ÇANKAYA UNIVERSITY

SOFTWARE ENGINEERING DEPARTMENT

Name Surname	**Duru Karacan**
Identity Number	**202128022**
Course	**SENG 201**
Experiment	**Programming Assignment 2 - part 2**
E-mail	**c2128022@student.cankaya.edu.tr**

QuickSort:

Best Case: **O(n log n)**: the pivot is always in the middle or close to the middle of the array.
Worst Case: **O(n^2)**: the pivot is always the smallest or largest element in the array.

Best Case Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Worst Case Array: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

MergeSort:

Best Case & Worst Case: **O(n log n)**.

Best Case & Worst Case Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Insertion Sort:

Best Case :**O(n)**: the array is already sorted.
Worst Case Time Complexity: **O(n^2)**: the array is sorted in reverse order.

Best Case Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Worst Case Array: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Bubble Sort:

Best Case: **O(n)**: the array is already sorted.
Worst Case: **O(n^2)**: the array is sorted in reverse order.

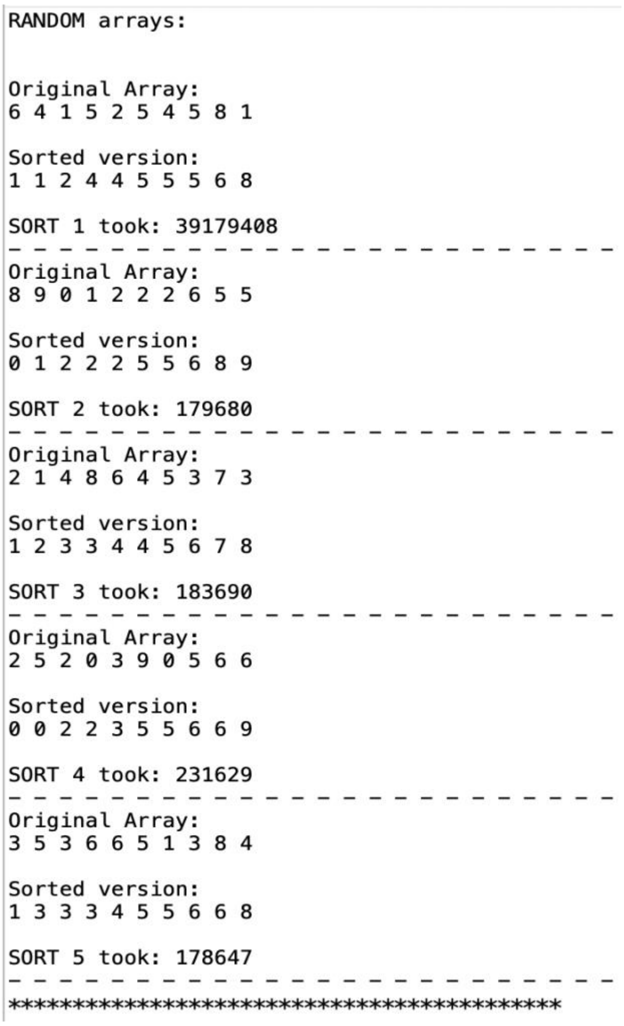
Best Case Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Worst Case Array: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Selection Sort:

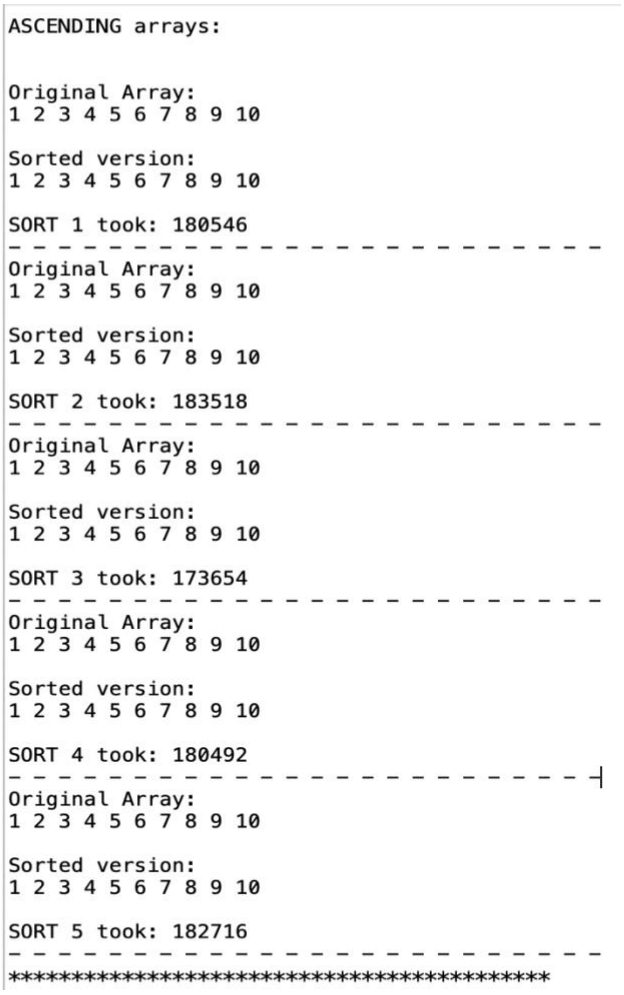
Best Case & Worst Case: **O(n^2):** every case because the algorithm makes the same number of comparisons in every case.

Best Case &Worst Case Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

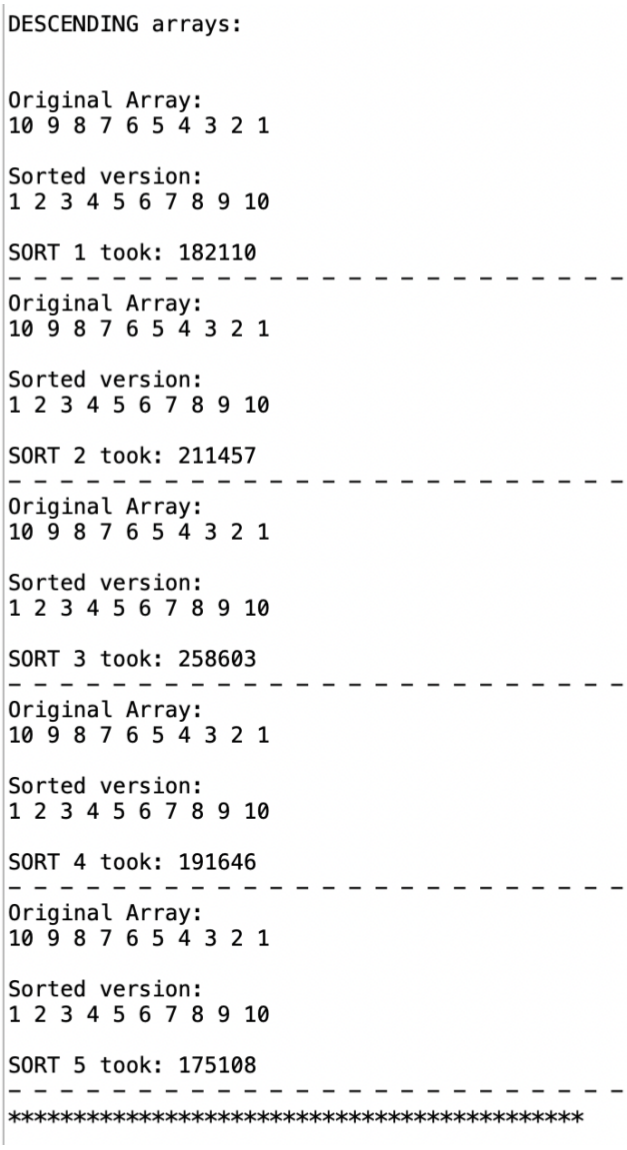
SAMPLE OUTPUT:



```
RANDOM arrays:


Original Array:
6 4 1 5 2 5 4 5 8 1

Sorted version:
1 1 2 4 4 5 5 5 6 8

SORT 1 took: 39179408
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
8 9 0 1 2 2 2 6 5 5

Sorted version:
0 1 2 2 2 5 5 6 8 9

SORT 2 took: 179680
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
2 1 4 8 6 4 5 3 7 3

Sorted version:
1 2 3 3 4 4 5 6 7 8

SORT 3 took: 183690
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
2 5 2 0 3 9 0 5 6 6

Sorted version:
0 0 2 2 3 5 5 6 6 9

SORT 4 took: 231629
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
3 5 3 6 6 5 1 3 8 4

Sorted version:
1 3 3 3 4 5 5 6 6 8

SORT 5 took: 178647
- - - - - - - - - - - - - - - - - - - - - - - -
*******************************************
```

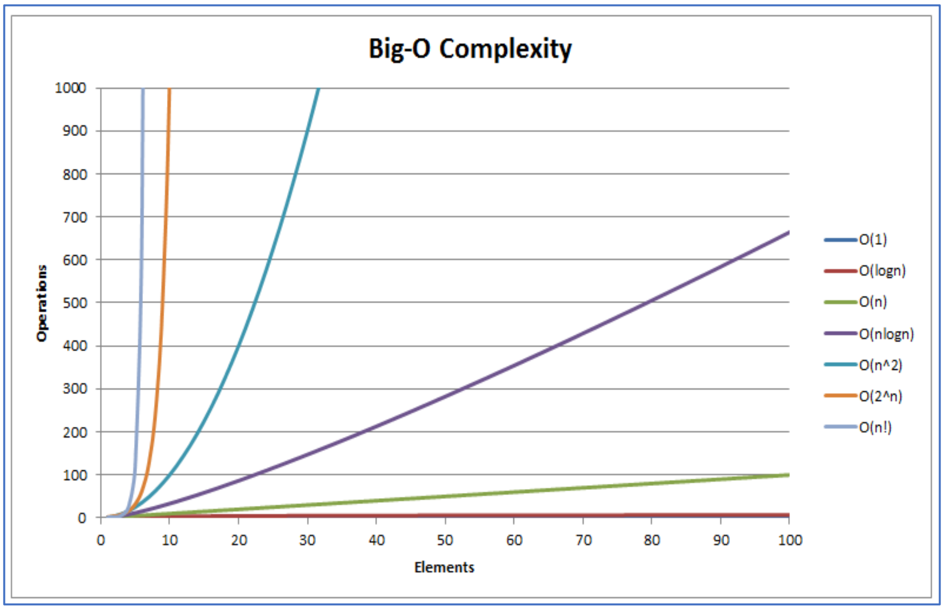


```
ASCENDING arrays:


Original Array:
1 2 3 4 5 6 7 8 9 10

Sorted version:
1 2 3 4 5 6 7 8 9 10

SORT 1 took: 180546
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
1 2 3 4 5 6 7 8 9 10

Sorted version:
1 2 3 4 5 6 7 8 9 10

SORT 2 took: 183518
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
1 2 3 4 5 6 7 8 9 10

Sorted version:
1 2 3 4 5 6 7 8 9 10

SORT 3 took: 173654
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
1 2 3 4 5 6 7 8 9 10

Sorted version:
1 2 3 4 5 6 7 8 9 10

SORT 4 took: 180492
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
1 2 3 4 5 6 7 8 9 10

Sorted version:
1 2 3 4 5 6 7 8 9 10

SORT 5 took: 182716
- - - - - - - - - - - - - - - - - - - - - - - -
*******************************************
```



```
DESCENDING arrays:


Original Array:
10 9 8 7 6 5 4 3 2 1

Sorted version:
1 2 3 4 5 6 7 8 9 10

SORT 1 took: 182110
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
10 9 8 7 6 5 4 3 2 1

Sorted version:
1 2 3 4 5 6 7 8 9 10

SORT 2 took: 211457
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
10 9 8 7 6 5 4 3 2 1

Sorted version:
1 2 3 4 5 6 7 8 9 10

SORT 3 took: 258603
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
10 9 8 7 6 5 4 3 2 1

Sorted version:
1 2 3 4 5 6 7 8 9 10

SORT 4 took: 191646
- - - - - - - - - - - - - - - - - - - - - - - -
Original Array:
10 9 8 7 6 5 4 3 2 1

Sorted version:
1 2 3 4 5 6 7 8 9 10

SORT 5 took: 175108
- - - - - - - - - - - - - - - - - - - - - - - -
*********************************************
```

- I will create an array for each sort in a random, ascending and descending order, follow their times and comment accordingly.
- My expectation for **quicksort** is that it will **only take longer to resolve the descending order**.
- My expectation for **merge sort** is that it will **resolve all orders very soon**.
- My expectation for **insertion sort** is that it **only takes longer to resolve the descending order**.
- My expectation for **bubble sort** is that it **only takes longer to solve the descending order**.
- My expectation for **selection sort** is that it will **resolve all orders very soon**.
- **In ascending order**, my expectation is that the **insertion sort will be the fastest** and **the 2 values closest to it will be merge and quick.**
- **In descending order,** my expectation is that **merge sort will run the fastest.**



Sort of **sorting algorithms from fastest to slowest**:

fastest

O(1)

O(logN)

O(N)

O(NlogN)

O(N^2)

O(2^N)

O(N!)

slowest

	RANDOM order	**ASCENDING order**	**DESCENDING order**
SORT 1	39179408 nsec	180546 nsec	182110 nsec
SORT 2	179680 nsec	183518 nsec	211457 nsec
SORT 3	183690 nsec	173654 nsec	258603 nsec
SORT 4	231629 nsec	180492 nsec	191646 nsec
SORT 5	178647 nsec	182716 nsec	175108 nsec

MY PREDICTIONS

SORT 1: Quicksort

Random Order: 39179408 nsec
Ascending Order: 180546 nsec
Descending Order: 182110 nsec
Quicksort aligns with my expectation of taking longer in descending order.

SORT 2: Merge Sort

Random Order: 179680 nsec
Ascending Order: 183518 nsec
Descending Order: 211457 nsec
Merge sort did not meet my expectation of being the fastest in all orders, but it's likely to be Merge Sort due to its efficiency.

SORT 3: Insertion Sort

Random Order: 183690 nsec
Ascending Order: 73654 nsec
Descending Order: 258603 nsec
Insertion sort aligns with my expectation of taking longer in descending order.

SORT 4: Bubble Sort

Random Order: 231629 nsec
Ascending Order: 180492 nsec
Descending Order: 191646 nsec
Bubble sort aligns with my expectation of taking longer in descending order.

SORT 5: Selection Sort

Random Order: 178647 nsec
Ascending Order: 182716 nsec
Descending Order: 175108 nsec
Selection sort aligns with your expectation of performing well in all orders.