176 177 179 180 181 182
183 191 204 205 209 214 216 219 219 226
227 229 229 235 242 249 250 254 254 258
261 273 278 282 292 301 311 316 322 331
342 349 366 368 377 384 386 395 398 403
406 410 430 433 437 444 444 446 461 462
470 475 481 489 490 492 495 495 496 497
```

Observations	The program that implements an Insertion Sort Algorithm. The program first generates an array of 100 random integers, then sorts it using the Insertion sort algorithm. The numbers are now sorted in ascending order. The insertion algorithm goes repeatedly through the list comparing each number on the array before inserting it to their ascending order positions.

Table 7-4. Insertion Sort Algorithm

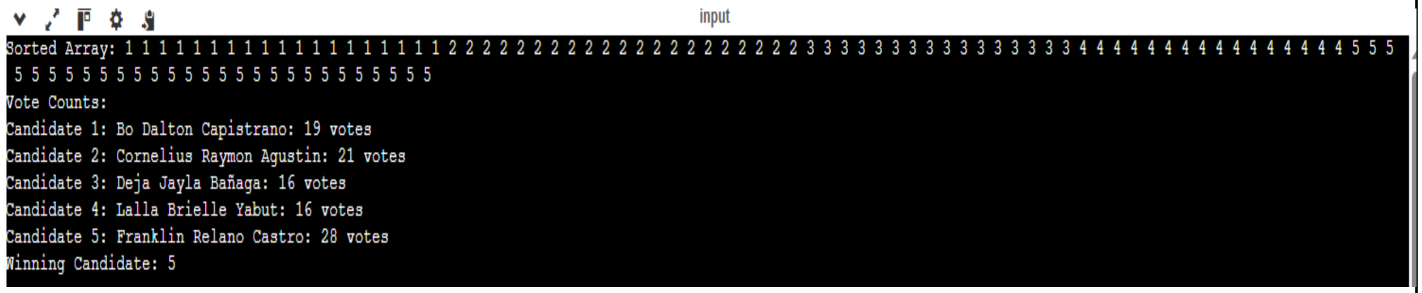
C/C++

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

const int max_size = 100;

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

int main() {
    int votes[max_size];
    srand(time(0));
    for (int i = 0; i < max_size; i++) {
        votes[i] = rand() % 5 + 1;
    }

    // Sort the array using bubble sort
    bubbleSort(votes, max_size);

    // Count the votes for each candidate
    int candidate_counts[5] = {0};
    for (int i = 0; i < max_size; i++) {
        candidate_counts[votes[i] - 1]++;
    }

    // Find the winning candidate
    int winning_candidate = 1;
    int max_votes = candidate_counts[0];
    for (int i = 1; i < 5; i++) {
        if (candidate_counts[i] > max_votes) {
            winning_candidate = i + 1;
            max_votes = candidate_counts[i];
        }
    }

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

    std::cout << "Vote Counts: " << std::endl;
    std::cout << "Candidate 1: Bo Dalton Capistrano: " << candidate_counts[0] << " votes "
<< std::endl; // Candidate 1
    std::cout << "Candidate 2: Cornelius Raymon Agustin: " << candidate_counts[1] << "
votes " << std::endl; // Candidate 2
```

```
    std::cout << "Candidate 3: Deja Jayla Bañaga: " << candidate_counts[2] << " votes " <<
std::endl; // Candidate 3
    std::cout << "Candidate 4: Lalla Brielle Yabut: " << candidate_counts[3] << " votes "
<< std::endl; // Candidate 4
    std::cout << "Candidate 5: Franklin Relano Castro: " << candidate_counts[4] << " votes
" << std::endl; // Candidate 5

    std::cout << "Winning Candidate: " << winning_candidate << std::endl;

    return 0;
}
```

Pseudocode:
```
Create an array to store 100 votes.
Use random numbers between 1 and 5 to represent votes for 5 candidates.
Compare each vote with the next one.
If a vote is bigger than the next one, swap them.
Repeat until all votes are in order.
Go through the sorted votes.
Add 1 to the count for each candidate based on the vote value (1 to 5).
Compare the vote counts for all 5 candidates.
The candidate with the highest vote count is the winner.
Display the sorted votes.
Show how many votes each candidate received.
Announce the candidate with the most votes as the winner.
```

Why did you choose Bubble sort?
I chose the bubble sort technique because it is easy to implement, to understand, and debug. It is good to use for small scale programs.



```
Sorted Array: 1 1 1 1 1 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 2 2 2 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 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
Vote Counts:
Candidate 1: Bo Dalton Capistrano: 19 votes
Candidate 2: Cornelius Raymon Agustin: 21 votes
Candidate 3: Deja Jayla Bañaga: 16 votes
Candidate 4: Lalla Brielle Yabut: 16 votes
Candidate 5: Franklin Relano Castro: 28 votes
Winning Candidate: 5
```

Output Console Showing Sorted Array	Manual Count	Count Result of Algorithm
1 1 1 1 1 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 2 2 2 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 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	Candidate 1 (Bo Dalton Capistrano): 19 votes Candidate 2 (Cornelius Raymon Agustin): 21 votes Candidate 3 (Deja Jayla Bañaga): 16 votes Candidate 4 (Lalla Brielle Yabut): 16 votes Candidate 5 (Franklin Relano Castro): 28 votes	Vote Counts: Candidate 1: Bo Dalton Capistrano: 19 votes Candidate 2: Cornelius Raymon Agustin: 21 votes Candidate 3: Deja Jayla Bañaga: 16 votes Candidate 4: Lalla Brielle Yabut: 16 votes Candidate 5: Franklin Relano Castro:

		28 votes

7. Supplementary Activity

8. Conclusion

In conclusion, I have learned about the different sort techniques including the Bubble Sort, Selection Sort and Insertion Sort. Bubble Sort organizes numbers in an array from smallest to biggest. It continues to compare and swap pairs until it reaches the end of the list or array. Each time this process happens the largest number would bubble up to the end of the list. The selection sort works by repeatedly finding the smallest element in the unsorted portion of the array and swapping it with the element at the beginning. The selection sort might be simple but it is not good to use for large datasets. The insertion algorithm goes repeatedly through the list comparing each number on the array before inserting it to their ascending order positions.

9. Assessment Rubric
