

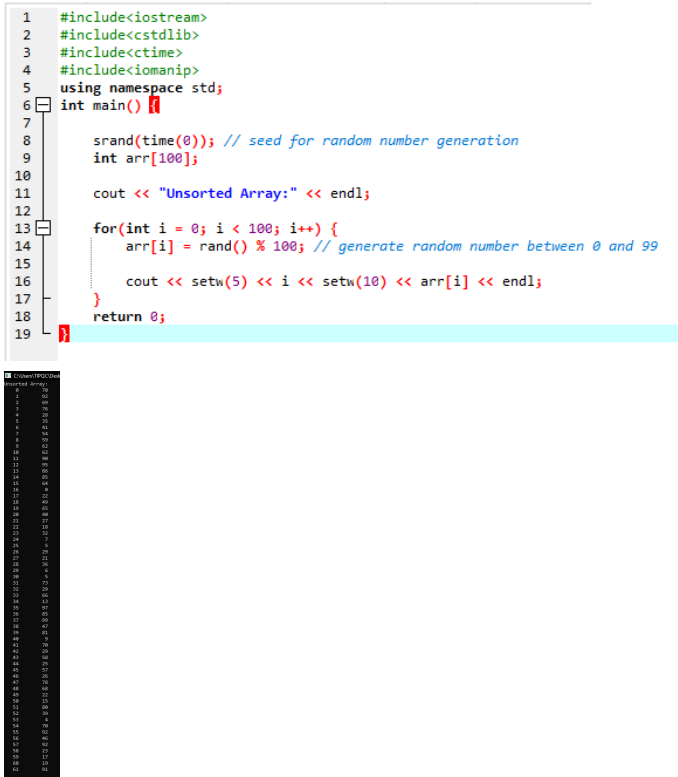
Activity No. 7	
SORTING ALGORITHMS: BUBBLE, SELECTION, AND INSERTION SORT	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: 10/16/2024
Section: CPE21S4	Date Submitted: 10/16/2024
Name(s): Mamaril, Justin Kenneth I.	Instructor: Prof. Sayo
6. Output	
Code + Console Screenshot	
Observations	<p>The given code is a C++ program that generates an array of 100 random integers between 0 and 99 and prints the unsorted array.</p> <p>The code includes necessary header files, seeds the random number generator with the current time, and declares an array to store the random integers. It then uses a for loop to generate and store the random numbers, and prints the index and value of each element with proper alignment using the setw function</p>

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

Code + Console Screenshot

```
1  #include<iostream>
2  using namespace std;
3
4  void bubbleSort(int arr[], int n) {
5      for(int i = 0; i < n-1; i++) {
6          for(int j = 0; j < n-i-1; j++) {
7              if(arr[j] > arr[j+1]) {
8                  int temp = arr[j];
9                  arr[j] = arr[j+1];
10
11                  arr[j+1] = temp;
12              }
13          }
14      }
15  }
16
17 void printArray(int arr[], int size) {
18     for (int i = 0; i < size; i++)
19         cout << arr[i] << " ";
20     cout << endl;
21 }
22
23 int main() {
24     int data[] = {5, 3, 4, 1, 2};
25     int n = sizeof(data)/sizeof(data[0]);
26     bubbleSort(data, n);
27     cout<<"Sorted array: \n";
28     printArray(data, n);
29     return 0;
30 }
```

C:\Users\TIPQC\Desktop\Untitled7.exe

Sorted array:
1 2 3 4 5

Process exited after 0.02184 seconds with return value 0
Press any key to continue . . .

Observations

This program implements the bubble sort algorithm to sort an array of integers in ascending order. The program consists of two functions: bubbleSort and printArray. The bubbleSort function takes an array and its size as input, and sorts the array using the bubble sort algorithm. The printArray function is used to print the elements of the array. In the main function, an array of integers is declared and initialized, and then passed to the bubbleSort function to sort it. Finally, the sorted array is printed using the printArray function.

Table 7-2. Bubble Sort Technique

Code + Console Screenshot

```
1  #include<iostream>
2  using namespace std;
3
4  void selectionSort(int arr[], int n) {
5      for(int i = 0; i < n-1; i++) {
6          int min_idx = i;
7          for(int j = i+1; j < n; j++) {
8              if(arr[j] < arr[min_idx])
9                  min_idx = j;
10         }
11         int temp = arr[min_idx];
12         arr[min_idx] = arr[i];
13         arr[i] = temp;
14     }
15 }
16 void printArray(int arr[], int size) {
17     for (int i = 0; i < size; i++)
18         cout << arr[i] << " ";
19     cout << endl;
20 }
21
22 int main() {
23     int data[] = {5, 3, 4, 1, 2};
24     int n = sizeof(data)/sizeof(data[0]);
25     selectionSort(data, n);
26     cout<<"Sorted array: \n";
27     printArray(data, n);
28     return 0;
29 }
```

```
C:\Users\TIPQC\Desktop\MAMARIL_SORTING ALGORITHMS BUBBLE, SELECTION, AN
Sorted array:
1 2 3 4 5

-----
Process exited after 0.03504 seconds with return value 0
Press any key to continue . . .
```

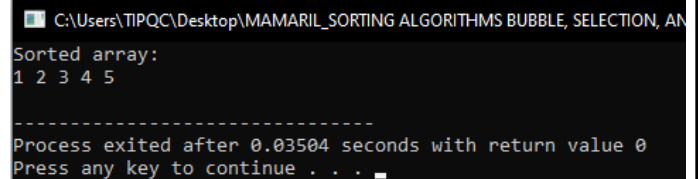
Observations

This program implements the selection sort algorithm to sort an array of integers in ascending order. The program consists of two functions: selectionSort and printArray. The selectionSort function takes an array and its size as input, and sorts the array by repeatedly finding the minimum element from the unsorted part and swapping it with the first unsorted element. The printArray function is used to print the elements of the array. In the main function, an array of integers is declared and initialized, and then passed to the selectionSort function to sort it. Finally, the sorted array is printed using the printArray function.

Table 7-3. Selection Sort Algorithm

Code + Console Screenshot

```
1  #include<iostream>
2  using namespace std;
3
4  void insertionSort(int arr[], int n) {
5      for(int i = 1; i < n; i++) {
6          int key = arr[i];
7          int j = i - 1;
8          while(j >= 0 && arr[j] > key) {
9              arr[j + 1] = arr[j];
10             j--;
11         }
12         arr[j + 1] = key;
13     }
14 }
15 void printArray(int arr[], int size) {
16     for (int i = 0; i < size; i++)
17         cout << arr[i] << " ";
18     cout << endl;
19 }
20
21 int main() {
22     int data[] = {5, 3, 4, 1, 2};
23     int n = sizeof(data)/sizeof(data[0]);
24     insertionSort(data, n);
25     cout<<"Sorted array: \n";
26     printArray(data, n);
27     return 0;
28 }
```



```
C:\Users\TIPQC\Desktop\MAMARIL_SORTING ALGORITHMS BUBBLE, SELECTION, AN
Sorted array:
1 2 3 4 5

Process exited after 0.03504 seconds with return value 0
Press any key to continue . . .
```

Observations

This program implements the insertion sort algorithm to sort an array of integers in ascending order. The program consists of two functions: `insertionSort` and `printArray`. The `insertionSort` function takes an array and its size as input, and sorts the array by iterating through the array one element at a time, inserting each element into its proper position in the sorted part of the array. The `printArray` function is used to print the elements of the array. In the `main` function, an array of integers is declared and initialized, and then passed to the `insertionSort` function to sort it. Finally, the sorted array is printed using the `printArray` function.

Table 7-4. Insertion Sort Algorithm

7. Supplementary Activity

Source Code:

```
1  #include <iostream>
2  #include <ctime>
3  #include <cstdlib>
4
5  using namespace std;
6
7  void insertionSort(int A[], int n) {
8      for (int i = 1; i < n; i++) {
9          int key = A[i];
10         int j = i - 1;
11         while (j >= 0 && A[j] > key) {
12             A[j + 1] = A[j];
13             j--;
14         }
15         A[j + 1] = key;
16     }
17 }
18
19 void countVotes(int A[], int n, int votes[]) {
20     for (int i = 0; i < n; i++) {
21         votes[A[i] - 1]++; // Increment the vote count for each candidate
22     }
23 }
24
25 int findWinner(int votes[]) {
26     int maxVotes = 0;
27     int winner = 0;
28     for (int i = 0; i < 5; i++) {
29         if (votes[i] > maxVotes) {
30             maxVotes = votes[i];
31             winner = i + 1; // Candidate number (1 to 5)
32         }
33     }
34     return winner;
35 }
36
```

```

36
37 int main() {
38     srand(time(0)); // Seed for random number generation
39     const int n = 101; // Size of the array
40     int A[n]; // Array to store votes
41     int votes[5] = {0}; // Array to store votes for each candidate
42
43     // Generate random votes
44     for (int i = 0; i < n; i++) {
45         A[i] = rand() % 5 + 1; // Random value between 1 and 5
46     }
47
48     // Print the original array
49     cout << "Original Array A: ";
50     for (int i = 0; i < n; i++) {
51         cout << A[i] << " ";
52     }
53     cout << endl;
54
55     // Sort the array using Insertion Sort
56     insertionSort(A, n);
57
58     // Print the sorted array
59     cout << "Sorted Array A: ";
60     for (int i = 0; i < n; i++) {
61         cout << A[i] << " ";
62     }
63     cout << endl;
64
65     // Count the votes
66     countVotes(A, n, votes);
67
68     // Print the votes
69     cout << "Votes: ";
70     for (int i = 0; i < 5; i++) {
71         cout << votes[i] << " ";
72     }
73     cout << endl;
74
75     // Find the winner
76     int winner = findWinner(votes);
77
78     // Print the winner
79     cout << "Winner: Candidate " << winner << " (with " << votes[winner - 1] << " votes)" << endl;
80
81     return 0;
82 }

```

Output Console Showing Sorted Array	Manual Count	Count Result of Algorithm
<pre>Original Array A: 3 4 5 1 2 1 2 1 1 4 4 1 5 5 4 3 5 4 3 4 4 1 5 1 3 1 5 5 4 1 1 2 4 3 5 3 3 2 1 1 4 1 1 1 5 1 3 2 4 5 1 3 2 2 2 5 5 1 2 5 5 3 2 1 1 5 4 2 4 4 5 2 3 5 2 5 5 2 5 2 5 2 2 4 4 3 1 3 4 3 3 5 5 4 3 5 3 4 4 5 1 Sorted Array A: 1 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 4 5</pre>	<pre>Votes: 22 17 17 20 25</pre>	<pre>Winner: Candidate 5 (with 25 votes)</pre>
<pre>Original Array A: 5 5 2 2 2 2 1 2 3 4 2 5 4 3 2 5 3 4 1 3 3 3 4 1 5 5 5 1 5 4 4 1 5 4 4 4 5 1 4 3 2 2 2 1 5 5 1 5 2 2 1 1 1 3 3 1 4 5 3 4 4 5 1 5 2 4 1 5 3 2 4 4 1 1 1 3 1 4 4 5 3 2 3 3 1 5 4 5 5 1 5 1 4 2 2 5 4 3 2 1 1 Sorted Array A: 1 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 5</pre>	<pre>Votes: 23 18 16 21 23</pre>	<pre>Winner: Candidate 1 (with 23 votes)</pre>

<pre>Original Array A: 4 1 2 3 4 5 2 2 2 5 4 1 1 2 5 1 4 2 4 4 1 3 4 3 3 3 3 4 2 5 1 4 5 2 1 4 2 3 3 4 5 3 4 2 2 5 5 2 5 5 4 2 2 2 1 1 3 2 2 4 3 3 5 4 4 2 2 1 4 4 1 3 3 4 2 3 2 5 3 3 4 1 5 4 2 5 2 3 4 1 1 4 5 5 4 2 1 4 1 5 1 Sorted Array A: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5</pre>	<pre>Votes: 17 24 18 25 17</pre>	<pre>Winner: Candidate 4 (with 25 votes)</pre>
--	----------------------------------	--

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

- Yes, because the vote counting algorithm effectively worked and the code generates an array of 101 random votes, sorts it using Insertion Sort, counts the votes for each candidate, and determines the winning candidate. The output will vary depending on the random values generated.

8. Conclusion

- In conclusion, all three algorithms, bubble sort, selection sort, and insertion sort, are simple sorting algorithms with a time complexity of $O(n^2)$ in the worst and average cases. They are suitable for small datasets or nearly sorted arrays, but may not be efficient for large datasets or arrays with a lot of randomness. And in the supplementary activity, we generated an array of 101 random votes, where each vote represents a candidate number between 1 and 5. We used the Insertion Sort algorithm to sort the array, which allowed us to efficiently count the votes for each candidate. By finding the candidate with the maximum votes, we determined the winning candidate. Overall, this problem demonstrates the application of sorting and searching techniques to solve a real-world problem, and highlights the importance of efficient algorithms in data analysis.

9. Assessment Rubric