**Problem Statement:** Stepwise Execution Analysis of Sorting Algorithms I

**Bubble Sort**

**Theory:**

* Initialize an integer array with user input
* Sort the array using Bubble Sort:
* Initialize two counters: cntSwap (to count swaps) and cntCompare (to count comparisons).
* For each pass from the first element to the second last element:
* Set a boolean flag swapped to false.
* For each element from the start up to the last unsorted element (which decreases with each

pass):

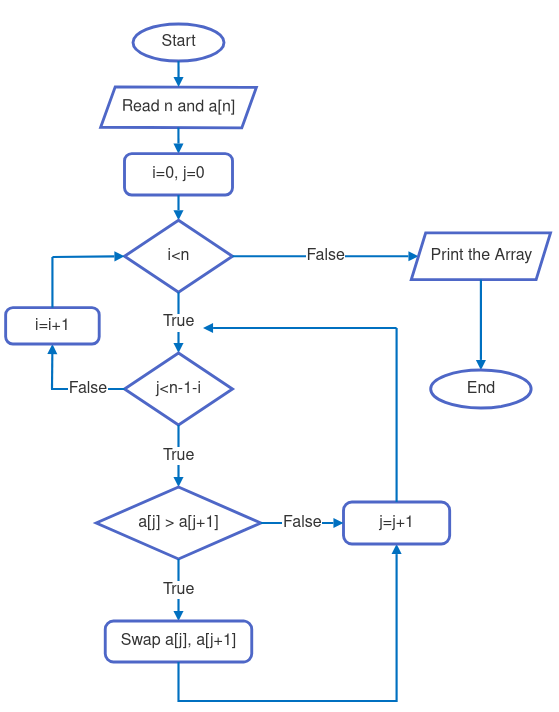
* Compare the current element with the next element.
* Increment the comparison counter cntCompare.
* If the current element is greater than the next element:

Swap the two elements.

Set swapped to true.

* Increment the swap counter cntSwap.
* If no swaps occurred during this pass (swapped is false), the array is sorted; break out of

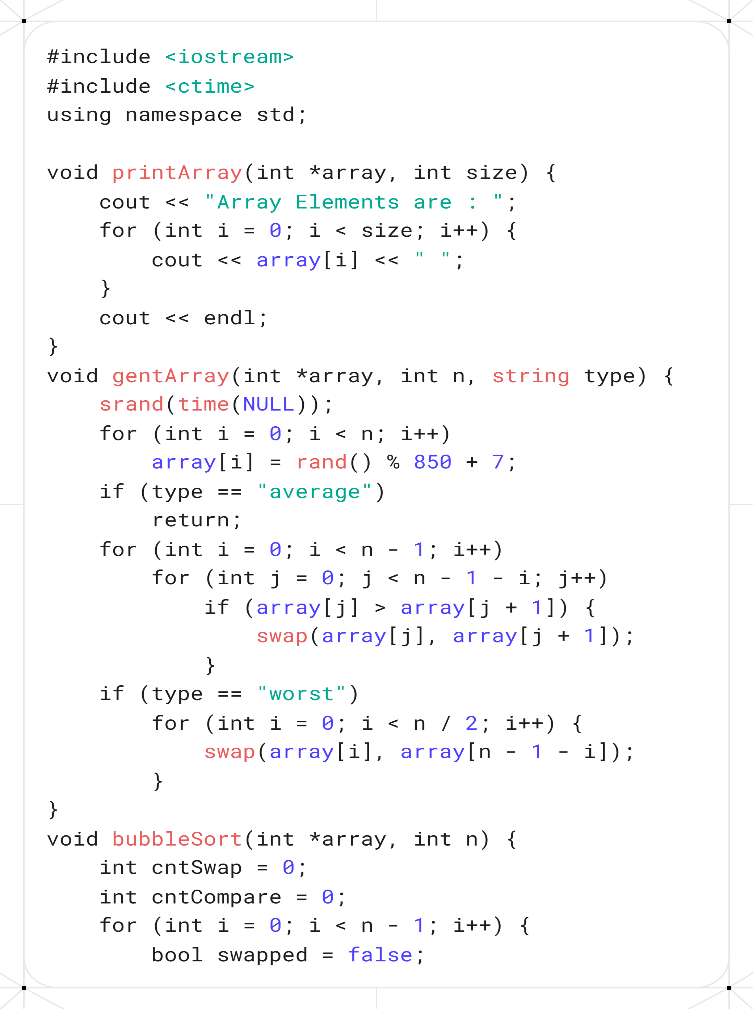
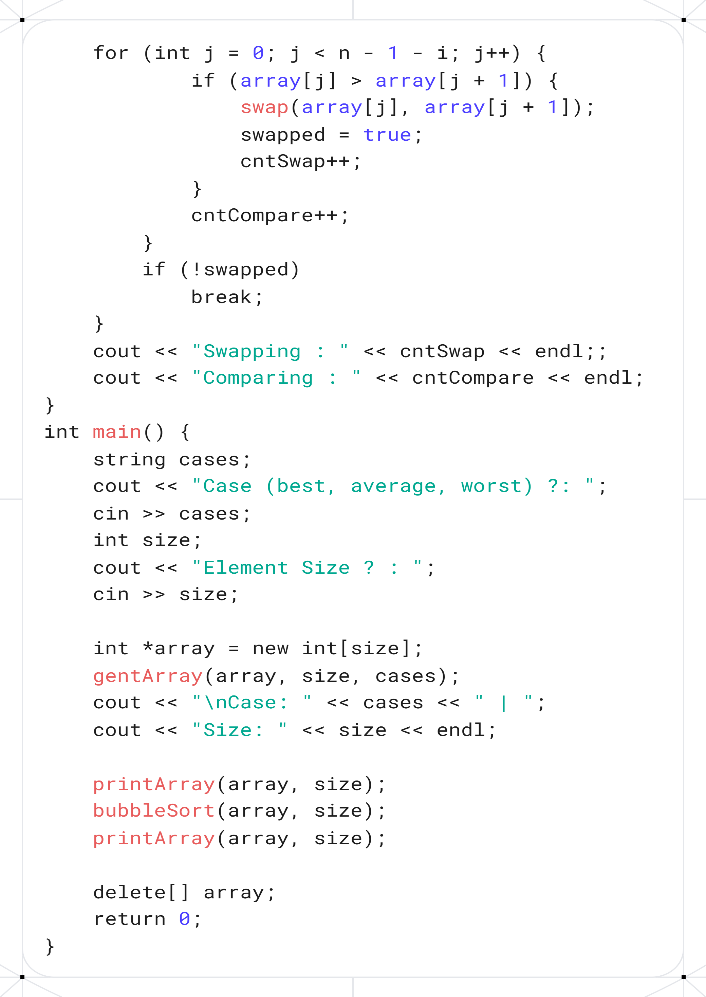
the loop early.

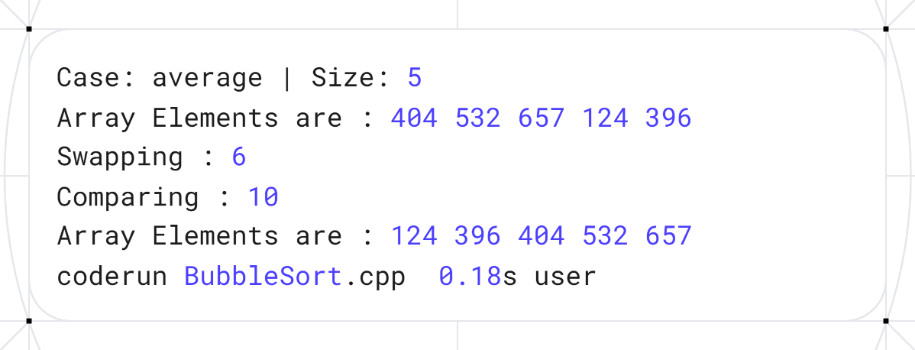
* Print the total number of swaps and comparisons performed during the sorting process.

**Code:**

|  |
| --- |
| 1. #include <iostream> 2. #include <ctime> 3. **using** **namespace** std; 5. **void** printArray(**int** \*array, **int** size) { 6. cout << "Array Elements are : "; 7. **for** (**int** i = 0; i < size; i++) { 8. cout << array[i] << " "; 9. } 10. cout << endl; 11. } 12. **void** gentArray(**int** \*array, **int** n, string type) { 13. **srand**(**time**(NULL)); 14. **for** (**int** i = 0; i < n; i++) 15. array[i] = **rand**() % 850 + 7; 16. **if** (type == "average") 17. **return**; 18. **for** (**int** i = 0; i < n - 1; i++) 19. **for** (**int** j = 0; j < n - 1 - i; j++) 20. **if** (array[j] > array[j + 1]) { 21. swap(array[j], array[j + 1]); 22. } 23. **if** (type == "worst") 24. **for** (**int** i = 0; i < n / 2; i++) { 25. swap(array[i], array[n - 1 - i]); 26. } 27. } 28. **void** bubbleSort(**int** \*array, **int** n) { 29. **int** cntSwap = 0; 30. **int** cntCompare = 0; 31. **for** (**int** i = 0; i < n - 1; i++) { 32. **bool** swapped = **false**; 33. **for** (**int** j = 0; j < n - 1 - i; j++) { 34. **if** (array[j] > array[j + 1]) { 35. swap(array[j], array[j + 1]); 36. swapped = **true**; 37. cntSwap++; 38. } 39. cntCompare++; 40. } 41. **if** (!swapped) 42. **break**; 43. } 44. cout << "Swapping : " << cntSwap << endl;; 45. cout << "Comparing : " << cntCompare << endl; 46. } 47. **int** main() { 48. string cases; 49. cout << "Case (best, average, worst) ?: "; 50. cin >> cases; 51. **int** size; 52. cout << "Element Size ? : "; 53. cin >> size; 55. **int** \*array = **new** **int**[size]; 56. gentArray(array, size, cases); 57. cout << "\nCase: " << cases << " | "; 58. cout << "Size: " << size << endl; 60. printArray(array, size); 61. bubbleSort(array, size); 62. printArray(array, size); 64. **delete**[] array; 65. **return** 0; 66. } |
|  |

**Screenshots:**

****

****

**Analysis:** Input Array: A = [404, 532, 657, 124, 396]

**Stepwise Execution (Pass by Pass):**

**Pass 1 (i = 1):**

Compare 404 and 532 → no swap → [404, 532, 657, 124, 396]

Compare 532 and 657 → no swap → [404, 532, 657, 124, 396]

Compare 657 and 124 → swap → [404, 532, 124, 657, 396]

Compare 657 and 396 → swap → [404, 532, 124, 396, 657]

**Pass 2 (i = 2):**

Compare 404 and 532 → no swap → [404, 532, 124, 396, 657]

Compare 532 and 124 → swap → [404, 124, 532, 396, 657]

Compare 532 and 396 → swap → [404, 124, 396, 532, 657]

**Pass 3 (i = 3):**

Compare 404 and 124 → swap → [124, 404, 396, 532, 657]

Compare 404 and 396 → swap → [124, 396, 404, 532, 657]

**Pass 4 (i = 4):**

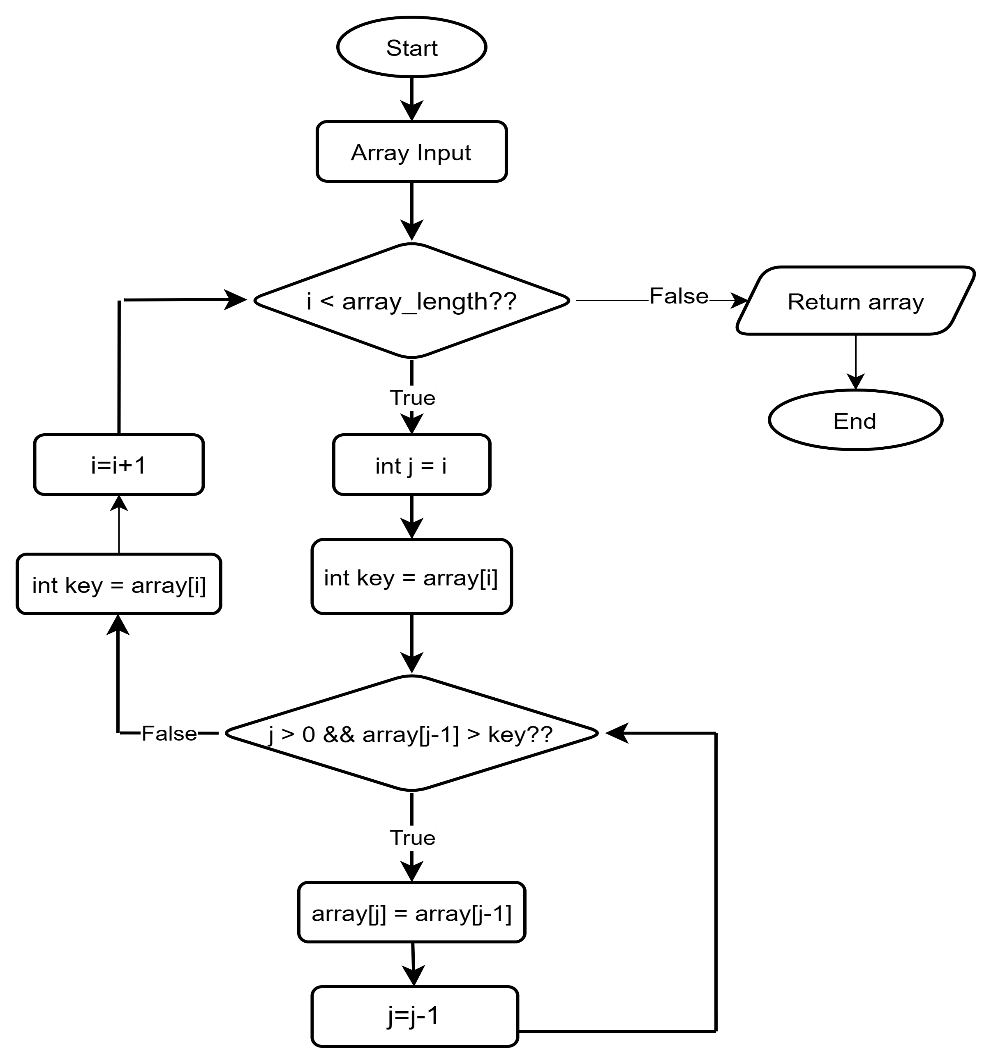
Compare 124 and 396 → no swap → [124, 396, 404, 532, 657]

**Final Sorted Array**: [124, 396, 404, 532, 657]

**Insertion Sort**

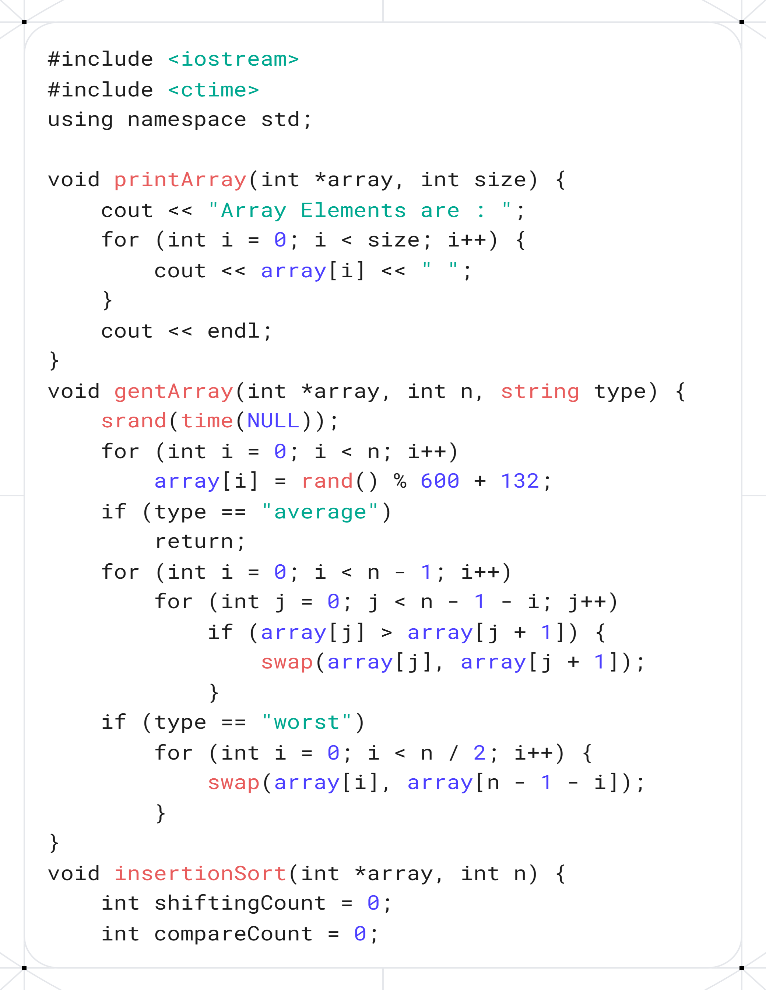
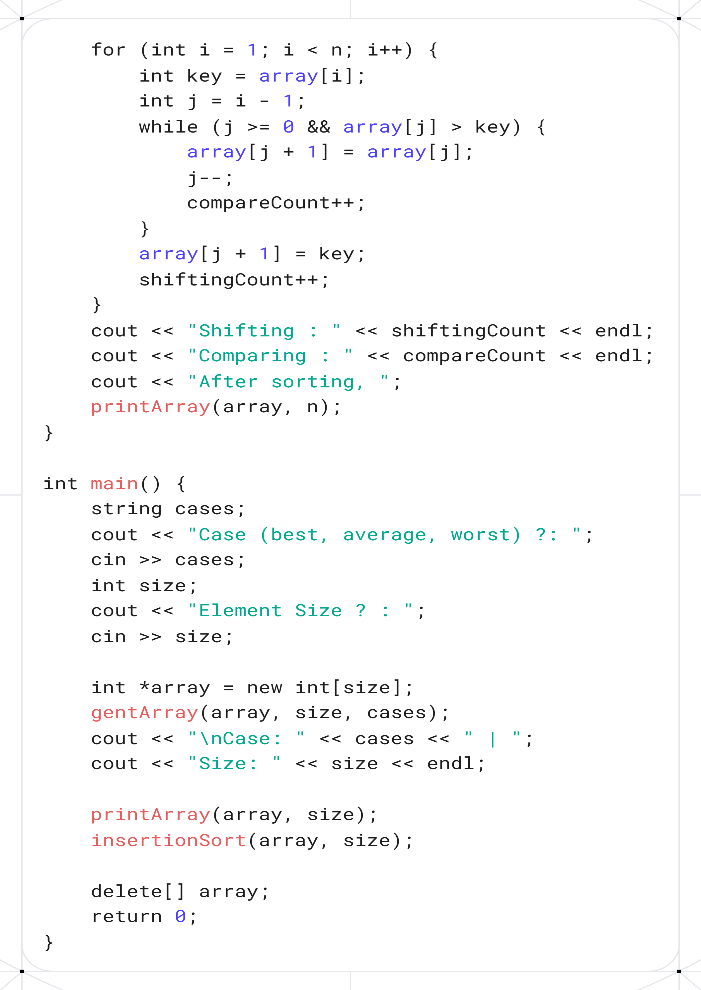
**Theory:**

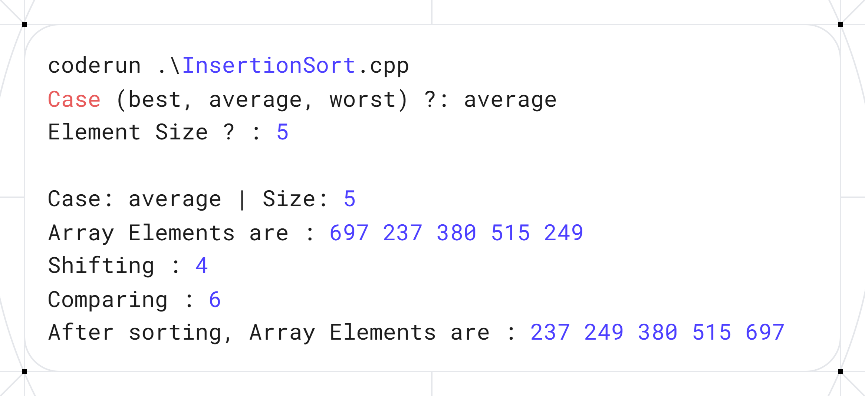
1. Initialize an integer array with user input.
2. Print the array elements:
   * Iterate through the array from the first to the last element.
   * Print each element followed by a space.
3. Sort the array using Insertion Sort:
   * Initialize two counters: shiftingCount (to count the number of shifts) and compareCount (to count the number of comparisons).
   * For each element from the second element (index 1) to the last:
     + Store the current element as key.
     + Initialize j as the index before the current element.
     + While j is non-negative and the element at j is greater than key:
       - Shift the element at j one position to the right (to j + 1).
       - Decrement j.
       - Increment the comparison counter compareCount.
     + Insert the key at position j + 1.
     + Increment the shifting counter shiftingCount.
4. Print the total number of shifts and comparisons performed during sorting.
5. Print the sorted array elements:
   * Iterate through the array and print each element followed by a space.



**Code:**

1. #include <iostream>
2. #include <ctime>
3. **using** **namespace** std;
5. **void** printArray(**int** \*array, **int** size) {
6. cout << "Array Elements are : ";
7. **for** (**int** i = 0; i < size; i++) {
8. cout << array[i] << " ";
9. }
10. cout << endl;
11. }
12. **void** gentArray(**int** \*array, **int** n, string type) {
13. **srand**(**time**(NULL));
14. **for** (**int** i = 0; i < n; i++)
15. array[i] = **rand**() % 600 + 132;
16. **if** (type == "average")
17. **return**;
18. **for** (**int** i = 0; i < n - 1; i++)
19. **for** (**int** j = 0; j < n - 1 - i; j++)
20. **if** (array[j] > array[j + 1]) {
21. swap(array[j], array[j + 1]);
22. }
23. **if** (type == "worst")
24. **for** (**int** i = 0; i < n / 2; i++) {
25. swap(array[i], array[n - 1 - i]);
26. }
27. }
28. **void** insertionSort(**int** \*array, **int** n) {
29. **int** shiftingCount = 0;
30. **int** compareCount = 0;
31. **for** (**int** i = 1; i < n; i++) {
32. **int** key = array[i];
33. **int** j = i - 1;
34. **while** (j >= 0 && array[j] > key) {
35. array[j + 1] = array[j];
36. j--;
37. compareCount++;
38. }
39. array[j + 1] = key;
40. shiftingCount++;
41. }
42. cout << "Shifting : " << shiftingCount << endl;
43. cout << "Comparing : " << compareCount << endl;
44. cout << "After sorting, ";
45. printArray(array, n);
46. }
48. **int** main() {
49. string cases;
50. cout << "Case (best, average, worst) ?: ";
51. cin >> cases;
52. **int** size;
53. cout << "Element Size ? : ";
54. cin >> size;
56. **int** \*array = **new** **int**[size];
57. gentArray(array, size, cases);
58. cout << "\nCase: " << cases << " | ";
59. cout << "Size: " << size << endl;
61. printArray(array, size);
62. insertionSort(array, size);
63. **delete**[] array;
64. **return** 0;
65. }

**Screenshots:**

****

**Analysis:** Input Array: A = [697, 237, 380, 515, 249]

#### Stepwise Execution (Pass by Pass):

**Pass 1 (i = 1):**

Key = 237

Compare 697 and 237 → 697 > 237 → shift 697 right → [697, 697, 380, 515, 249]

Insert 237 at position 0 → [237, 697, 380, 515, 249]

**Pass 2 (i = 2):**

Key = 380

Compare 697 and 380 → 697 > 380 → shift 697 right → [237, 697, 697, 515, 249]

Compare 237 and 380 → 237 < 380 → stop shifting

Insert 380 at position 1 → [237, 380, 697, 515, 249]

**Pass 3 (i = 3):**

Key = 515

Compare 697 and 515 → 697 > 515 → shift 697 right → [237, 380, 697, 697, 249]

Compare 380 and 515 → 380 < 515 → stop shifting

Insert 515 at position 2 → [237, 380, 515, 697, 249]

**Pass 4 (i = 4):**

Key = 249

Compare 697 and 249 → 697 > 249 → shift 697 right → [237, 380, 515, 697, 697]

Compare 515 and 249 → 515 > 249 → shift 515 right → [237, 380, 515, 515, 697]

Compare 380 and 249 → 380 > 249 → shift 380 right → [237, 380, 380, 515, 697]

Compare 237 and 249 → 237 < 249 → stop shifting

Insert 249 at position 1 → [237, 249, 380, 515, 697]

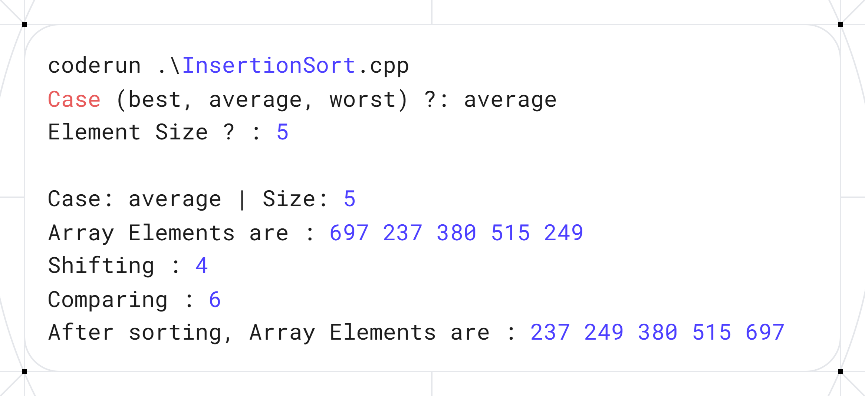
### **Final Sorted Array:** [237, 249, 380, 515, 697]

**Selection Sort**

**Theory:**.

**Code:**

**Screenshots:**

****

**Analysis:**

**Observation Table**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case No. | Input Size (n) | Number of Comparison | | Number of Swap/Shift | | Execution Time | | |
| Bubble Sort | Selection Sort | Bubble Sort | Selection Sort | Bubble Sort | Selection Sort |
| 01 | 10 (Best) |  |  |  |  |  |  |
| 02 | 10 (Average) |  |  |  |  |  |  |
| 03 | 10 (Worst) |  |  |  |  |  |  |
| 04 | 100 (Best) |  |  |  |  |  |  |
| 05 | 100 (Average) |  |  |  |  |  |  |
| 06 | 100 (Worst) |  |  |  |  |  |  |
| 07 | 1000 (Best) |  |  |  |  |  |  |
| 08 | 1000 (Average) |  |  |  |  |  |  |
| 09 | 1000 (Worst) |  |  |  |  |  |  |

Conclusion: Write a brief conclusion stating for what kind of data and when it is suitable to use selection/bubble/insertion sort algorithms.