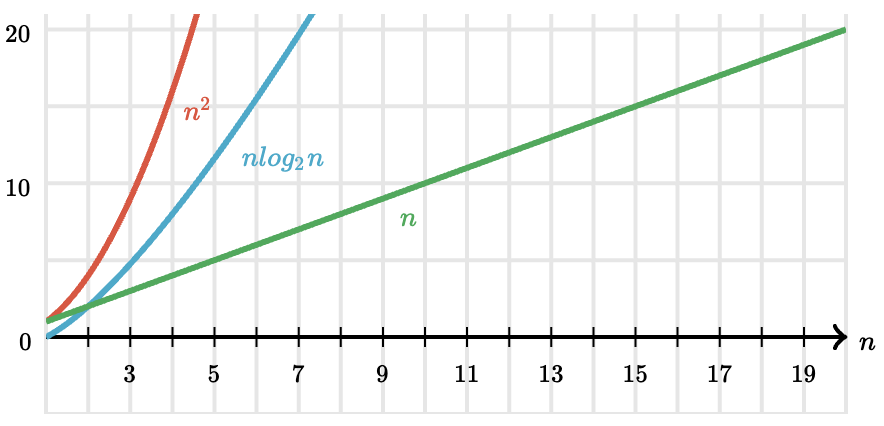
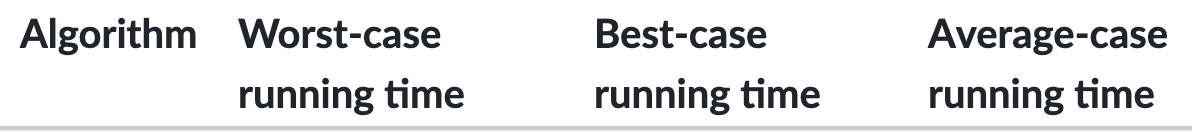
Selection sort algorithm uses brute brute force to perform its means therefore we will analyse its properties and compare them to the divide and conquer algorithm studied in class.

We will get into further details by deriving the time complexity of this one and therefore reaffirm our assumption of what learned in class.

We already know from our class notes that the algorithm (merge sort)s’ time complexity has the following properties:



Selection sort

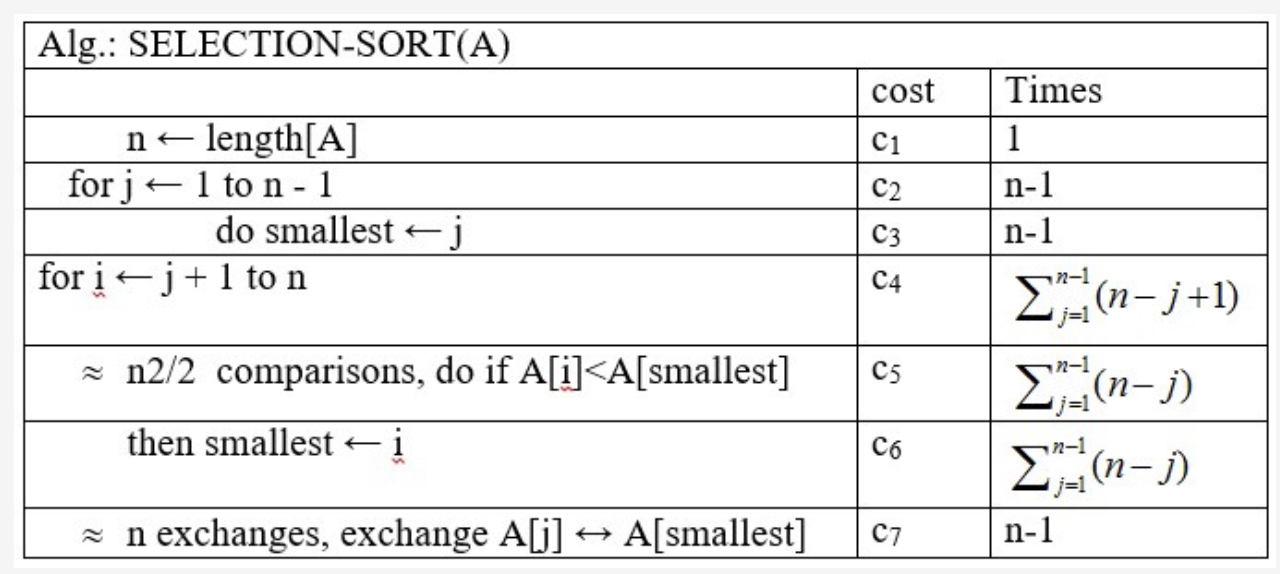
**Time Complexity:** O(n2) as there are two nested loops.

**Auxiliary-Space:** O(1)  
The good thing about selection sort is it never makes more than O(n) swaps and can be useful when memory write is a costly operation.

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.  
2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.



Derivations:

**Inside the selectionSort function:**

Line 1: **COST**= C1, **TIME**= 1, where C1 is some constant  
Line 2: **COST**=C2, **TIME**= n+1, where C2 is some constant  
Line 3: **COST**= C3, **TIME**= n, where C3 is some constant  
Line 4: **COST**= C4, **TIME**= (n²-n) / 2 + n, where C4 is some constant  
Line 5: **COST**= C5, **TIME**= (n²-n) / 2, where C5 is some constant  
Line 6: **COST**= C6, **TIME**= (n²-n) / 2, where C6 is some constant  
Line 7: **COST**= C7, **TIME**= n, where C7 is some constant  
Line 8: **COST**= C8, **TIME**= n, where C5 is some constant  
Line 9: **COST**= C9, **TIME**= n, where C9 is some constant

Now that we have all of the **costs**and the **times**, we must sum up all of the costs times the time to get the runtime:

**Runtime**= (C1 \*1) + (C2 \*(n+1)) + (C3 \*n) + (C4 \* ((n²-n)/2) + n) + (C5 \* (n²-n) / 2) + (C6 \* (n²-n) / 2) + (C7 \* n)+ (C8 \* n)+ (C9 \* n)

Where U,V, and W are constants  
= U +Vn + Wn²  
= **O(n²) //**  Just like it was derived and predicted in class.

**Note:**  the running time of divide and conquer algorithms is naturally characterized by recurrences and solving the recurrences gives us the asymptotic running time.

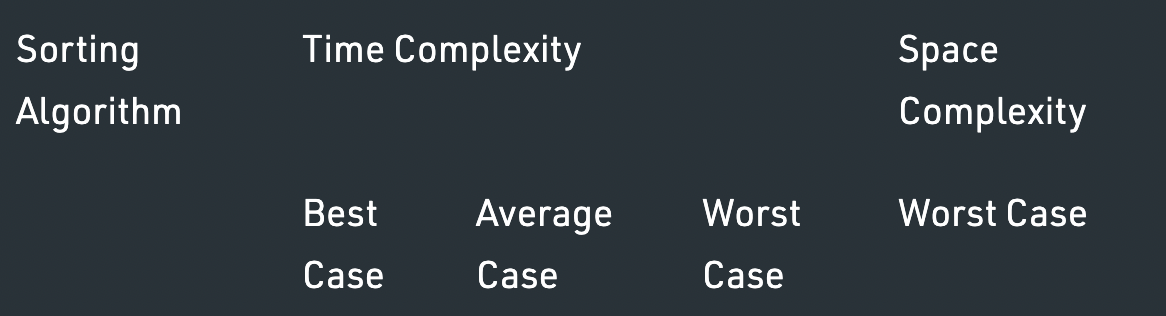
The three methods introduced in the class are :

a)- Substitution method

b)-Recursion method

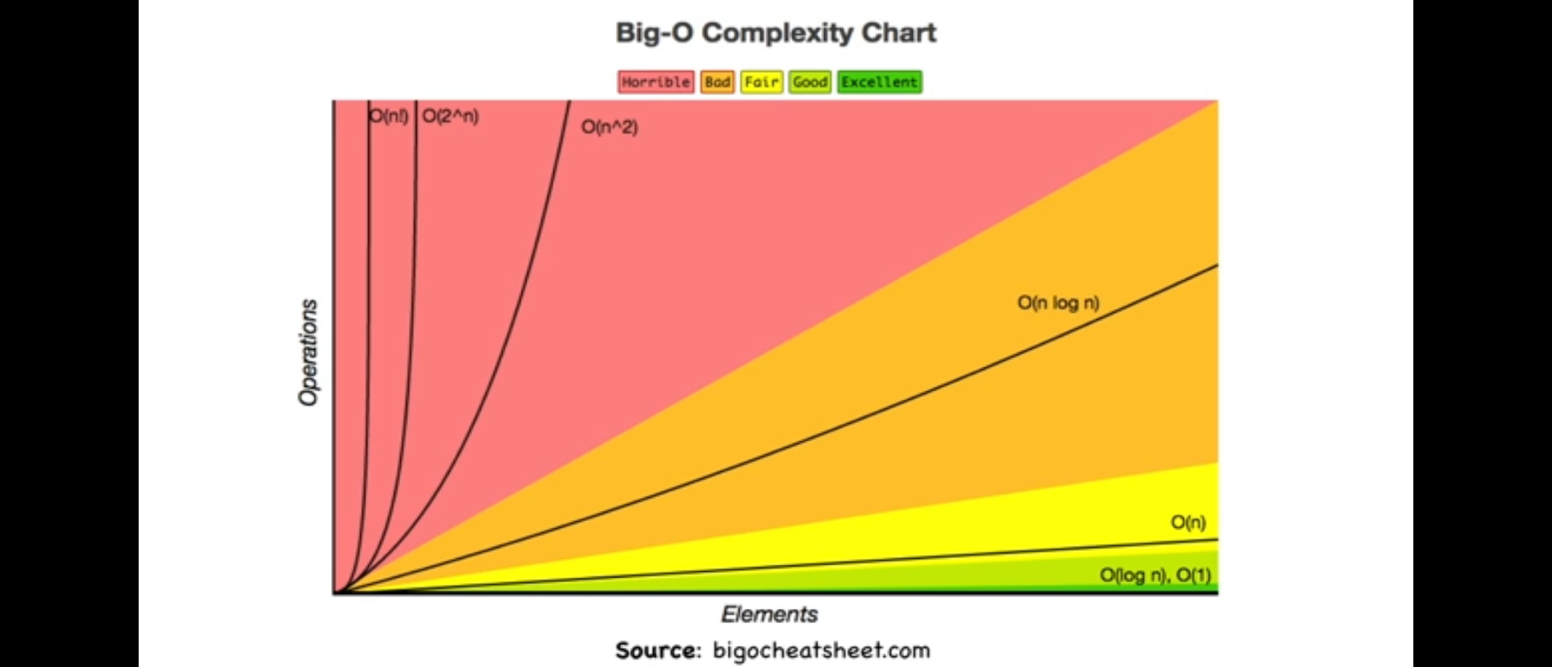
c)-Master method for solving recurrences.

Here we summarize the different algorithms and their respective big O notation.



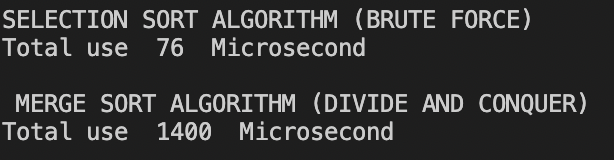
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Selection Sort** | **Ω(N2)** | **Θ(N2)** | **O(N2)** | **O(1)** |
| **Insertion Sort** | **Ω(N)** | **Θ(N2)** | **O(N2)** | **O(1)** |
| **Merge Sort** | **Ω(N log N)** | **Θ(N log N)** | **O(N log N)** | **O(N)** |

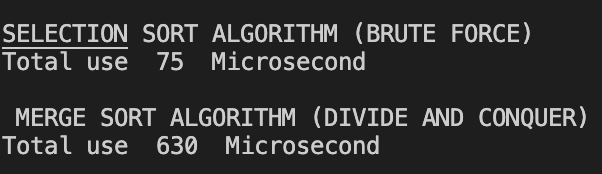
This is further explained by the help of this graph :



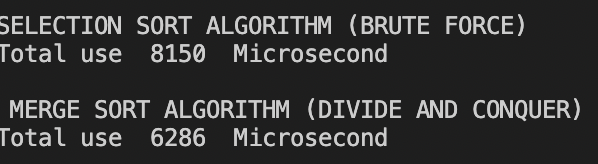
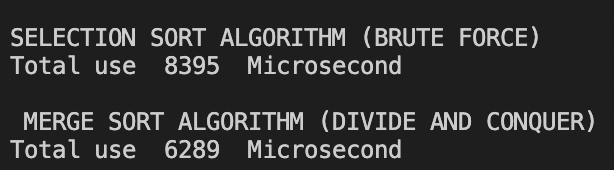
Those are some screenshot of the results we got after performing different simulation of different array sizes for the different sorting algorithms in our program console.

With 100 elements(x 2):

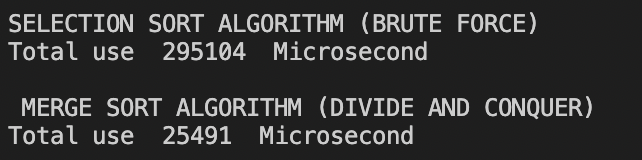
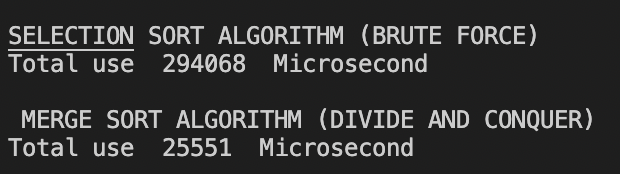




With 1000 elements(x 2):



With 10 000 elements(x 2):



Sources:

Class notes

<https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques/> (chart)

<https://www.bigocheatsheet.com/> (Chart)