Problem Statement: Stepwise Execution Analysis of Sorting Algorithms I

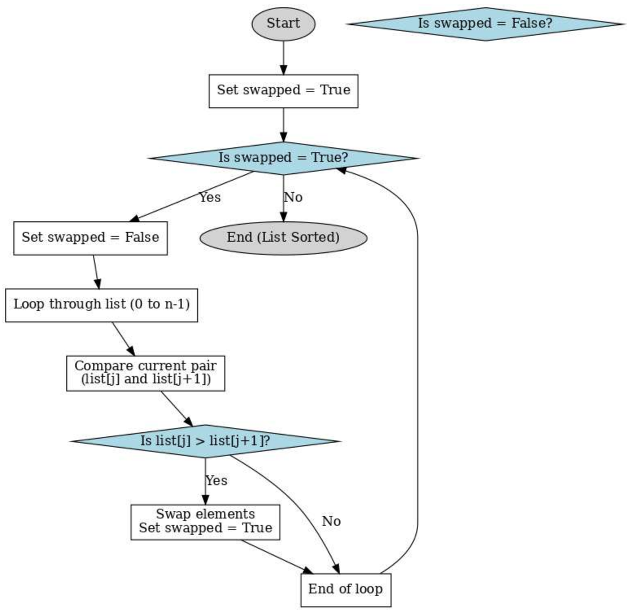
Bubble Sort

Theory:

Algorithm:

1. Look at the first two elements; if the first one is bigger, switch their positions.
2. Go to the next pair and do the same comparison and switch if needed.
3. Keep doing this until you reach the end of the list — now the biggest value has moved to the last position**.**
4. Repeat the process for the rest of the list, leaving out the part that is already sorted at the end.
5. Stop when a full round happens with no swaps**,** meaning the list is completely sorted.

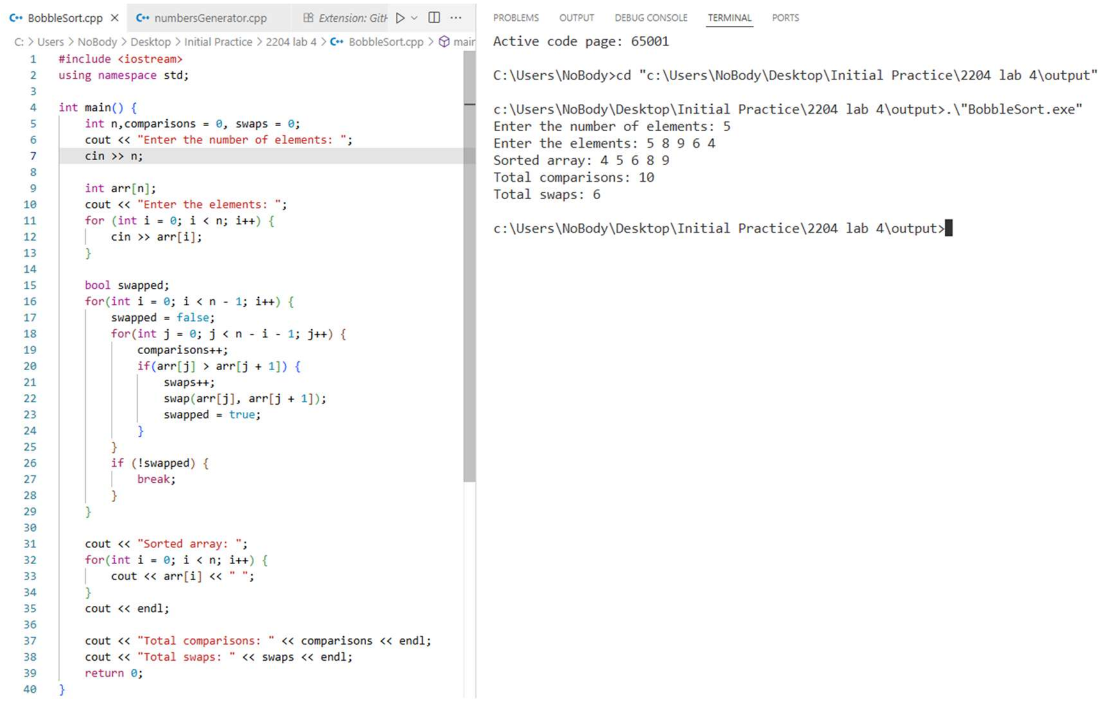
Flowchart:



Code:

```cpp
#include <iostream>
using namespace std;

int main() {
    int n,comparisons = 0, swaps = 0;
    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    bool swapped;
    for(int i = 0; i < n - 1; i++) {
        swapped = false;
        for(int j = 0; j < n - i - 1; j++) {
            comparisons++;
            if(arr[j] > arr[j + 1]) {
                swaps++;
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
        if (!swapped) {
            break;
        }
    }

    cout << "Sorted array: ";
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    cout << "Total comparisons: " << comparisons << endl;
    cout << "Total swaps: " << swaps << endl;
    return 0;
}
```

Screenshots:



```cpp
#include <iostream>
using namespace std;

int main() {
    int n,comparisons = 0, swaps = 0;
    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    bool swapped;
    for(int i = 0; i < n - 1; i++) {
        swapped = false;
        for(int j = 0; j < n - i - 1; j++) {
            comparisons++;
            if(arr[j] > arr[j + 1]) {
                swaps++;
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
        if (!swapped) {
            break;
        }
    }

    cout << "Sorted array: ";
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    cout << "Total comparisons: " << comparisons << endl;
    cout << "Total swaps: " << swaps << endl;
    return 0;
}
```

```
Active code page: 65001

C:\Users\NoBody>cd "c:\Users\NoBody\Desktop\Initial Practice\2204 lab 4\output"

c:\Users\NoBody\Desktop\Initial Practice\2204 lab 4\output>.\"BobbleSort.exe"
Enter the number of elements: 5
Enter the elements: 5 8 9 6 4
Sorted array: 4 5 6 8 9
Total comparisons: 10
Total swaps: 6

c:\Users\NoBody\Desktop\Initial Practice\2204 lab 4\output>
```

Analysis:

Input Array: A = [5, 8, 9, 6, 4]

Stepwise Execution (Pass by Pass)

Pass 1 (i = 1):

- Compare 5 and 8 → no swap → [5, 8, 9, 6, 4]
- Compare 8 and 9 → no swap → [5, 8, 9, 6, 4]
- Compare 9 and 6 → swap → [5, 8, 6, 9, 4]
- Compare 9 and 4 → swap → [5, 8, 6, 4, 9]

Pass 2 (i = 2):

- Compare 5 and 8 → no swap → [5, 8, 6, 4, 9]
- Compare 8 and 6 → swap → [5, 6, 8, 4, 9]
- Compare 8 and 4 → swap → [5, 6, 4, 8, 9]

Pass 3 (i = 3):

- Compare 5 and 6 → no swap → [5, 6, 4, 8, 9]
- Compare 6 and 4 → swap → [5, 4, 6, 8, 9]

Pass 4 (i = 4):

- Compare 5 and 4 → swap → [4, 5, 6, 8, 9]

Final Sorted Array: [4, 5, 6, 8, 9]

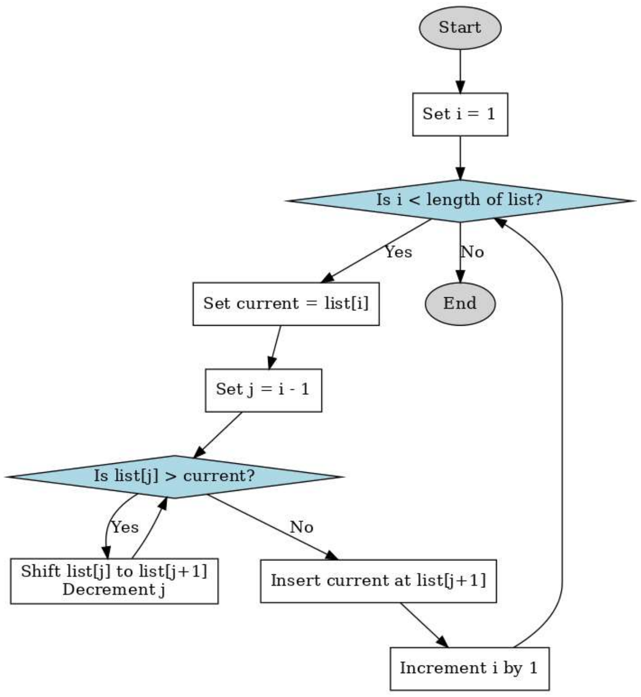
<u>Insertion Sort</u>

Theory:

Algorithm:

1. Start from the second element of the list.
2. Compare the current element with the previous elements of the sorted section.
3. Move all the elements that are greater than the current element one position to the right to make space for the current element.
4. Insert the current element into the correct position in the sorted section.
5. Repeat this process until all elements are processed.

Flowchart:

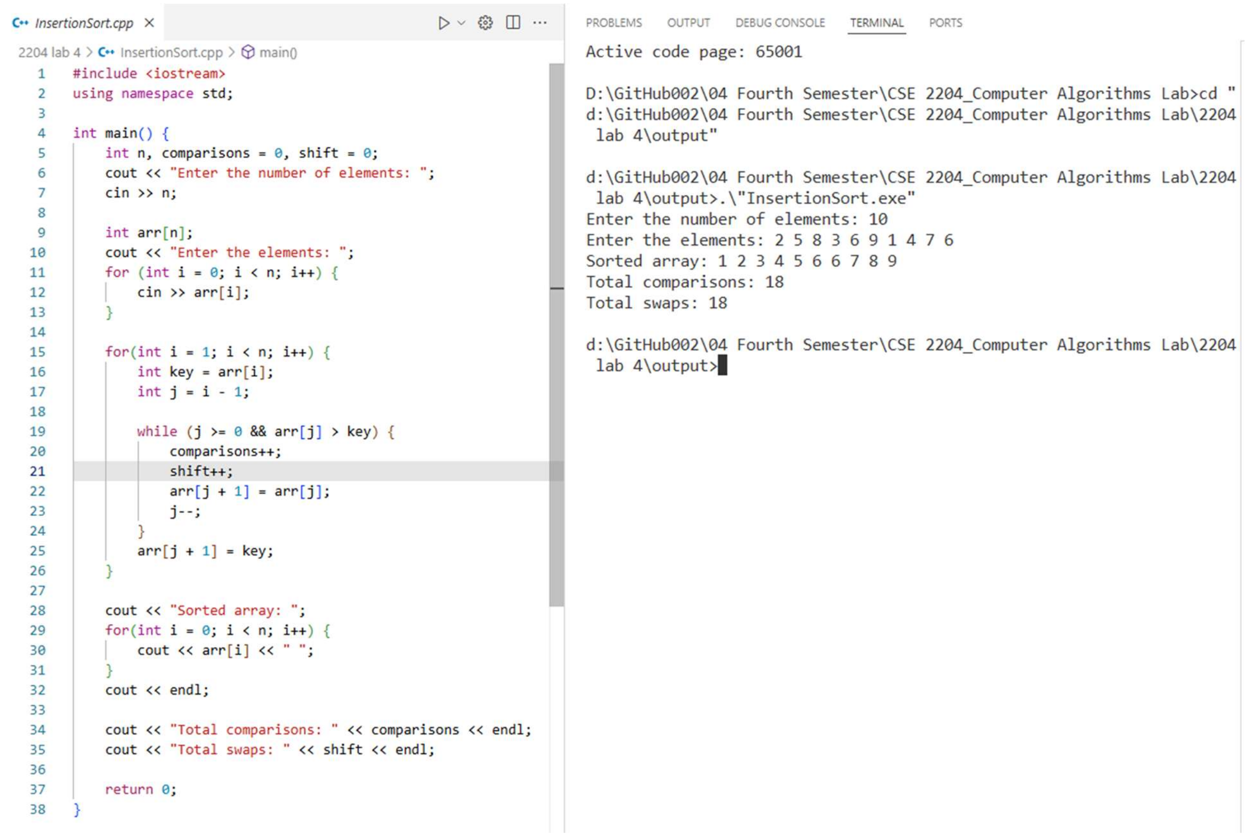


Code:

```
#include <iostream>
using namespace std;

int main() {
    int n, comparisons = 0, shift = 0;
    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++) {
```

```cpp
        cin >> arr[i];
    }

    for(int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            comparisons++;
            shift++;
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }

    cout << "Sorted array: ";
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    cout << "Total comparisons: " << comparisons << endl;
    cout << "Total swaps: " << shift << endl;

    return 0;
}
```

Screenshots:



```cpp
#include <iostream>
using namespace std;

int main() {
    int n, comparisons = 0, shift = 0;
    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    for(int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            comparisons++;
            shift++;
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }

    cout << "Sorted array: ";
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    cout << "Total comparisons: " << comparisons << endl;
    cout << "Total swaps: " << shift << endl;

    return 0;
}
```

```
Active code page: 65001

D:\GitHub002\04 Fourth Semester\CSE 2204_Computer Algorithms Lab>cd "
d:\GitHub002\04 Fourth Semester\CSE 2204_Computer Algorithms Lab\2204
 lab 4\output"

d:\GitHub002\04 Fourth Semester\CSE 2204_Computer Algorithms Lab\2204
 lab 4\output>.\"InsertionSort.exe"
Enter the number of elements: 10
Enter the elements: 2 5 8 3 6 9 1 4 7 6
Sorted array: 1 2 3 4 5 6 6 7 8 9
Total comparisons: 18
Total swaps: 18

d:\GitHub002\04 Fourth Semester\CSE 2204_Computer Algorithms Lab\2204
 lab 4\output>
```

Analysis:

Initial Array:
[5, 8, 9, 6, 4]

Pass 1 (Second element → 8)

- Compare 8 with 5 (left element).
  8 is not smaller than 5, so it stays in place.
  Array after Pass 1:
  [5, 8, 9, 6, 4]

Pass 2 (Third element → 9)

- Compare 9 with 8.
  9 is not smaller than 8, so it stays in place.
  Array after Pass 2:
  [5, 8, 9, 6, 4]

Pass 3 (Fourth element → 6)

- Compare 6 with 9 → $6 < 9$, shift 9 to the right.
- Compare 6 with 8 → $6 < 8$, shift 8 to the right.
- Compare 6 with 5 → $6 > 5$, insert 6 after 5.
  Array after Pass 3:
  [5, 6, 8, 9, 4]

Pass 4 (Fifth element → 4)

- Compare 4 with 9 → $4 < 9$, shift 9 to the right.
- Compare 4 with 8 → $4 < 8$, shift 8 to the right.
- Compare 4 with 6 → $4 < 6$, shift 6 to the right.
- Compare 4 with 5 → $4 < 5$, shift 5 to the right.
  Insert 4 at the beginning.
  Array after Pass 4:
  [4, 5, 6, 8, 9]

Final Sorted Array:
[4, 5, 6, 8, 9]

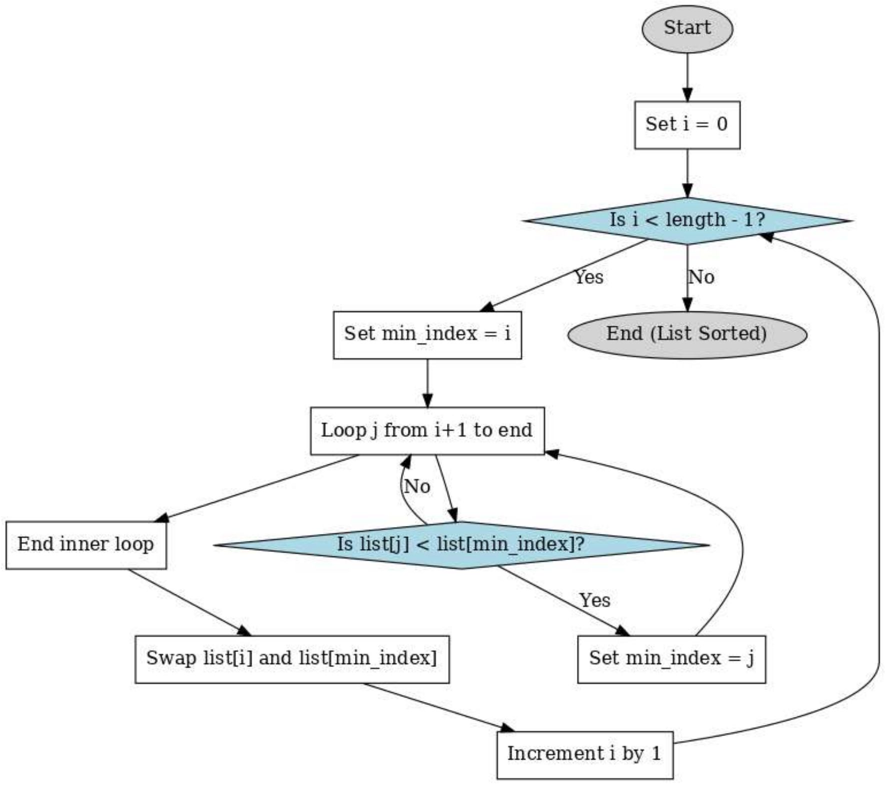
Selection Sort

Theory:

Algorithm:

1. Begin with the first element and scan the entire list to locate the smallest value.
2. Swap that smallest value with the first element.
3. Shift your focus to the second element and repeat the search within the remaining unsorted part of the list.
4. Keep repeating this process until every element is in its correct position and the list is fully sorted.

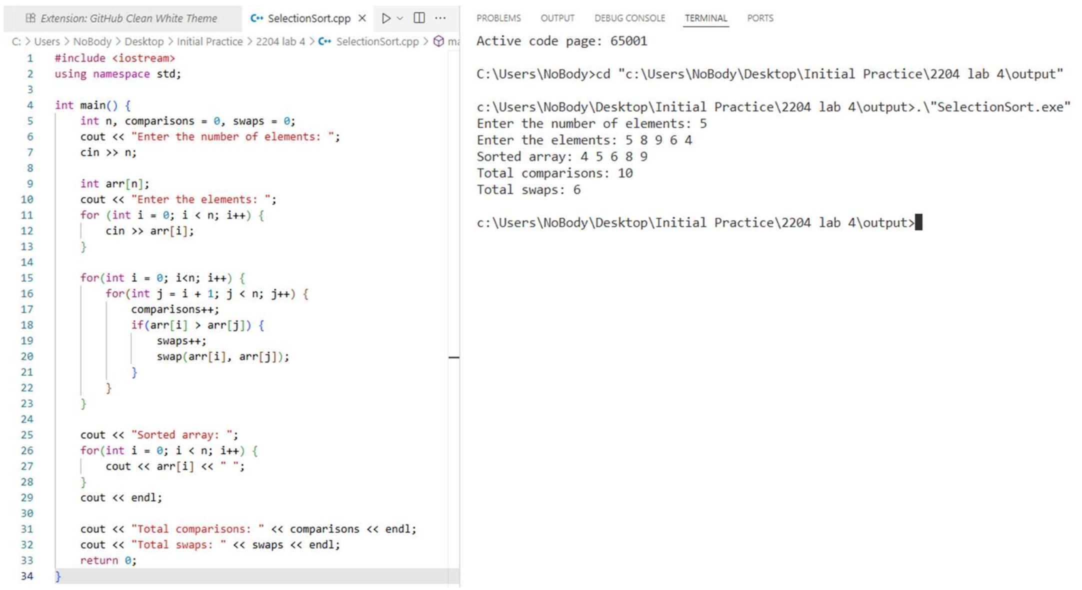
Flowchart:



Code:

```cpp
#include <iostream>
using namespace std;

int main() {
    int n, comparisons = 0, swaps = 0;
    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    for(int i = 0; i<n; i++) {
        for(int j = i + 1; j < n; j++) {
            comparisons++;
            if(arr[i] > arr[j]) {
                swaps++;
                swap(arr[i], arr[j]);
            }
        }
    }

    cout << "Sorted array: ";
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    cout << "Total comparisons: " << comparisons << endl;
    cout << "Total swaps: " << swaps << endl;
    return 0;
}
```

Screenshots:


Extension: GitHub Clean White Theme
SelectionSort.cpp
C: > Users > NoBody > Desktop > Initial Practice > 2204 lab 4 > SelectionSort.cpp
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS

```cpp
#include <iostream>
using namespace std;

int main() {
    int n, comparisons = 0, swaps = 0;
    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    for(int i = 0; i<n; i++) {
        for(int j = i + 1; j < n; j++) {
            comparisons++;
            if(arr[i] > arr[j]) {
                swaps++;
                swap(arr[i], arr[j]);
            }
        }
    }

    cout << "Sorted array: ";
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    cout << "Total comparisons: " << comparisons << endl;
    cout << "Total swaps: " << swaps << endl;
    return 0;
}
```

```
Active code page: 65001

C:\Users\NoBody>cd "c:\Users\NoBody\Desktop\Initial Practice\2204 lab 4\output"

c:\Users\NoBody\Desktop\Initial Practice\2204 lab 4\output>.\"SelectionSort.exe"
Enter the number of elements: 5
Enter the elements: 5 8 9 6 4
Sorted array: 4 5 6 8 9
Total comparisons: 10
Total swaps: 6

c:\Users\NoBody\Desktop\Initial Practice\2204 lab 4\output>
```

Analysis:

Initial Array:
[5, 8, 9, 6, 4]

Pass 1 (First position)

- Find the smallest element in the entire array → 4
- Swap 4 with the first element 5
  Array after Pass 1: [4, 8, 9, 6, 5]

Pass 2 (Second position)

- Look through the remaining array [8, 9, 6, 5]
- Smallest element is 5
- Swap 5 with 8
  Array after Pass 2: [4, 5, 9, 6, 8]

Pass 3 (Third position)

- Look through the remaining [9, 6, 8]
- Smallest element is 6
- Swap 6 with 9
  Array after Pass 3: [4, 5, 6, 9, 8]

Pass 4 (Fourth position)

- Look through the remaining [9, 8]
- Smallest element is 8
- Swap 8 with 9
  Array after Pass 4: [4, 5, 6, 8, 9]

Pass 5 (Last position)

- Only one element left, no action needed.

Final Sorted Array: [4, 5, 6, 8, 9]

<u>Observation Table</u>

Test Case No.	**Input Size (n)**	**Number of Comparisons**			**Number of Swap/Shift**			**Execution Time**		
		Bubble Sort	Insertion Sort	Selection Sort	Bubble Sort	Insertion Sort	Selection Sort	Bubble Sort	Insertion Sort	Selection Sort
01	10 (Best)	9	0	45	0	0	0	4.012 s	9.287 s	9.454 s
02	10 (Average)	44	19	45	19	19	15	6.629 s	7.971 s	7.907 s
03	10 (Worst)	45	45	45	45	45	45	2.387 s	9.258 s	7.548 s
04	100 (Best)	99	0	4950	0	0	0	2.936 s	9.159 s	7.325 s
05	100 (Average)	4814	2404	4950	2404	2404	1490	7.265 s	9.638 s	7.465 s
06	100 (Worst)	4950	4950	4950	4950	4950	4950	4.136 s	9.250 s	8.598 s
07	1000 (Best)	999	0	499500	0	0	0	5.044 s	8.964 s	6.935 s
08	1000 (Average)	498465	241331	499500	241331	241331	171500	4.063 s	6.799 s	8.547 s
09	1000 (Worst)	499500	499500	499500	499500	499500	499500	3.203 s	6.358 s	8.487 s

Conclusion:

Bubble Sort, Insertion Sort and Selection Sort are best suited for small datasets, but their efficiency varies with data conditions. **Bubble Sort** works well only for very small or nearly sorted data but becomes highly inefficient as input size grows. **Insertion Sort** is the most efficient among the three for **small or nearly sorted datasets**, offering faster execution with fewer comparisons and shifts. **Selection Sort**, though predictable with a fixed number of comparisons, is slower on nearly sorted data and less efficient for large inputs. For practical purposes, **Insertion Sort** is generally the preferred choice for small datasets, while Bubble and Selection Sort are mostly useful for educational demonstrations or very limited data sizes.