| **Hands-on Activity 7.1** | |
| --- | --- |
| **Sorting Algorithms** | |
| **Course Code:** CPE010 | **Program:** Computer Engineering |
| **Course Title:** Data Structures and Algorithms | **Date Performed:** 10/16/2024 |
| **Section:** CPE21S1 | **Date Submitted:** 10/16/2024 |
| **Name(s):** Bonifacio, Redj Guillian F. | **Instructor:** Sayo, Maria Rizette |
| **6. Output** | |
| | Code + Console Screenshot |  | | --- | --- | | Observations | | The bubble sort implementation in the code compares each element with the | | --- | | following ones, resulting in a time complexity of O(n^2), which can be inefficient for | | large arrays. Additionally, the sorting function does not account for already sorted | | sections, which can potentially lead to unnecessary comparisons | |   Table 7-1. Array of Values for Sort Algorithm Testing   | Code + Console Screenshot |  | | --- | --- | | Observations | The code implements a bubble sort algorithm that sorts an array of integers in descending order and then prints the sorted array. |   Table 7-2. Bubble Sort Technique   | Code + Console Screenshot |  | | --- | --- | | Observations | | The code implements a selection sort algorithm, utilizing a helper function to f | | --- | | ind the smallest element's index in the remaining unsorted portion of the array. It | | effectively populates the array with random integers, sorts them, and prints both the | | original and sorted arrays. | |   Table 7-3. Selection Sort Algorithm   | Code + Console Screenshot |  | | --- | --- | | Observations |  |   Table 7-4. Insertion Sort Algorithm | |
| **7. Supplementary Activity** | |
| | Pseudocode of Algorithm | | --- | | 1. Generate an array 'votes' of 100 random integers between 1 and 5.  2. Use Bubble Sort to sort the 'votes' array:  a. For i from 0 to n-1:  b. For j from 0 to n-i-2:  i. If votes[j] > votes[j+1], then swap votes[j] and votes[j+1].  3. Initialize a dictionary 'counts' to store the count of each candidate's votes (from 1 to 5).  4. Traverse the sorted 'votes' array and count occurrences:  a. For each vote in 'votes':  i. Increment counts[vote] by 1.  5. Find the candidate with the maximum count in 'counts' to determine the winner.  6. Output the sorted 'votes' array, the counts, and the winner. |  | Screenshot of Algorithm Code | | --- | |  |  | Output Testing | | --- | |  |  | Output Console Showing Sorted Array | Manual Count | Count Result of Algorithm | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | | |
| **8. Conclusion** | |
| |  | | --- | |  | |  | |  | | |
| **9. Assessment Rubric** | |
|  | |