Name: Junth Basnet
Roll.No: 05
Group: CE
Level: 3rd year/ 2nd Sem

COMP 314: Algorithms and Complexity
Lab work 2: Sorting

Github Link:
https://github.com/Junth19/COMP-314-Algorithms-and-Complexity-Lab/tree/master/CE__III__05__Lab2

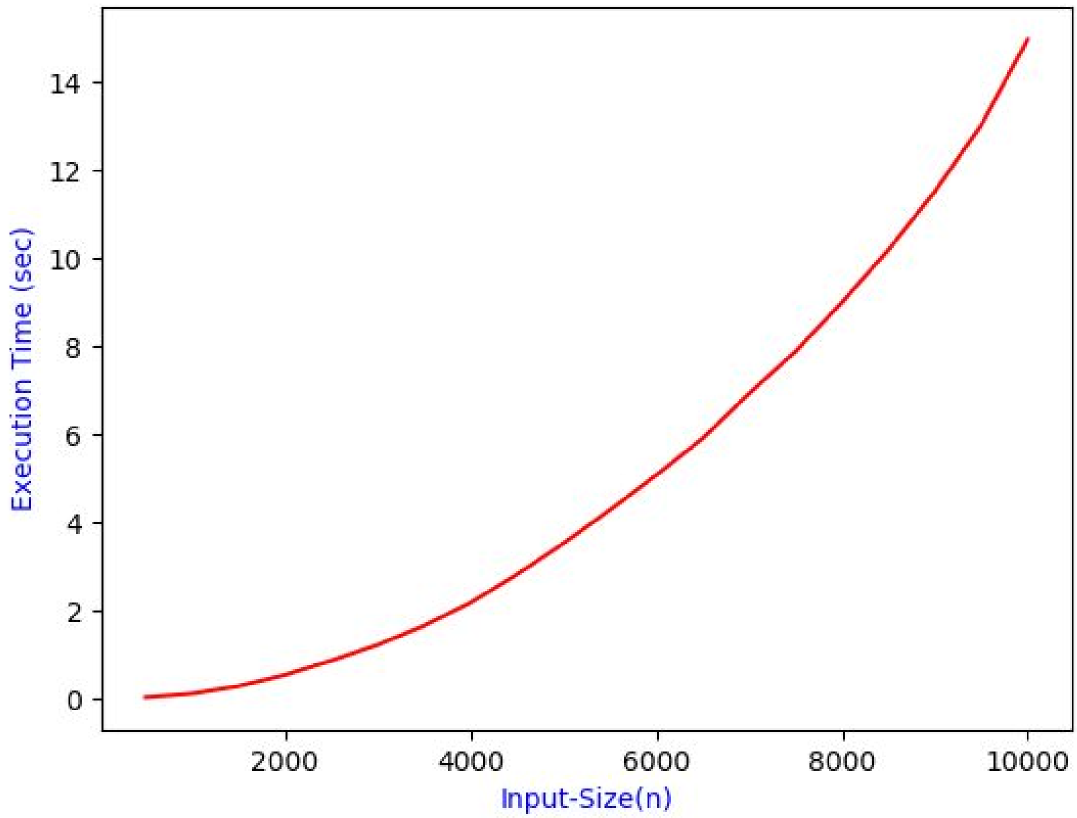
Insertion Sort

The insertion sort algorithm iterates through an input array and removes one element per iteration, finds the place the element belongs in the array, and then places it there. This process grows a sorted list from left to right.

- Worst Case: $O(n^2)$

Input Size(n)	Execution Time(sec)
500	0.04690384864807129
1000	0.1315937042236328
1500	0.3013467788696289
2000	0.5533123016357422
2500	0.8784968852996826
3000	1.2374794483184814
3500	1.6719088554382324
4000	2.1939640045166016
4500	2.8319811820983887
5000	3.53788685798645
5500	4.29442286491394
6000	5.085555791854858
6500	5.9187257289886475
7000	6.938089370727539
7500	7.893252849578857
8000	9.006293535232544
8500	10.196868896484375
9000	11.520684957504272
9500	13.025882720947266

Graph: $O(n^2)$

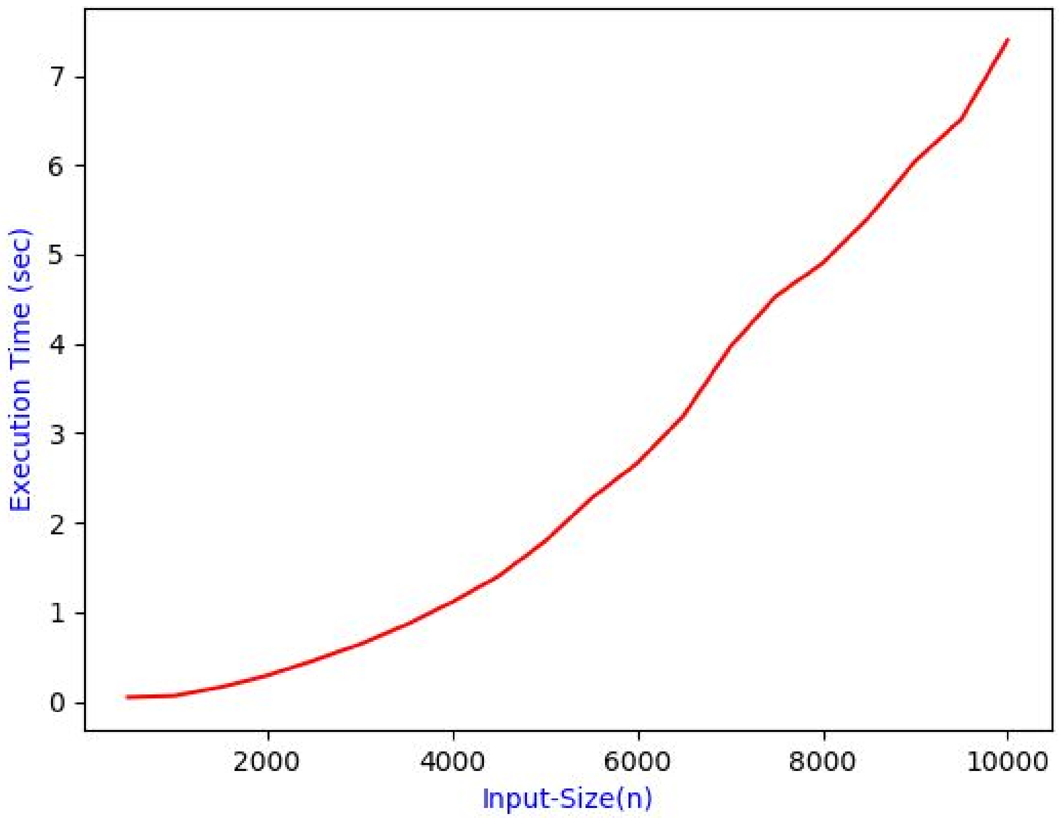


The simplest worst-case input is an array sorted in reverse order. The set of all worst-case inputs consists of all arrays where each element is the smallest or second-smallest of the elements before it. In these cases, every iteration of the inner loop will scan and shift the entire sorted subsection of the array before inserting the next element. This gives insertion sort a quadratic running time (i.e., $O(n^2)$).

❖ Average Case: $O(n^2)$

Input Size(n)	Execution Time(sec)
500	0.04681134223937988
1000	0.06245088577270508
1500	0.16046404838562012
2000	0.2882354259490967
2500	0.4565401077270508
3000	0.6378993988037109
3500	0.8562748432159424
4000	1.1122260093688965
4500	1.3997390270233154
5000	1.7873189449310303
5500	2.2673544883728027
6000	2.664799213409424
6500	3.196070432662964
7000	3.9591143131256104
7500	4.534657716751099
8000	4.899816513061523
8500	5.41332221031189
9000	6.0402185916900635
9500	6.512476205825806

Graph: $O(n^2)$

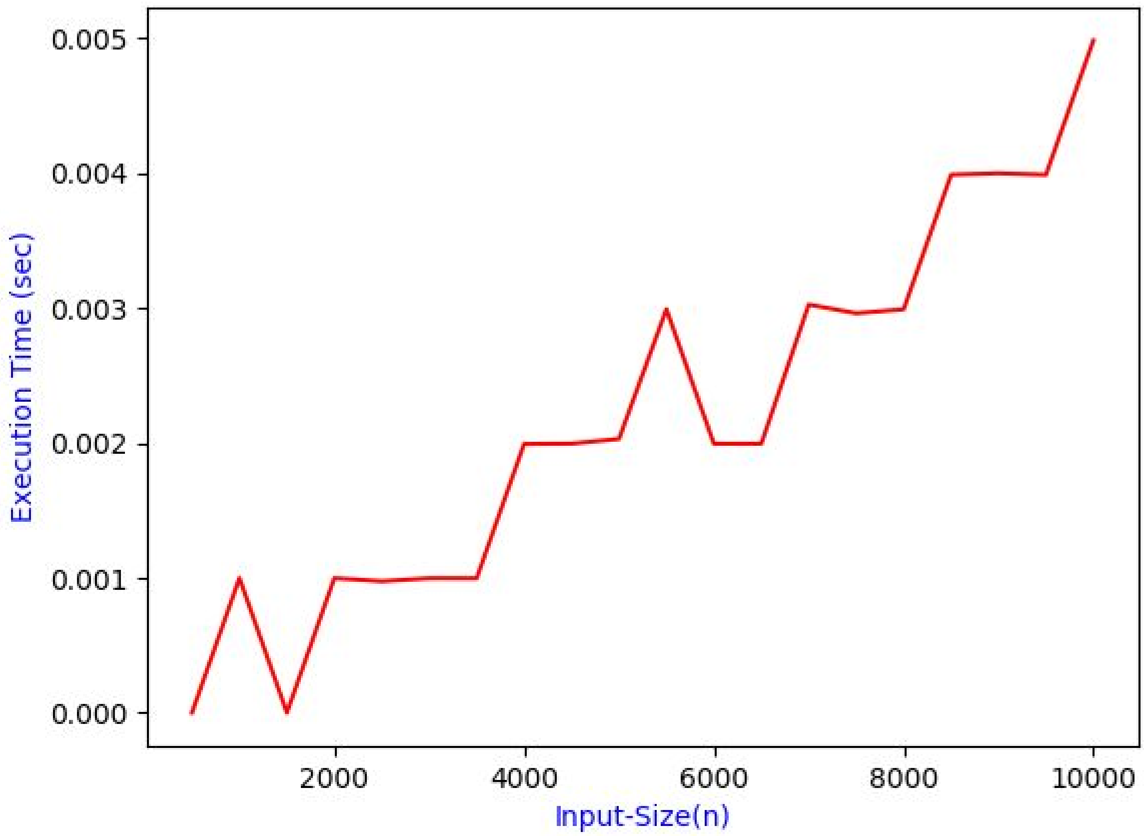


The average case is also quadratic, which makes insertion sort impractical for sorting large arrays.

- Best Case : O(n)

Input Size(n)	Execution Time(sec)
500	0.0
1000	0.0009982585906982422
1500	0.0
2000	0.000997781753540039
2500	0.0009748935699462891
3000	0.0009970664978027344
3500	0.0009970664978027344
4000	0.0019927024841308594
4500	0.0019948482513427734
5000	0.002029895782470703
5500	0.002991914749145508
6000	0.001993894577026367
6500	0.001994371414184570З
7000	0.0030260086059570312
7500	0.002961397171020508
8000	0.0029909610748291016
8500	0.003988742828369141
9000	0.003999471664428711
9500	0.003988504409790039

Graph: O(n)



The best case input is an array that is already sorted. In this case insertion sort has a linear running time (i.e., O(n)). During each iteration, the first remaining element of the input is only compared with the right-most element of the sorted subsection of the array.

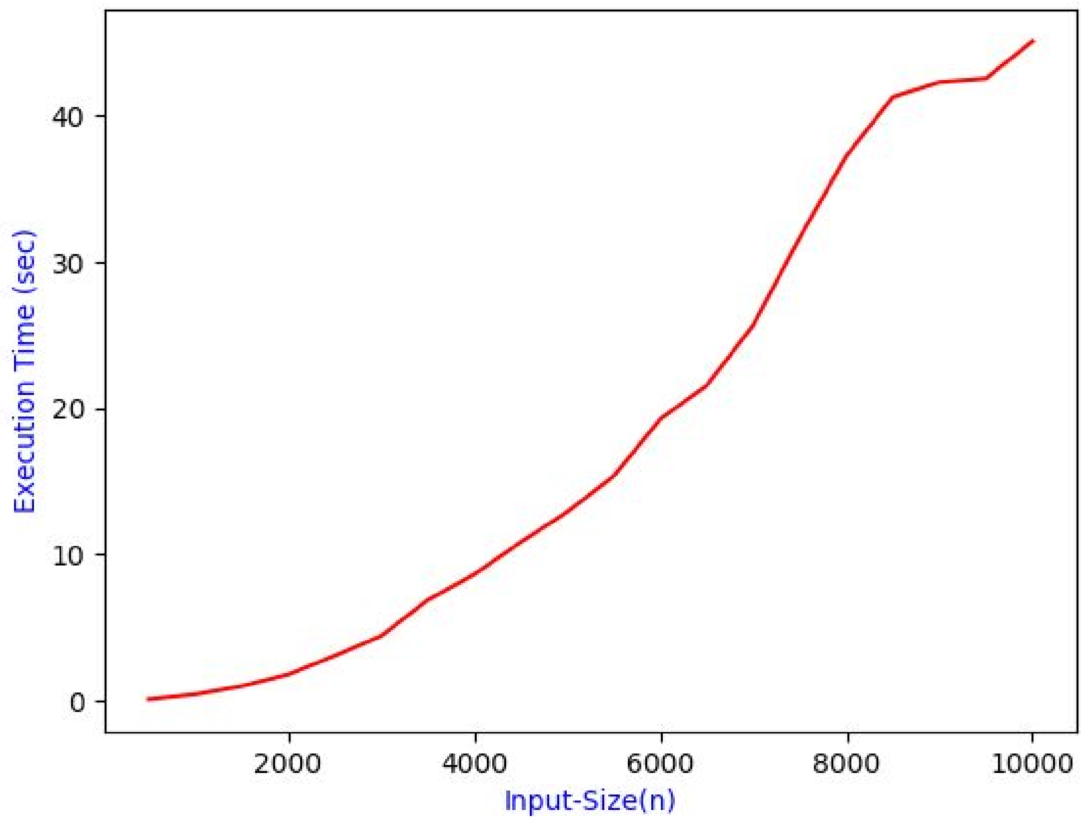
Merge Sort

Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves and then merges the two sorted halves.

- Worst Case: O(nlogn)

Input Size(n)	Execution Time(sec)
500	0.10771346092224121
1000	0.4497954845428467
1500	1.0042991638183594
2000	1.794447422027588
2500	3.0780818462371826
3000	4.4286789894104
3500	6.883840084075928
4000	8.637100458145142
4500	10.845529794692993
5000	12.909912824630737
5500	15.355231046676636
6000	19.261536359786987
6500	21.552732467651367
7000	25.685041427612305
7500	31.6725409030914
8000	37.24225950241089
8500	41.27059197425842
9000	42.287206411361694
9500	42.54072284698486

Graph: O(nlogn)

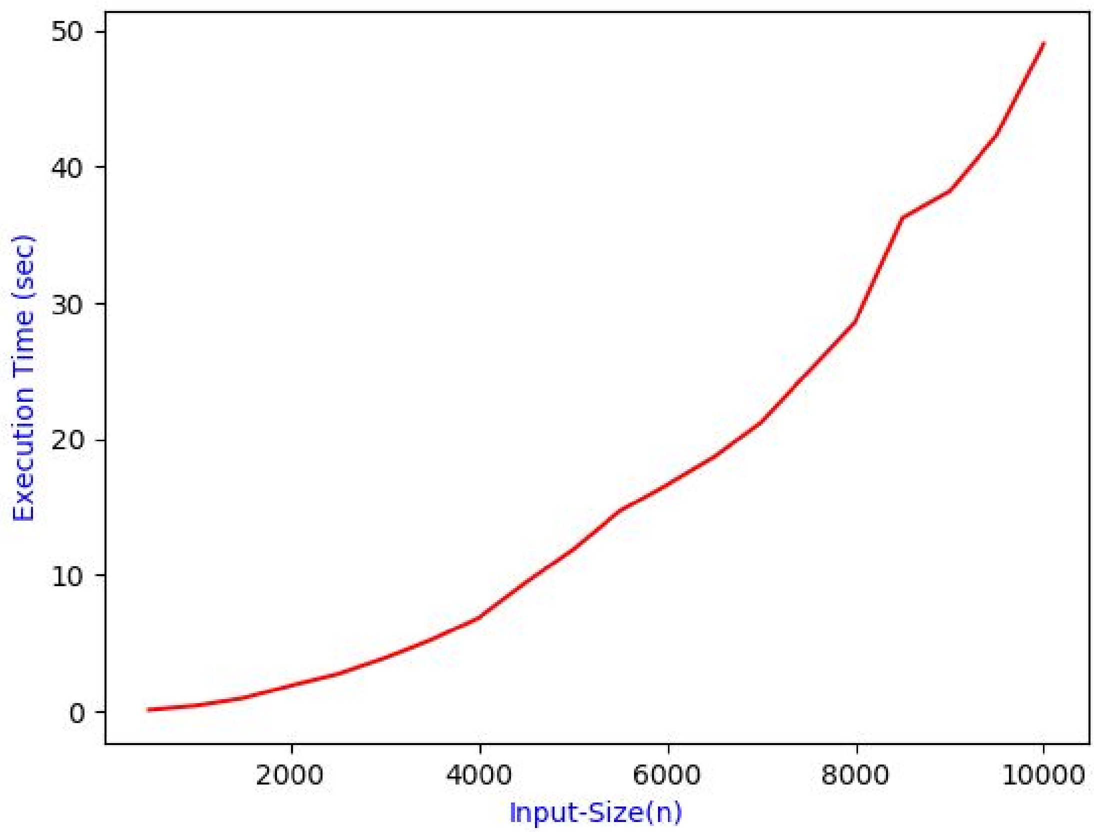


In sorting *n* objects, merge sort has an average and worst-case performance of O($n \log n$). If the running time of merge sort for a list of length *n* is $T(n)$, then the recurrence $T(n) = 2T(n/2) + n$ follows from the definition of the algorithm.

❖ Average Case: O(nlogn)

Input Size(n)	Execution Time(sec)
500	0.10932183265686035
1000	0.42177653312683105
1500	0.9685547351837158
2000	1.8891375064849854
2500	2.7453551292419434
3000	3.9080986976623535
3500	5.260749340057373
4000	6.873384475708008
4500	9.459927558898926
5000	11.863524913787842
5500	14.70813512802124
6000	16.6305570602417
6500	18.677709102630615
7000	21.21852684020996
7500	24.914501190185547
8000	28.60091233253479
8500	36.22527456283569
9000	38.156410932540894
9500	42.286498069763184

Graph: O(nlogn)



- Best Case: O(nlogn)

Input Size(n)	Execution Time(sec)
500	0.0997316837310791
1000	0.4200928211212158
1500	1.0856542587280273
2000	2.2834055423736572
2500	5.587977409362793
3000	5.988343000411987
3500	5.91169548034668
4000	7.348695516586304
4500	9.177350759506226
5000	11.173984050750732
5500	13.889242887496948
6000	18.092814922332764
6500	20.00065588951111
7000	23.773826599121094
7500	27.113330841064453
8000	28.393465995788574
8500	32.80443024635315
9000	37.42474818229675
9500	44.60211253166199

Graph: O(nlogn)