Bilgehan Geçici 150117072 – Anıl Şenay 150117023

Report

Insertion Sort

Insertion sort is based on the idea that one element from the input elements is consumed in each iteration to find its correct position i.e. the position to which it belongs in a sorted array.

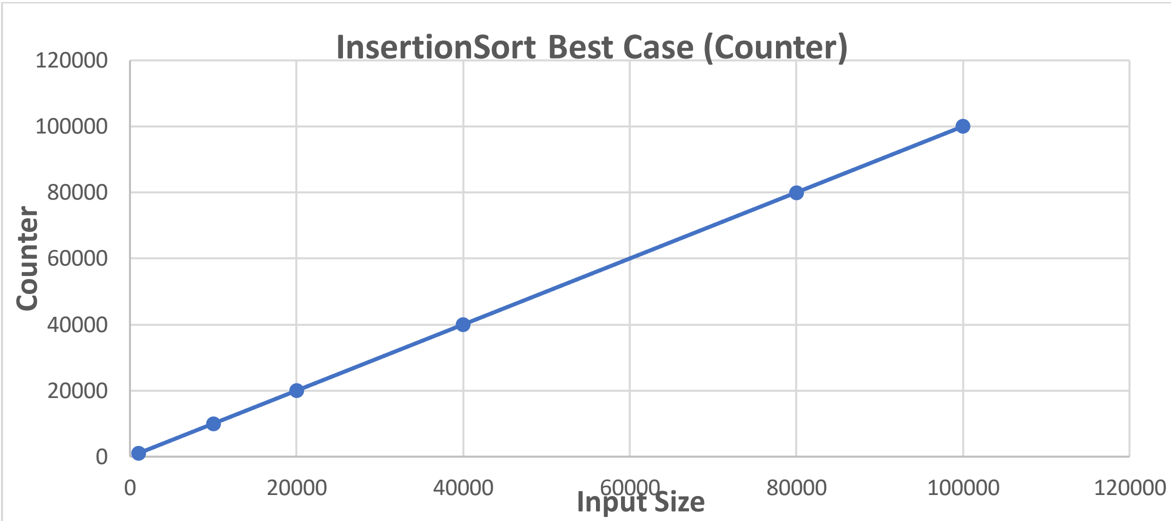
For Insertion Sort we prepare 6 best, average and worst input cases with different kind of inputs and different size (Range: 1000-100000) integers file to measure the efficiency of insertion sort.

Best Case Input

We thought that for the best case, the inputs should be ordered from large to small or small to large or be already sorted. And the size of inputs should be as small as possible.

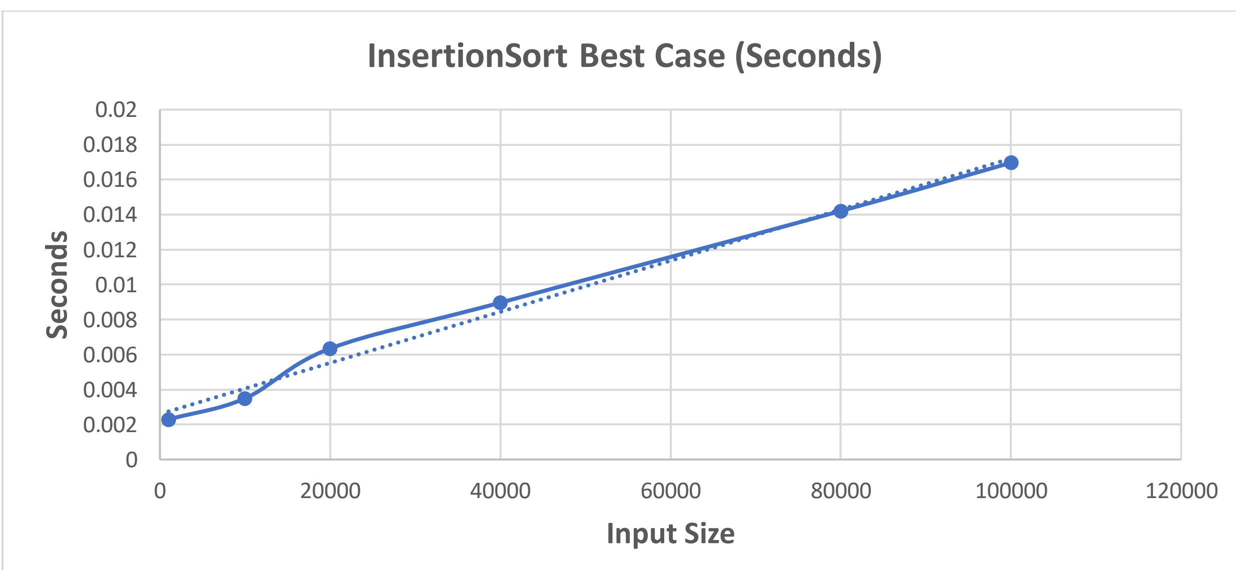
Input size	Insertion Sort Best Case (Counter)
1000	999
10000	9999
20000	19999
40000	39999
80000	79999
100000	99999

By the empirical results time complexity are approximately n for n input. So, we can say O(n) for empirical results and O(n) for theoretical results. And our findings meet theoretical expectations.



(Figure1.1 Insertion Sort Best Case (Counter))

Input size	Insertion Sort Best Case (Seconds)
1000	0.0022928
10000	0.003492
20000	0.0063419
40000	0.008966
80000	0.0142048
100000	0.0169631



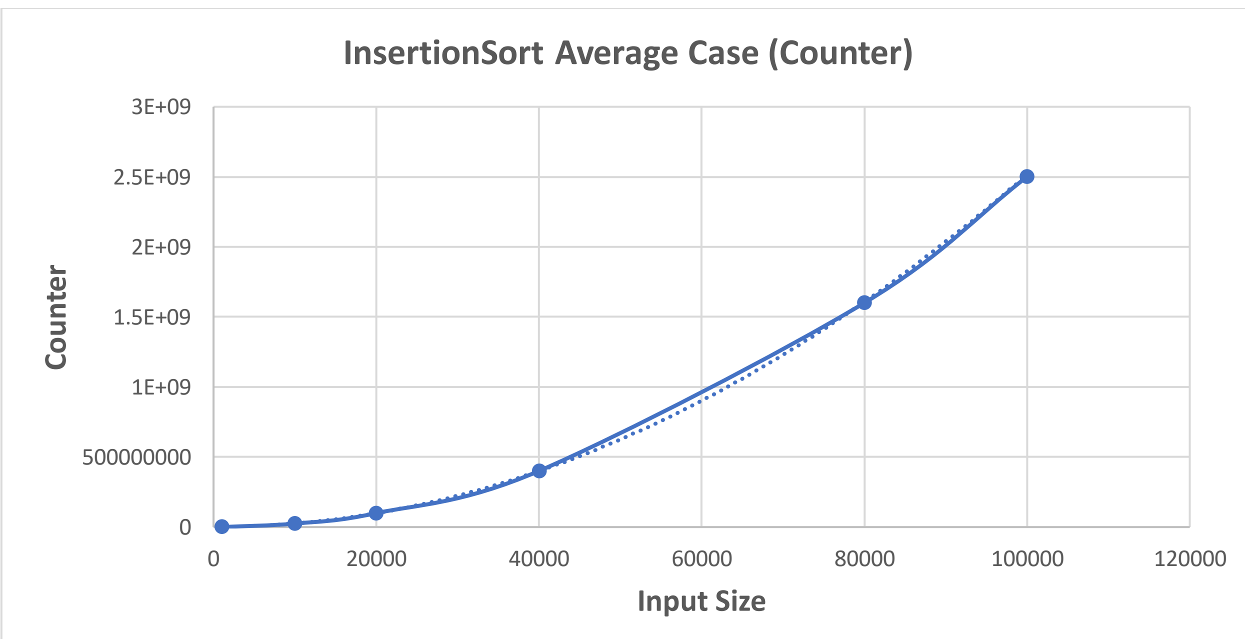
(Figure 1.2 Insertion Sort Best Case (Seconds))

Average Case Input

We thought that for the average case, the inputs should be ordered randomly to get the average results. Also, the size of inputs should be as small as possible.

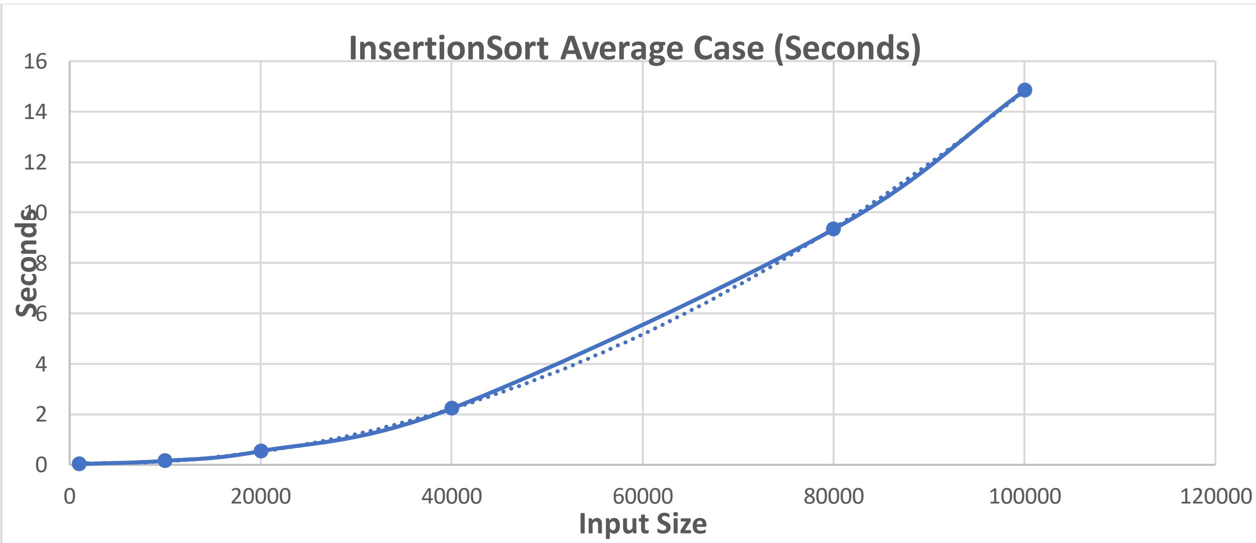
Input size	Insertion Sort Average Case (Counter)
1000	241115
10000	25175543
20000	99551173
40000	400457968
80000	1600237997
100000	2501151014

By the empirical results time complexity are approximately n^2/4 for n input. So, we can say O(n^2) for empirical results and O(n^2) for theoretical results. And our findings meet theoretical expectations.



(Figure 1.3 Insertion Sort Average Case (Counter))

Input size	Insertion Sort Average Case (Seconds)
1000	0.0306733
10000	0.1572863
20000	0.5426019
40000	2.2313613
80000	9.3342903
100000	14.8359811



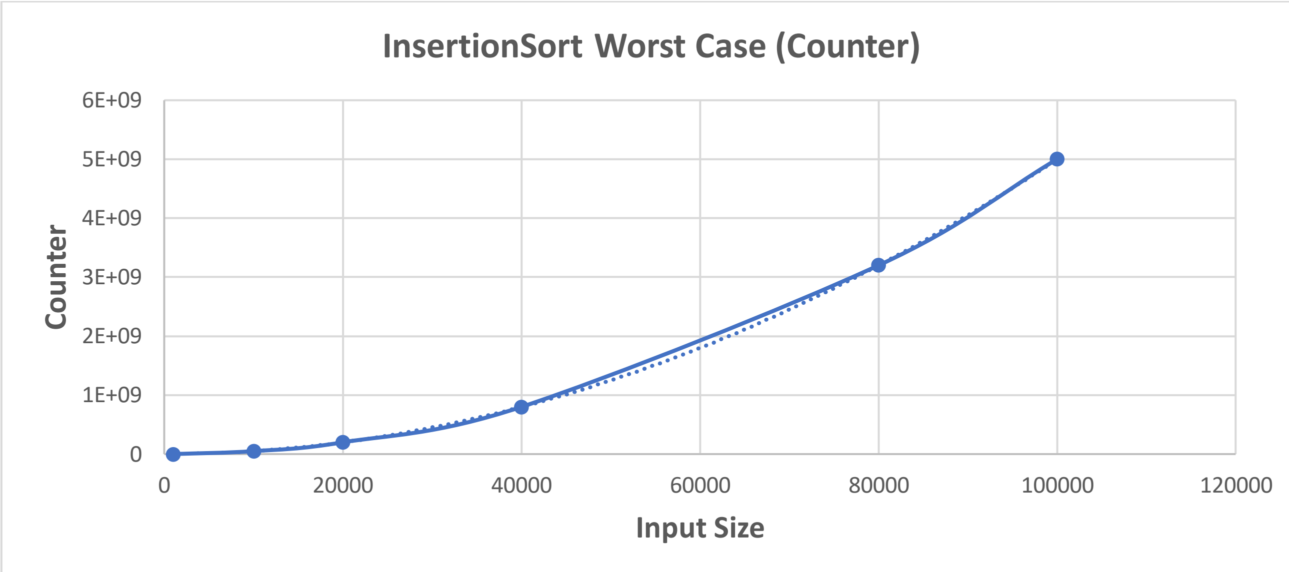
(Figure 1.4 Insertion Sort Average Case (Seconds))

Worst Case Input

We thought that for the worst case is when our list is in the exact opposite order our need. For example, we need to order our list increasing order but list is in descending order.

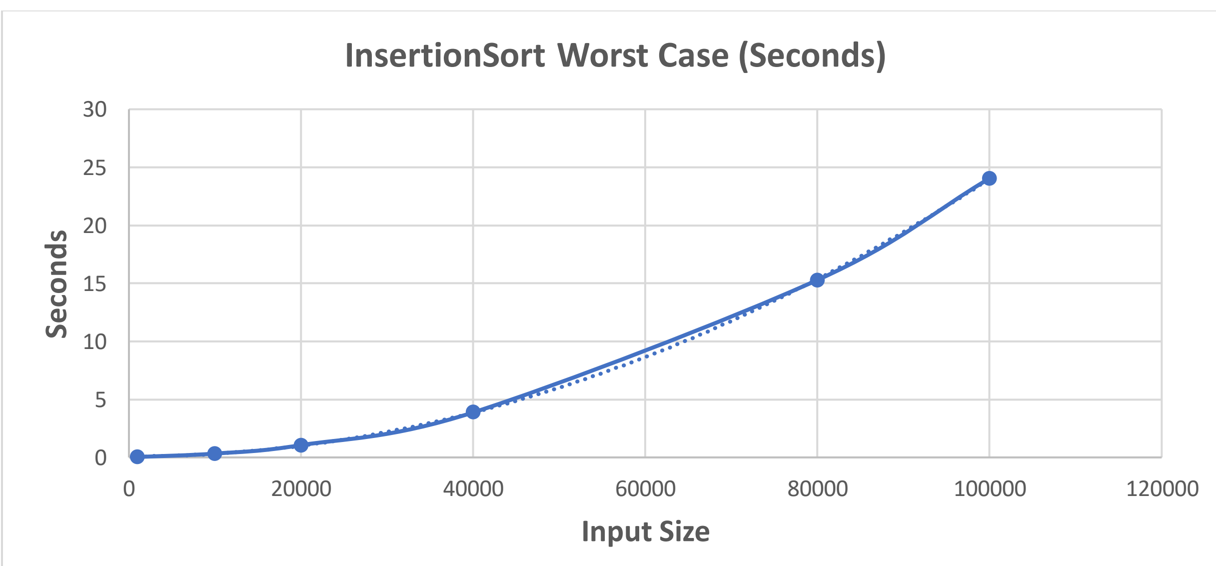
Input size	Insertion Sort Worst Case (Counter)
1000	500499
10000	50004999
20000	200009999
40000	800019999
80000	3200039999
100000	5000049999

By the empirical results time complexity are approximately n^2/2 for n input. So, we can say O(n^2) for empirical results and O(n^2) for theoretical results. And our findings meet theoretical expectations.



(Figure 1.5 Insertion Sort Worst Case (Counter))

Input size	Insertion Sort Worst Case (Seconds)
1000	0.0354163
10000	0.3303428
20000	1.0590523
40000	3.8760297
80000	15.285835
100000	24.0435508



(Figure 1.6 Insertion Sort Worst Case(Seconds))

Merge Sort

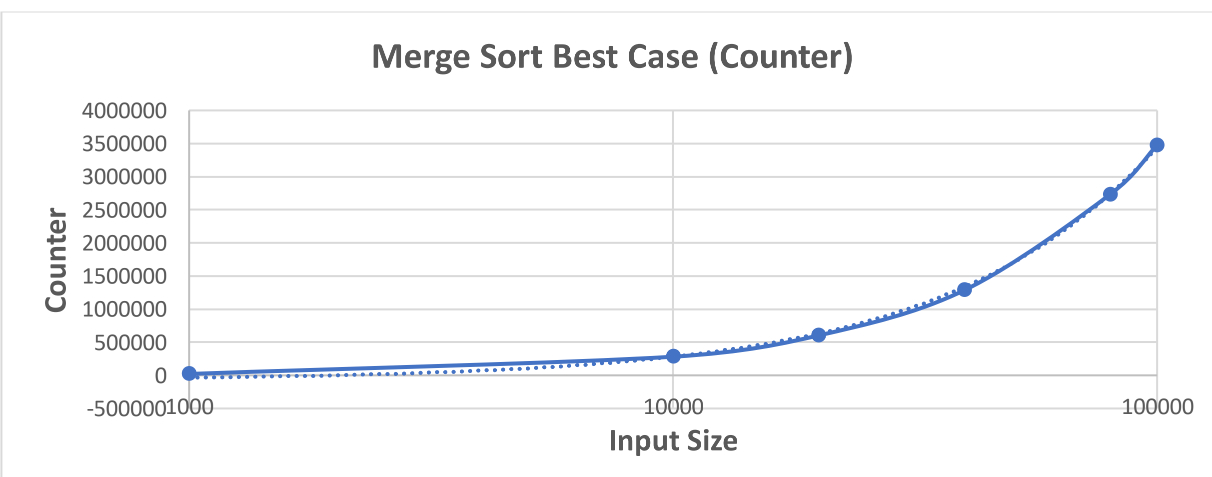
Merge sort (also commonly spelled mergesort) is an efficient, general-purpose, comparison-based sorting algorithm. Most implementations produce a stable sort, which means that the order of equal elements is the same in the input and output.

For Merge Sort we prepare 6 best, average and worst input cases with different kind of inputs and different size (Range: 1000-100000) integers file to measure the efficiency of Merge Sort.

Best Case Input

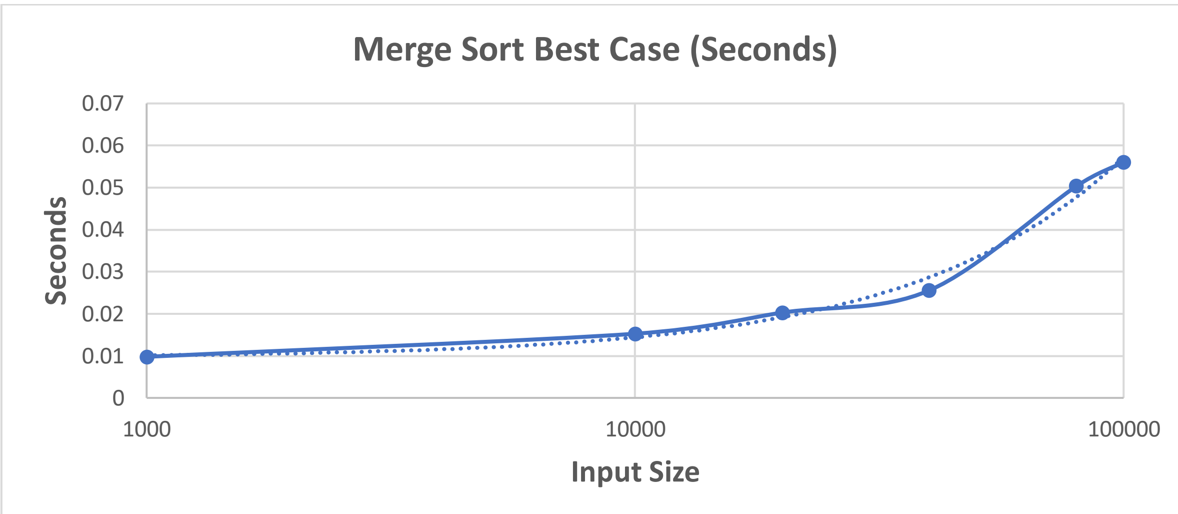
Input size	Merge Sort Best Case (Counter)
1000	21063
10000	281631
20000	603263
40000	1286527
80000	2733055
100000	3476735

By the empirical results time complexity are approximately (n*logn) for n input. So, we can say O(n*logn) for empirical results and O(n*logn) for theoretical results. And our findings meet theoretical expectations.



(Figure 2.1 Merge Sort Best Case (Counter))

Input size	Merge Sort Best Case (Seconds)
1000	0.0097699
10000	0.0152537
20000	0.0202777
40000	0.0255444
80000	0.0503117
100000	0.0560261

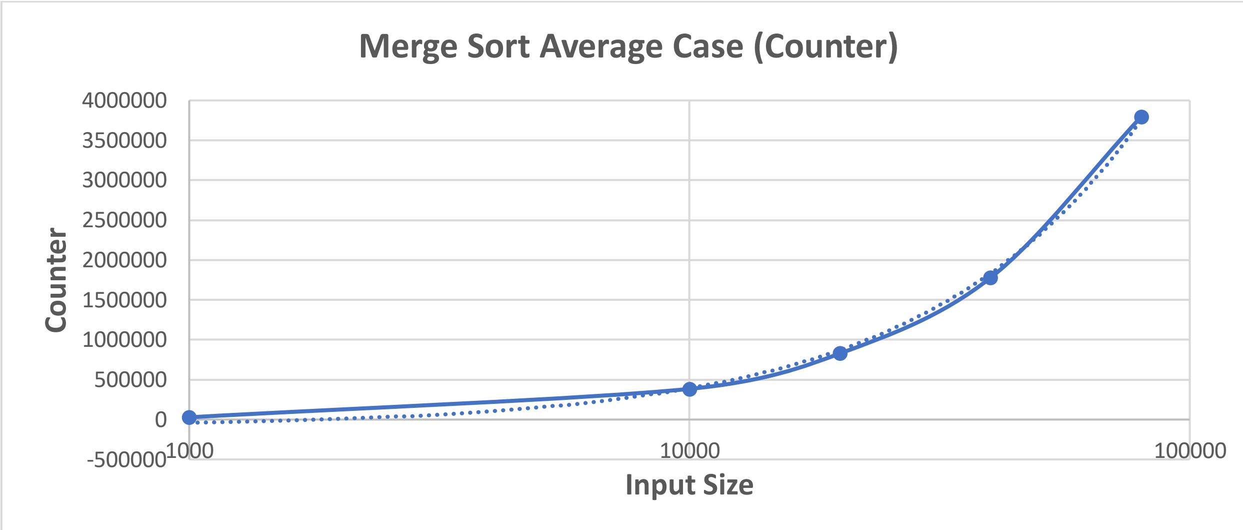


(Figure 2.2 Merge Sort Best Case (Seconds))

Average Case Input

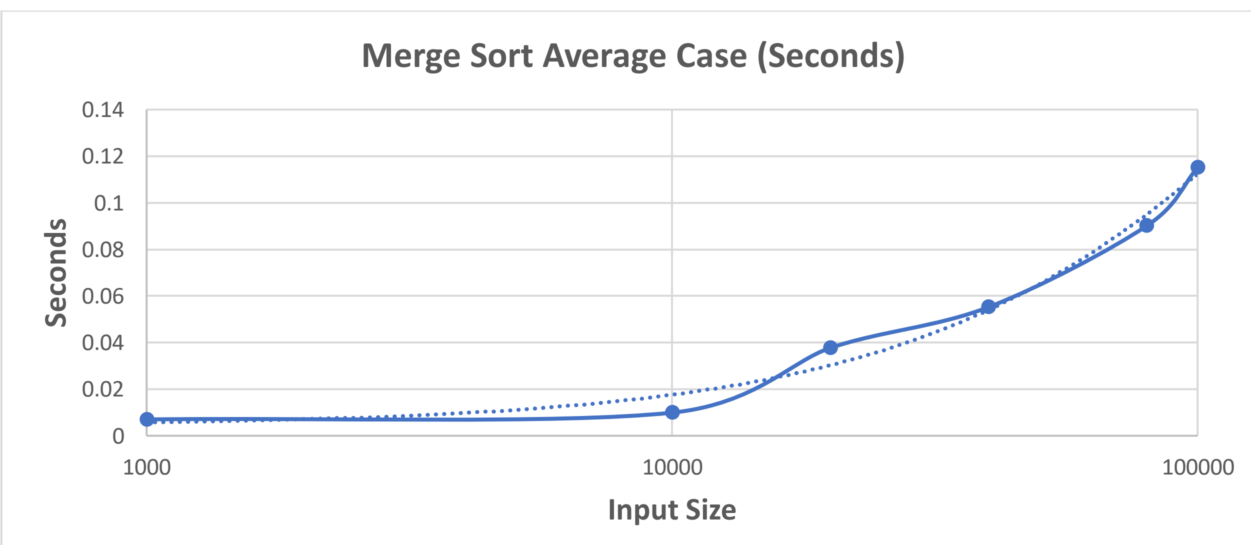
Input size	Merge Sort Average Case (Counter)
1000	28413
10000	384471
20000	829139
40000	1778449
80000	3796207
100000	4841801

By the empirical results time complexity are approximately (n*logn) for n input. So, we can say O(n*logn) for empirical results and O(n*logn) for theoretical results. And our findings meet theoretical expectations.



(Figure 2.3 Merge Sort Average Case (Counter))

Input size	Merge Sort Best Case (Seconds)
1000	0.006994604
10000	0.009916462
20000	0.037701756
40000	0.055205195
80000	0.090076175
100000	0.115092291

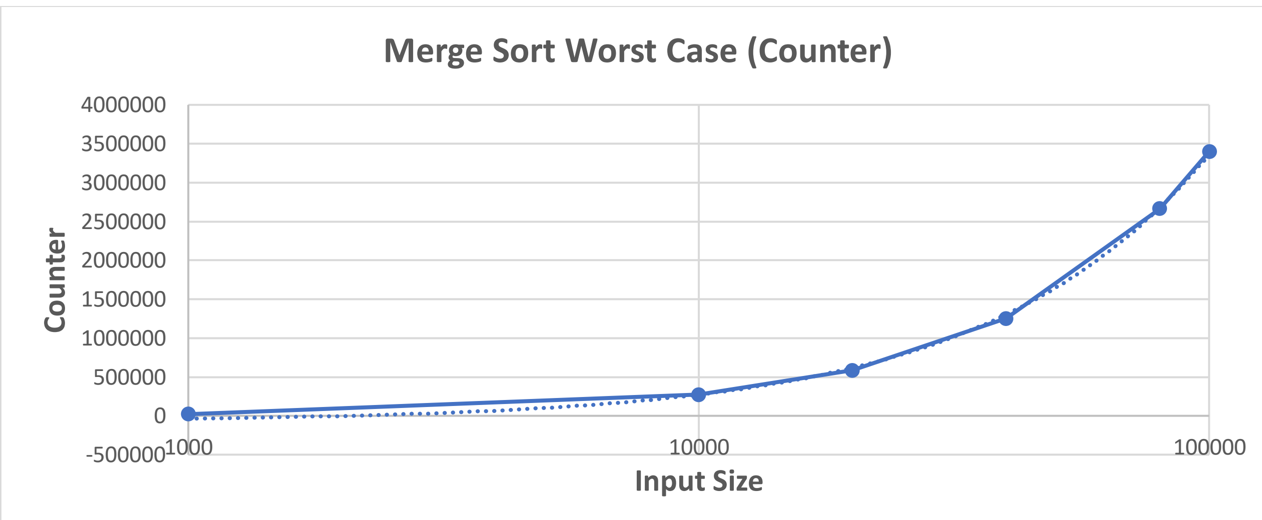


(Figure 2.4 Merge Sort Average Case (Seconds))

Worst Case Input

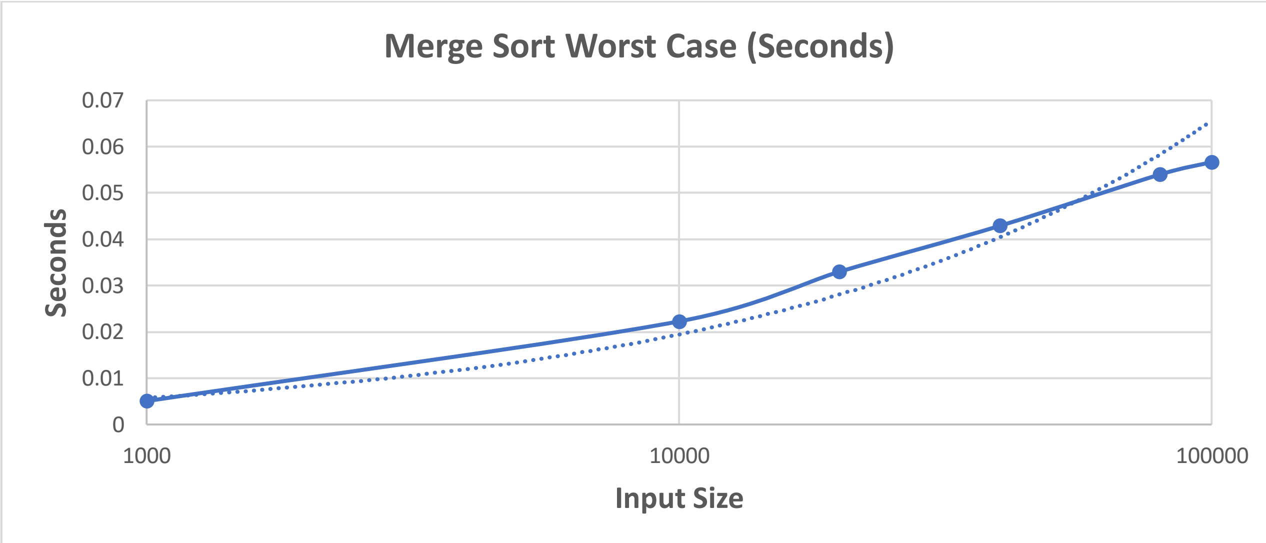
Input size	Merge Sort Worst Case (Counter)
1000	20839
10000	272831
20000	585663
40000	1251327
80000	2662655
100000	3398975

By the empirical results time complexity are approximately (n*logn) for n input. So, we can say O(n*logn) for empirical results and O(n*logn) for theoretical results. And our findings meet theoretical expectations.



(Figure 2.5 Merge Sort Worst Case (Counter))

Input size	Merge Sort Worst Case (Seconds)
1000	0.0050519
10000	0.0222638
20000	0.0329898
40000	0.0428908
80000	0.0539911
100000	0.056651



(Figure 2.6 Merge Sort Worst Case (Seconds))

Max Heap

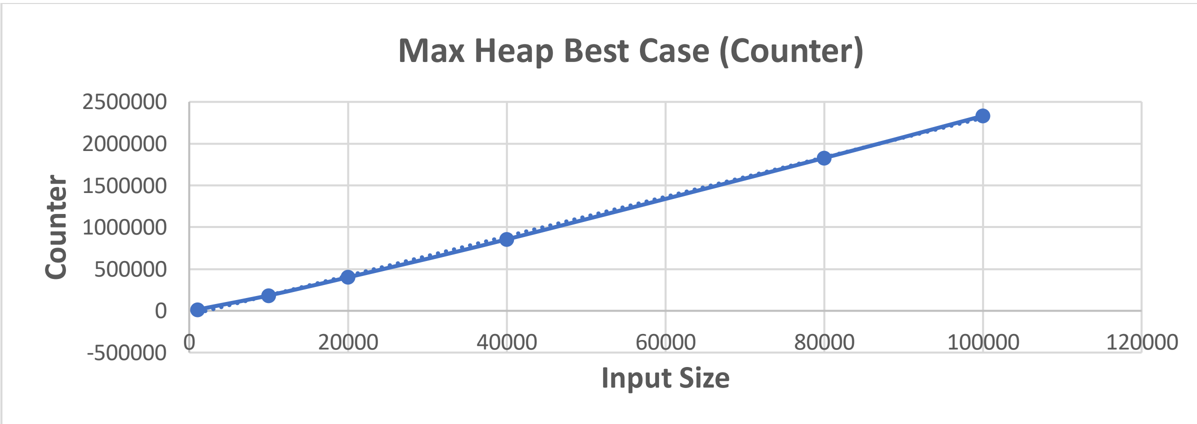
In computer science, a min-max heap is a complete binary tree data structure which combines the usefulness of both a min-heap and a max-heap, that is, it provides constant time retrieval and logarithmic time removal of both the minimum and maximum elements in it. This makes the min-max heap a very useful data structure to implement a double-ended priority queue.

For Max Heap we prepare 6 best, average and worst input cases with different kind of inputs and different size (Range: 1000-100000) integers file to measure the efficiency of max-heap.

Best Case Input

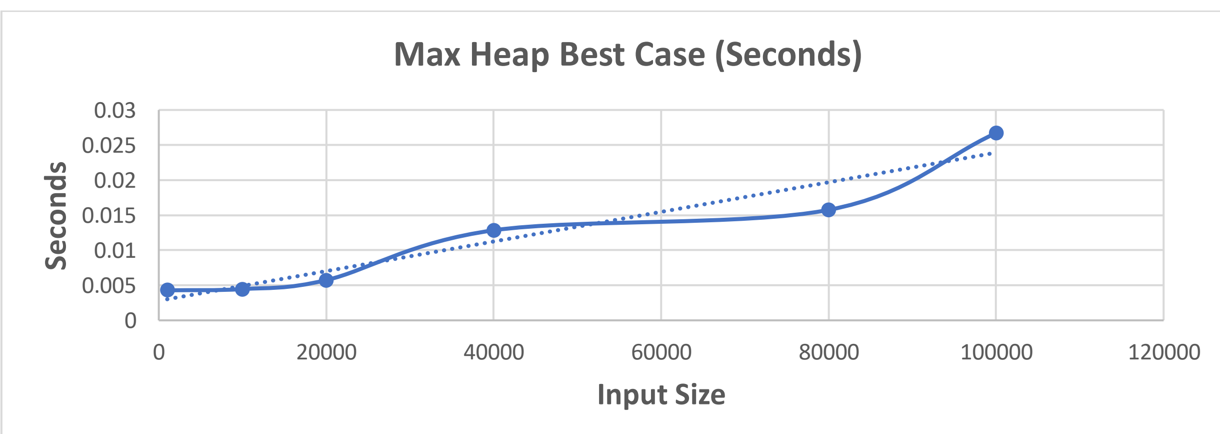
Input size	Max Heap Best Case (Counter)
1000	13295
10000	183261
20000	400764
40000	854858
80000	1828269
100000	2332240

In order to get best case results, we used sorted decreased order input. Because after each insert iteration next number will not be checked for provide max-heap rule. At the end we have done n/2 times max removal. By the empirical results time complexity are approximately (2/5*n) for n input. So, we can say O(n) for empirical results and O(n) for theoretical results. And our findings meet theoretical expectations.



(Figure 3.1 Max Heap Best Case (Counter))

Input size	Max Heap Best Case (Seconds)
1000	0.0042835
10000	0.0044614
20000	0.0057378
40000	0.0128441
80000	0.0157498
100000	0.0267158

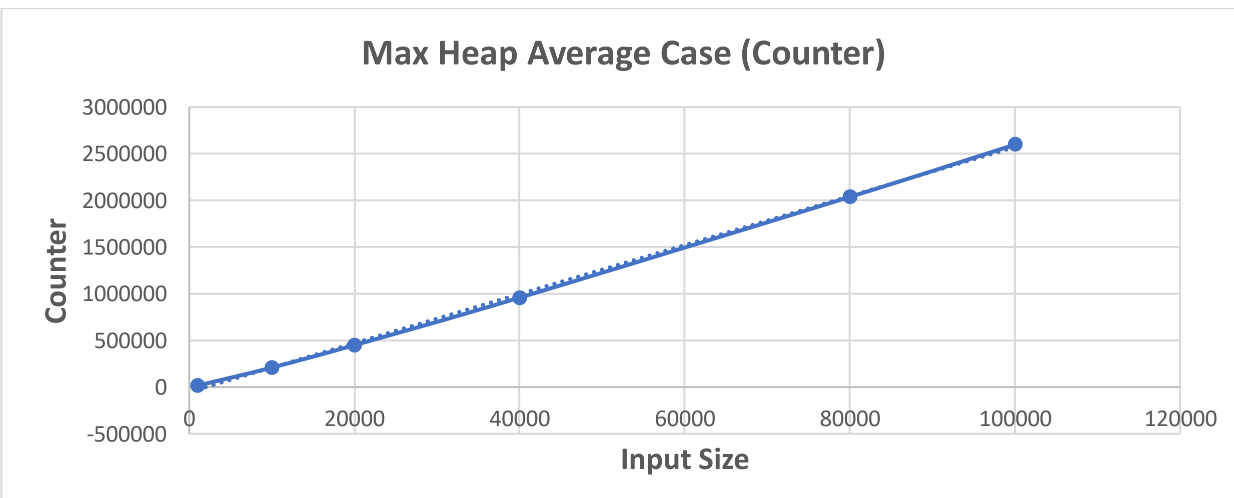


(Figure 3.2 Max Heap Best Case (Seconds))

Average Case Input

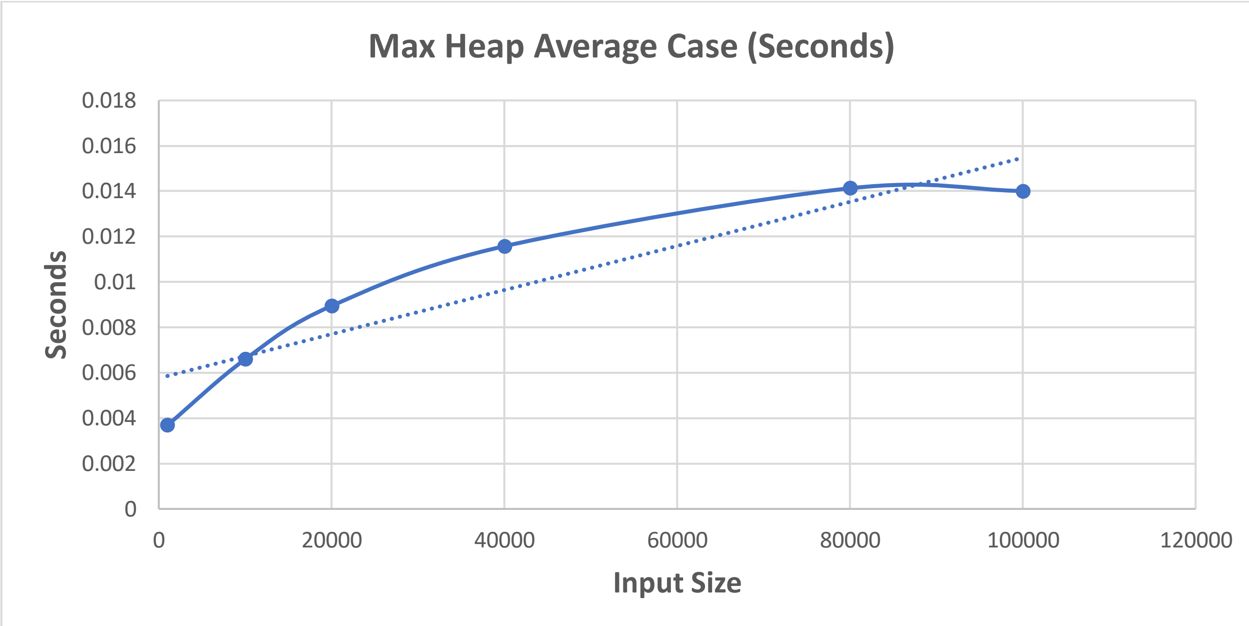
Input size	Max Heap Average Case (Counter)
1000	16007
10000	209604
20000	449897
40000	958454
80000	2037682
100000	2598244

In order to get average case results, we simply used random ordered numbers. At the end we have done n/2 times max removal. By the empirical results time complexity are approximately (2/5*n) for n input. So, we can say O(n) for empirical results and O(n) for theoretical results. And our findings meet theoretical expectations.



(Figure 3.3 Max Heap Average Case (Counter))

Input size	Max Heap Best Case (Seconds)
1000	0.0036866
10000	0.0065945
20000	0.0089449
40000	0.0115759
80000	0.0141289
100000	0.0140092

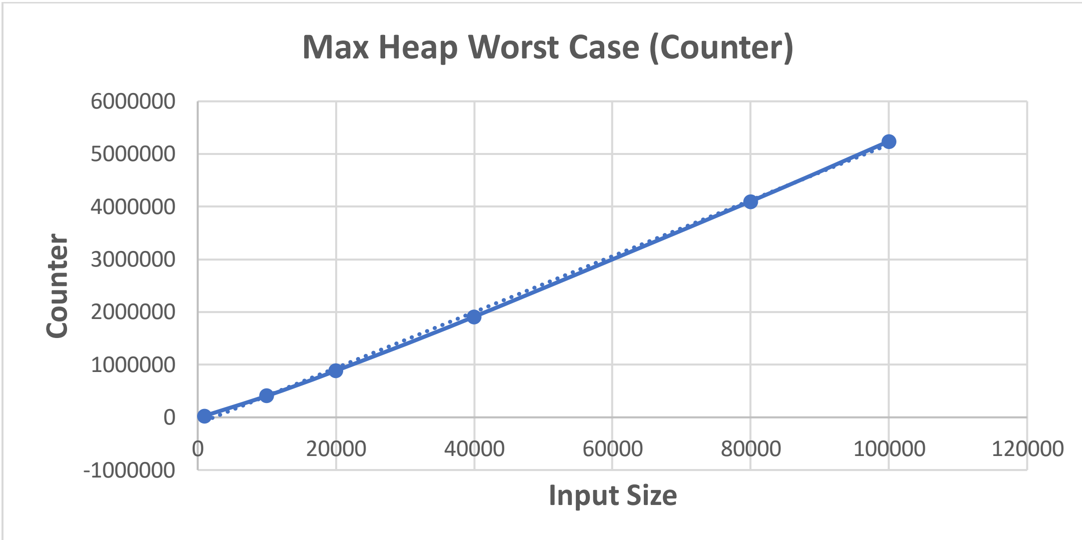


(Figure 3.4 Max Heap Average Case (Seconds))

Worst Case Input

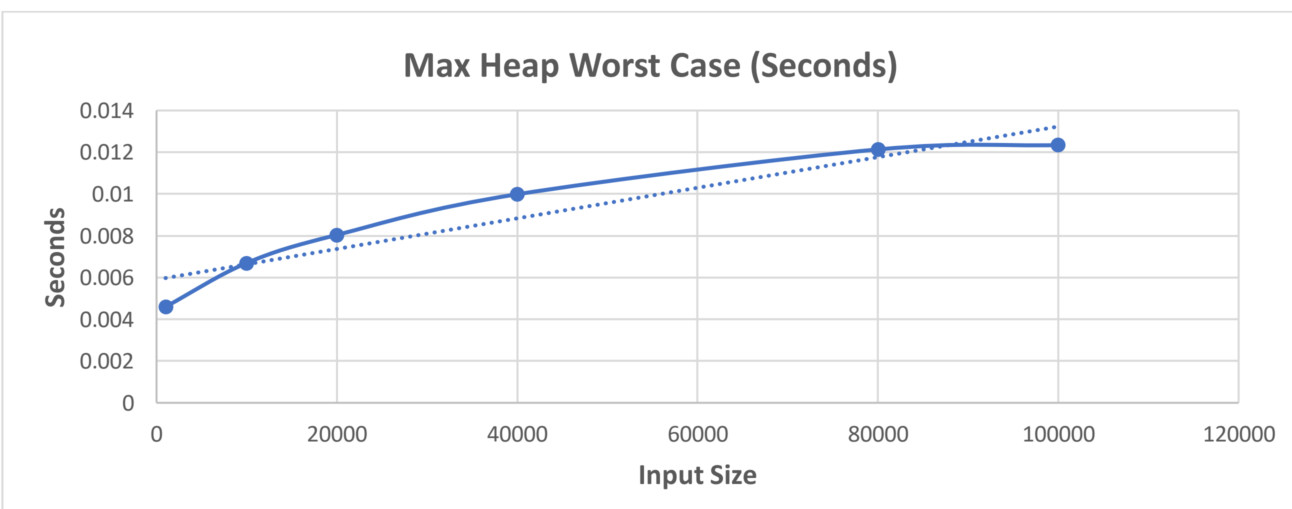
Input size	Max Heap Worst Case (Counter)
1000	28963
10000	407723
20000	885384
40000	1910701
80000	4101330
100000	5241330

In worst case we used sorted increased input in order to get worst results. Every insert iteration it checks the number and put the number to the bottom of the heap. At the end we have done n/2 times max removal. By the empirical results time complexity are approximately (3/5*n) for n input. So, we can say O(n) for empirical results and O(n) for theoretical results. And our findings meet theoretical expectations.



(Figure 3.5 Max Heap Worst Case (Counter))

Input size	Max Heap Worst Case (Seconds)
1000	0.0045998
10000	0.0066974
20000	0.0080421
40000	0.0099881
80000	0.0121347
100000	0.012344



(Figure 3.6 Max Heap Worst Case (Seconds))

Quick Select

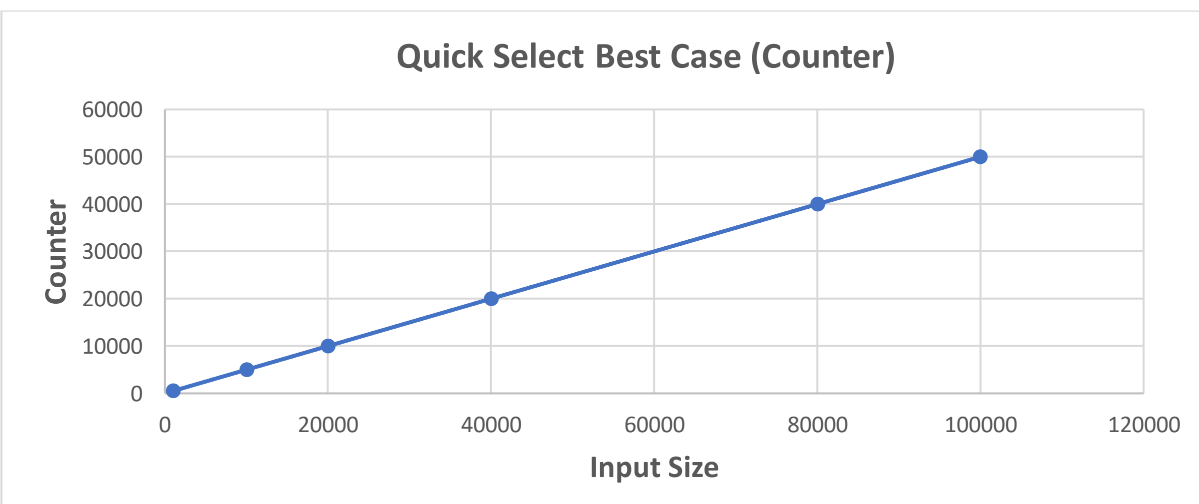
In computer science, quick select is a selection algorithm to find the kth smallest element in an unordered list. It is related to the quicksort sorting algorithm.

For Quick Select we prepare 6 best, average and worst input cases with different kind of inputs and different size (Range: 1000-100000) integers file to measure the efficiency of insertion sort.

Best Case Input

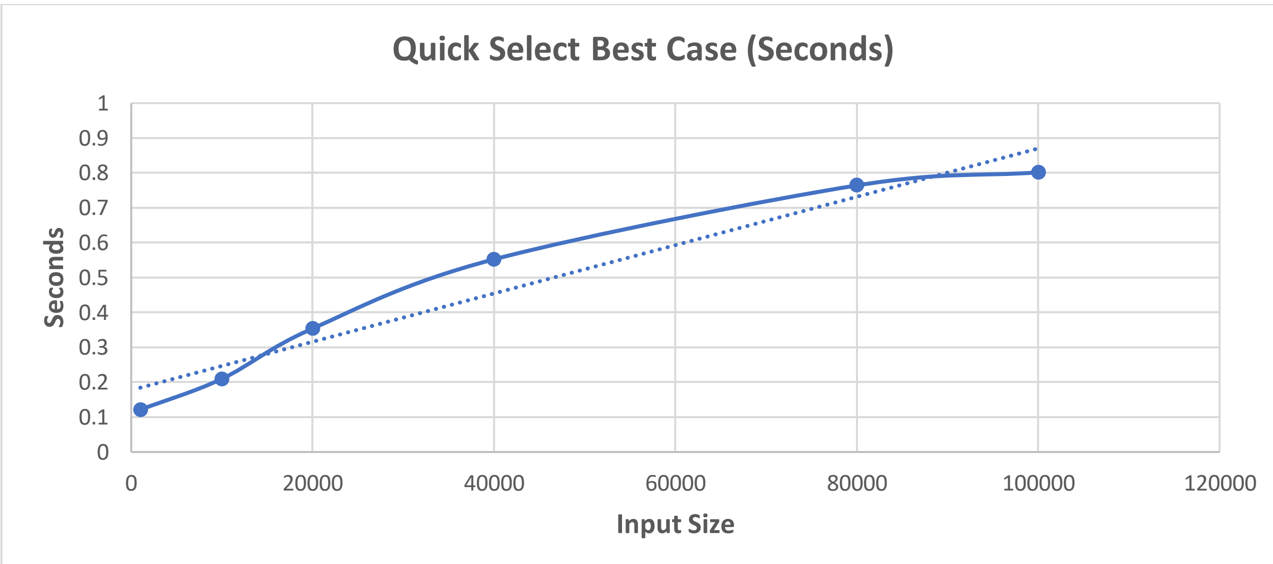
Input size	Quick Select Best Case (Counter)
1000	500
10000	5000
20000	10000
40000	20000
80000	40000
100000	50000

By the empirical results time complexity are exactly n for n input. So, we can say O(n) for empirical results and O(n) for theoretical results. And our findings meet theoretical expectations.



(Figure 4.1 Quick Select Best Case (Counter))

Input size	Quick Select Best Case (Seconds)
1000	0.121261667
10000	0.209836939
20000	0.353977112
40000	0.552340998
80000	0.764149795
100000	0.801555904

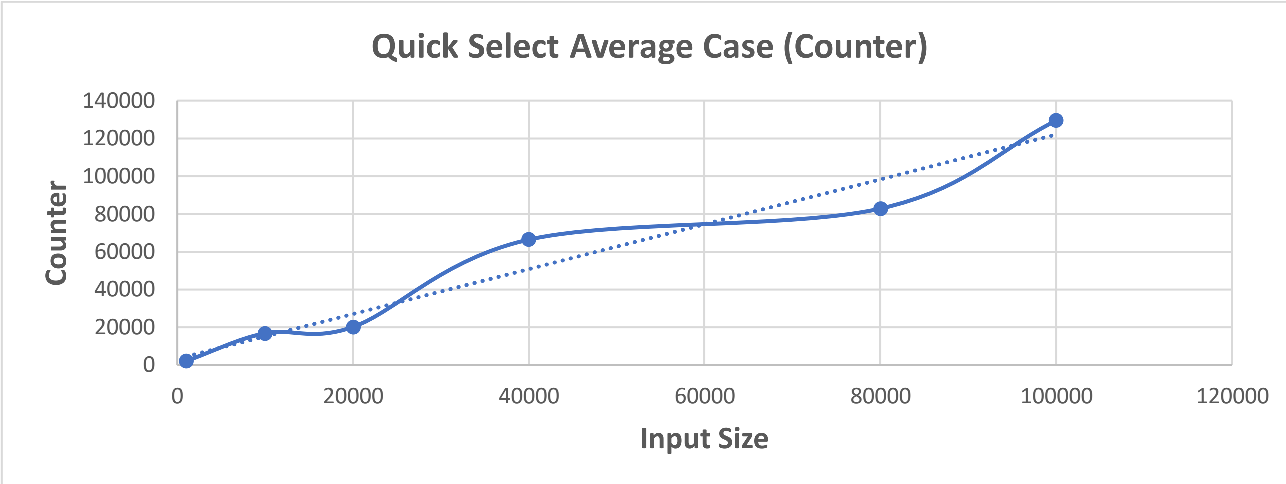


(Figure 4.2 Quick Select Best Case (Seconds))

Average Case Input

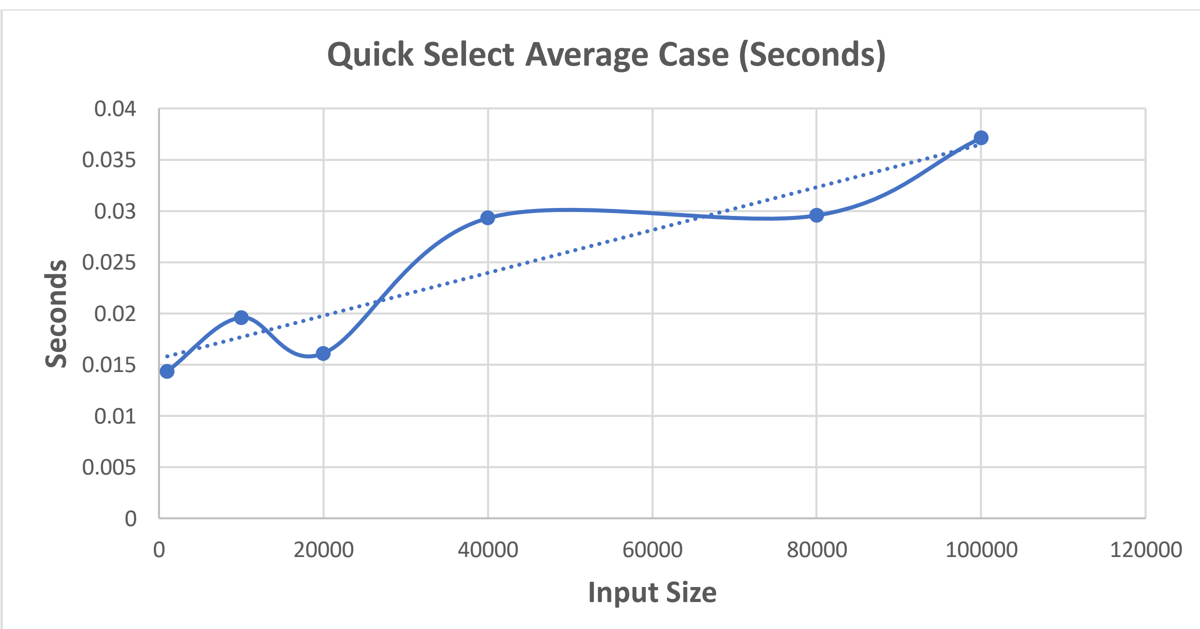
Input size	Quick Select Average Case (Counter)
1000	1951
10000	16767
20000	20040
40000	66321
80000	82747
100000	129402

By the empirical results time complexity are approximately n for n input. So, we can say O(n) for empirical results and O(n) for theoretical results. And our findings meet theoretical expectations.



(Figure 4.3 Quick Select Average Case (Counter))

Input size	Quick Select Average Case (Seconds)
1000	0.014349914
10000	0.019584677
20000	0.01609808
40000	0.029288781
80000	0.029553486
100000	0.037107899

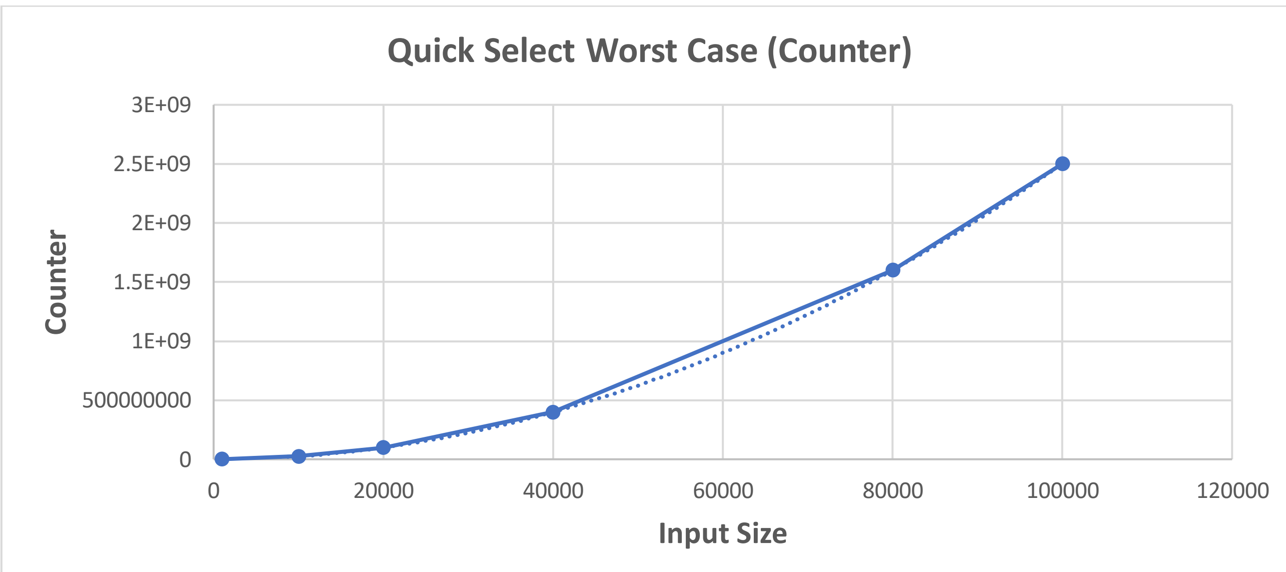


(Figure 4.4 Quick Select Average Case (Seconds))

Worst Case Input

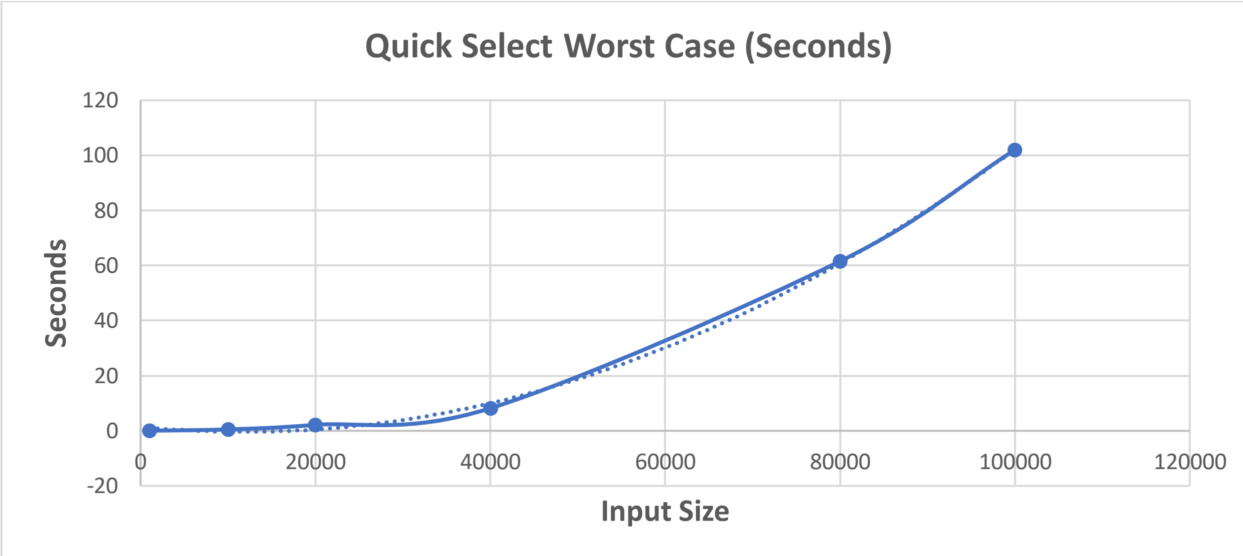
Input size	Quick Select Worst Case (Counter)
1000	251000
10000	25010000
20000	100020000
40000	400040000
80000	1600080000
100000	2500100000

By the empirical results time complexity are approximately n^2 for n input. So, we can say O(n^2) for empirical results and O(n^2) for theoretical results. And our findings meet theoretical expectations.



(Figure 4.5 Quick Select Worst Case (Counter))

Input size	Quick Select Worst Case (Seconds)
1000	0.044210648
10000	0.518777626
20000	2.16898097
40000	8.189583243
80000	61.56037577
100000	101.9711237



(Figure 4.6 Quick Select Worst Case (Counter))

Quick Select Median of Three

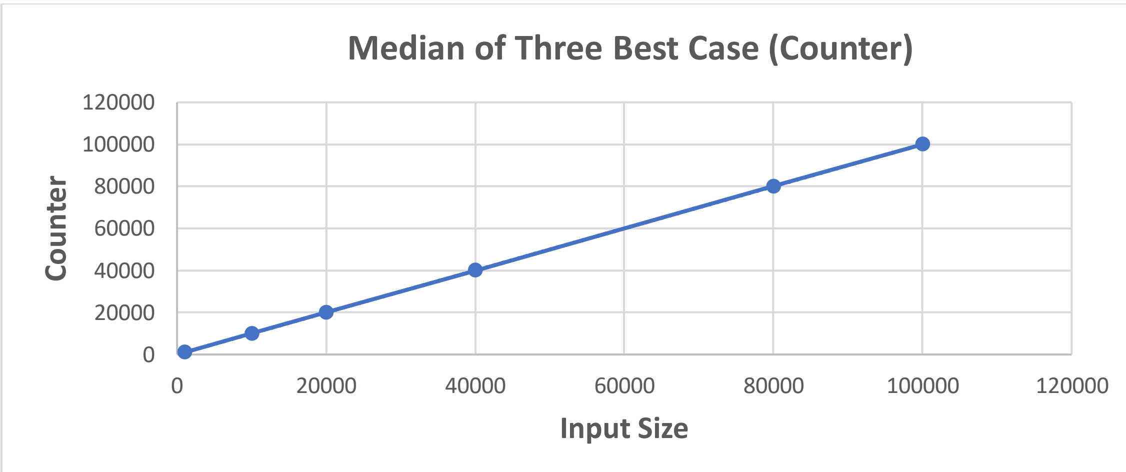
In computer science, quick select is a selection algorithm to find the kth smallest element in an unordered list. It is related to the quicksort sorting algorithm. In this section we make decision on choosing pivot by applying Median-of-Three algorithm to optimize worst case scenario in order to eliminate O(n^2) time complexity to O(nlogn) complexity.

For Median of Three we prepare 6 best, average and worst input cases with different kind of inputs and different size (Range: 1000-100000) integers file to measure the efficiency of Median-of-Three algorithm.

Best Case Input

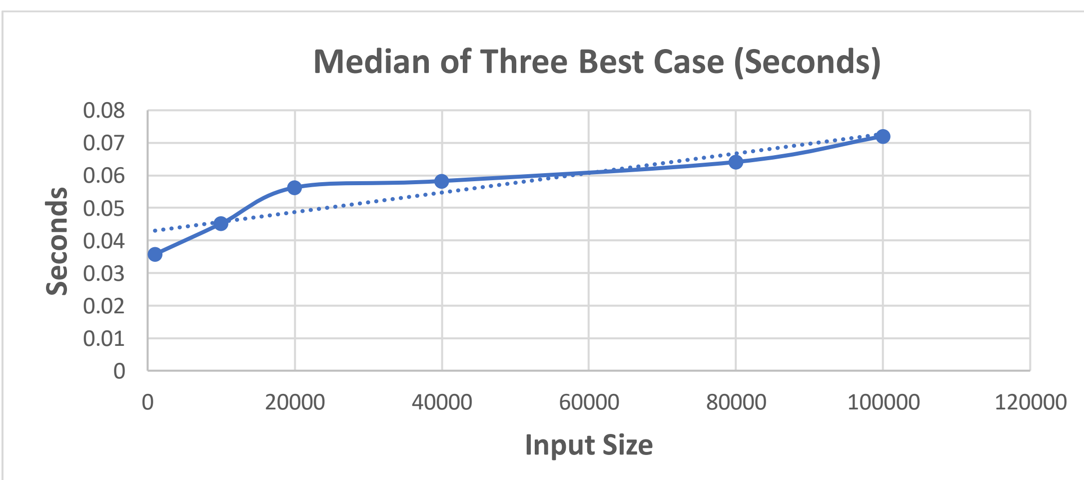
Input size	Median of Three Best Case (Counter)
1000	998
10000	9998
20000	19998
40000	39998
80000	79998
100000	99998

By the empirical results time complexity are approximately n for n input. So, we can say O(n) for empirical results and O(n) for theoretical results. And our findings meet theoretical expectations.



(Figure 5.1 Quick Select Median of Three Best Case (Counter))

Input size	Median of Three Best Case (Seconds)
1000	0.035776266
10000	0.045186816
20000	0.056269975
40000	0.058269975
80000	0.064120395
100000	0.072008299

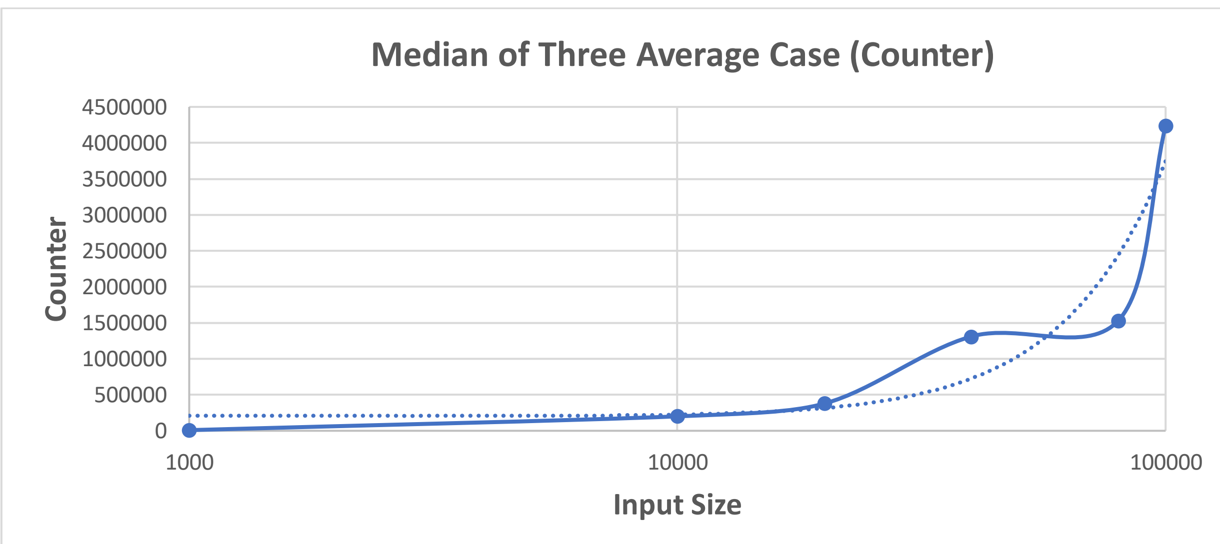


(Figure 5.2 Quick Select Median of Three Best Case (Seconds))

Average Case Input

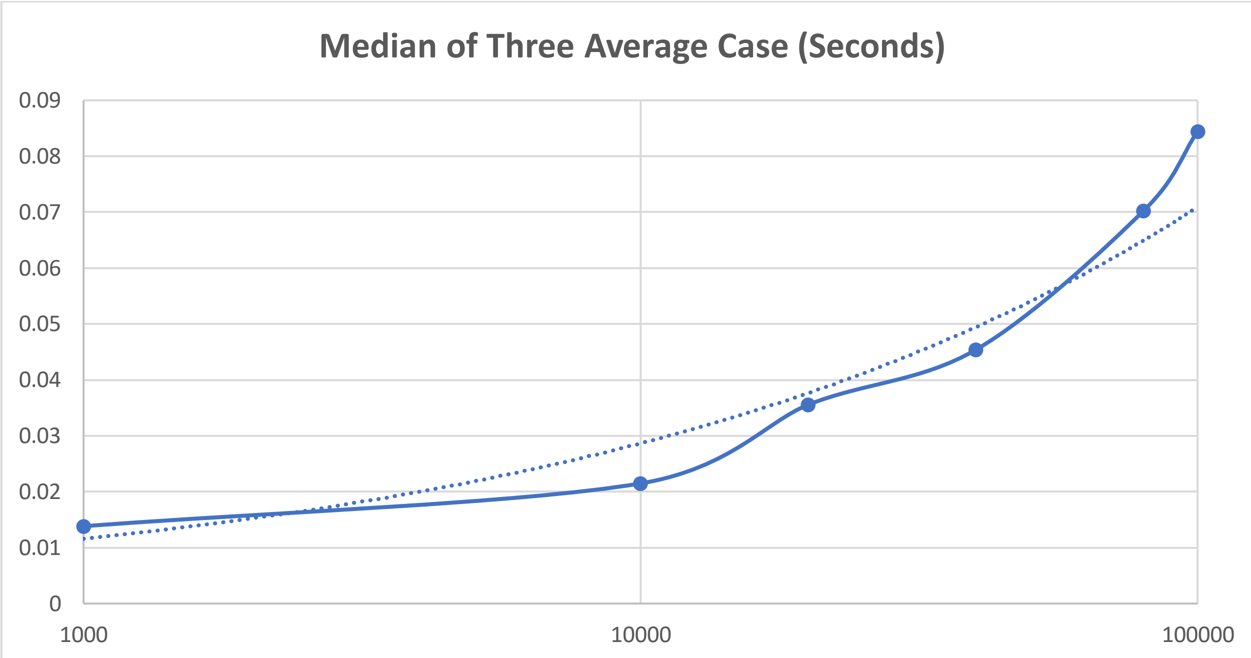
Input size	Median of Three Average Case (Counter)
1000	8750
10000	200322
20000	378192
40000	1310390
80000	1530708
100000	4241660

By the empirical results time complexity are approximately 3/2* nlogn for n input. So, we can say O(nlogn) for empirical results and O(nlogn) for theoretical results. And our findings meet theoretical expectations.



(Figure 5.3 Quick Select Median of Three Average Case (Counter))

Input size	Median of Three Average Case (Seconds)
1000	0.013825269
10000	0.02148734
20000	0.035565078
40000	0.045399149
80000	0.070256272
100000	0.084463293

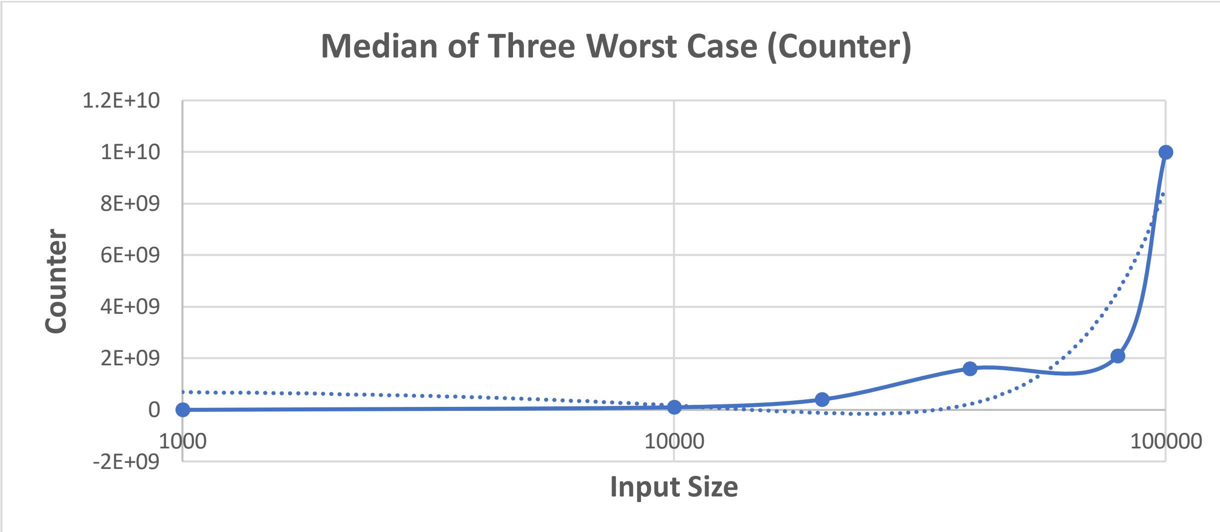


(Figure 5.4 Quick Select Median of Three Average Case (Seconds))

Worst Case Input

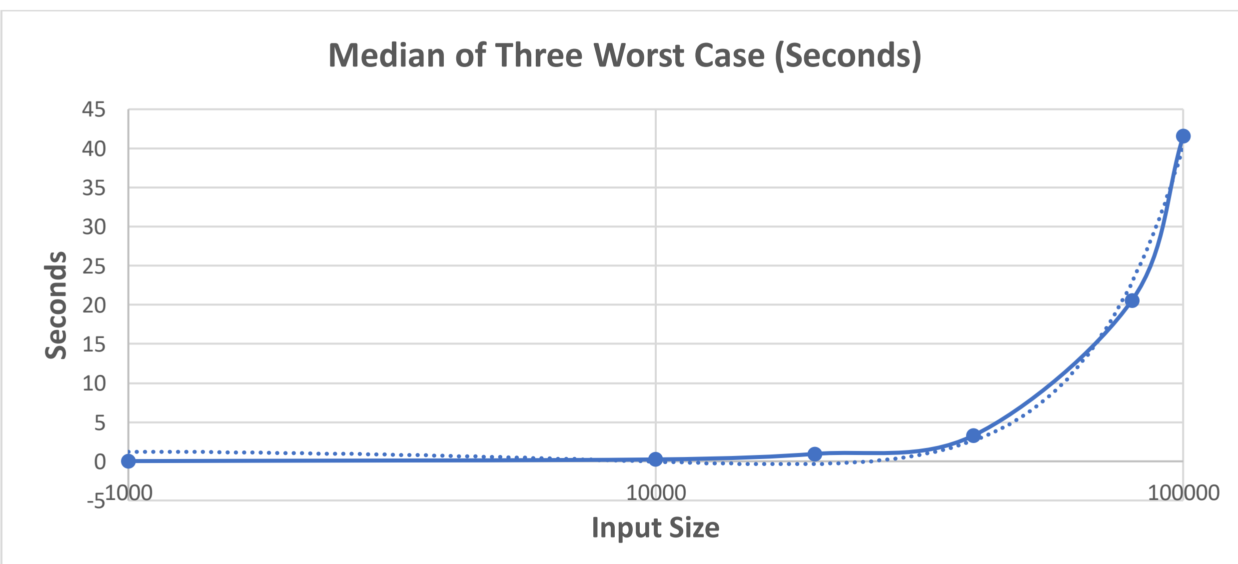
Input size	Median of Three Worst Case (Counter)
1000	995008
10000	99950008
20000	399900008
40000	1599800008
80000	2104632712
100000	9999500008

By the empirical results time complexity are approximately 2* nlogn for n input. So, we can say O(nlogn) for empirical results and O(nlogn) for theoretical results. And our findings meet theoretical expectations.



(Figure 5.5 Quick Select Median of Three Worst Case (Counter))

Input size	Median of Three Worst Case (Seconds)
1000	0.030868316
10000	0.245377336
20000	0.952799105
40000	3.288331774
80000	20.58991833
100000	41.54238538



(Figure 5.6 Quick Select Median of Three Worst Case (Seconds))

Quick Select Median of Medians

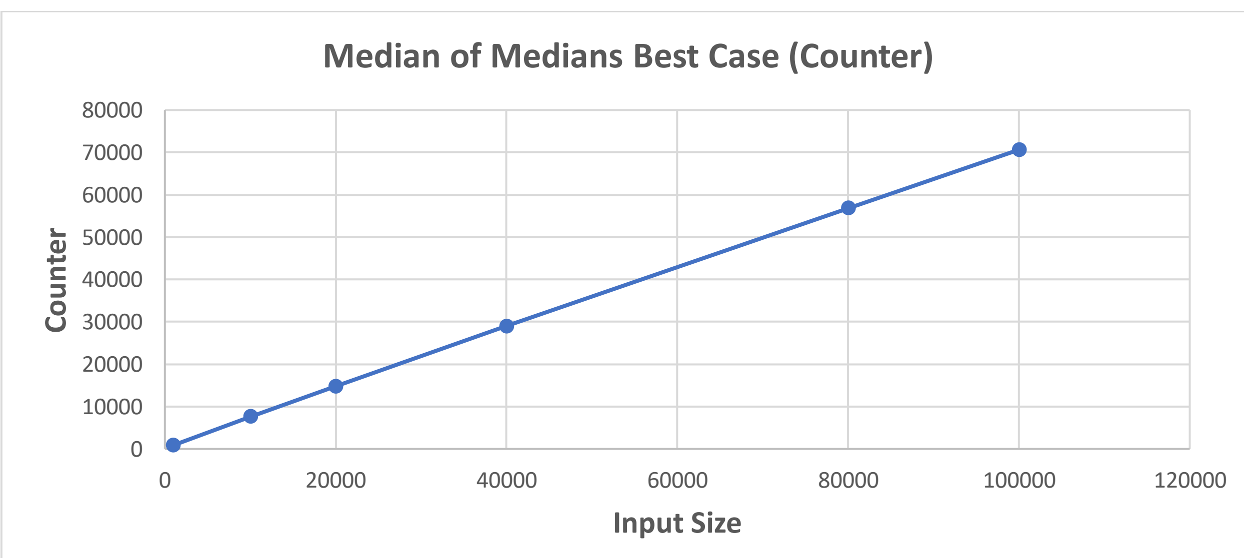
In computer science, the median of medians is an approximate (median) selection algorithm, frequently used to supply a good pivot for an exact selection algorithm, mainly the quick select, that selects the *k*th largest element of an initially unsorted array. Median of medians finds an approximate median in linear time only, which is limited but an additional overhead for quick select.

For Medians of Medians we prepare 6 best, average and worst input cases with different kind of inputs and different size (Range: 1000-100000) integers file to measure the efficiency of insertion sort.

Best Case

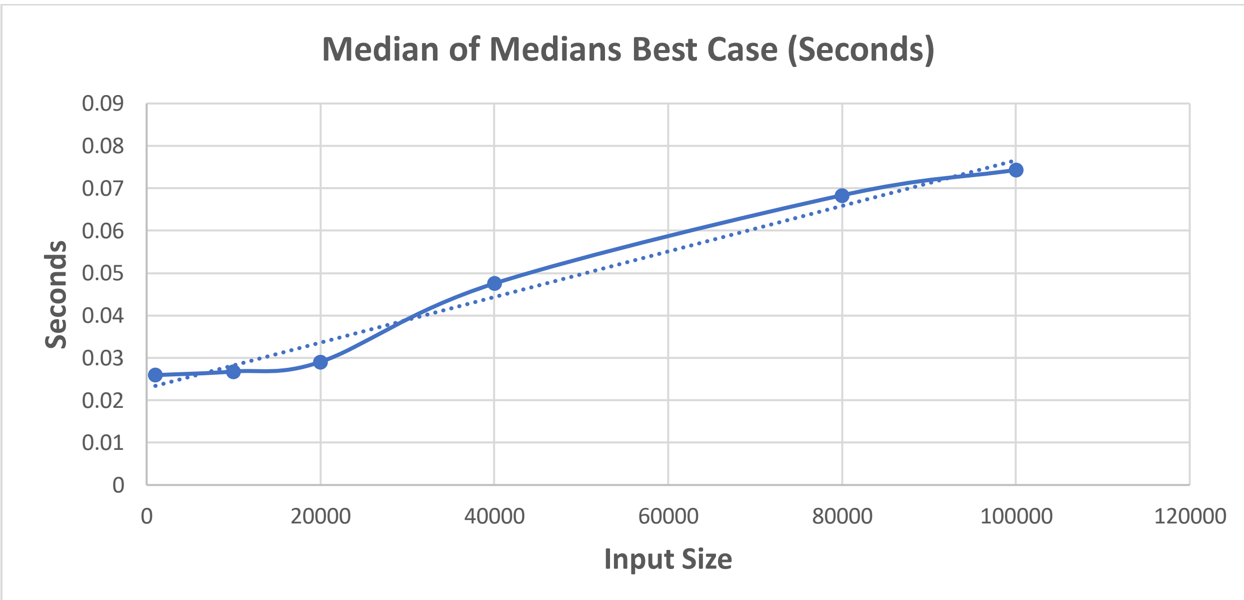
Input size	Median of Medians Best Case (Counter)
1000	883
10000	7632
20000	14807
40000	28939
80000	56832
100000	70622

By the empirical results time complexity are approximately n for n input. So, we can say O(n) for empirical results and O(n) for theoretical results. And our findings meet theoretical expectations.



(Figure 6.1 Quick Select Median of Medians Best Case (Counter))

Input size	Median of Medians Best Case (Seconds)
1000	0.025920005
10000	0.026788438
20000	0.029044931
40000	0.04754105
80000	0.068344903
100000	0.074328295

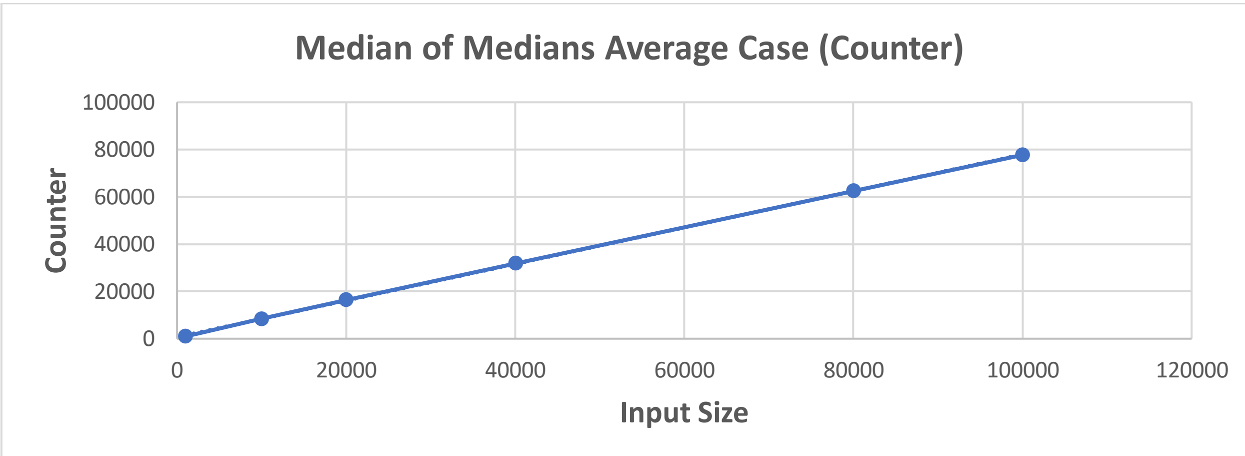


(Figure 6.2 Quick Select Median of Medians Best Case (Seconds))

Average Case Input

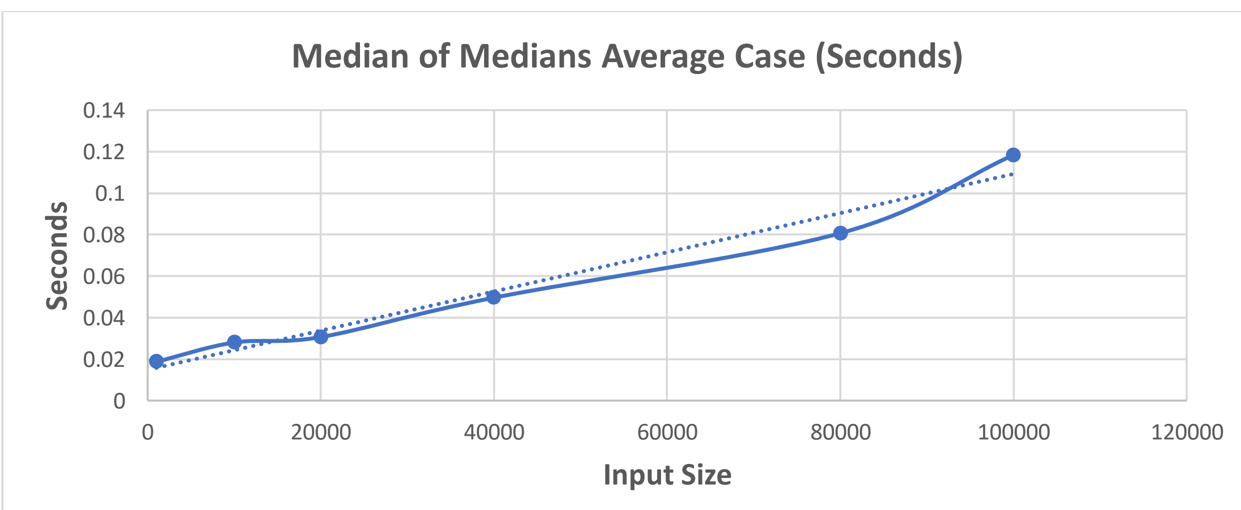
Input size	Median of Medians Average Case (Counter)
1000	970
10000	8464
20000	16393
40000	31916
80000	62523
100000	77724

By the empirical results time complexity are approximately n for n input. So, we can say O(n) for empirical results and O(n) for theoretical results. And our findings meet theoretical expectations.



(Figure 6.3 Quick Select Median of Medians Average Case (Counter))

Input size	Median of Medians Average Case (Seconds)
1000	0.018720693
10000	0.028007913
20000	0.030632883
40000	0.049620261
80000	0.080701835
100000	0.118408608

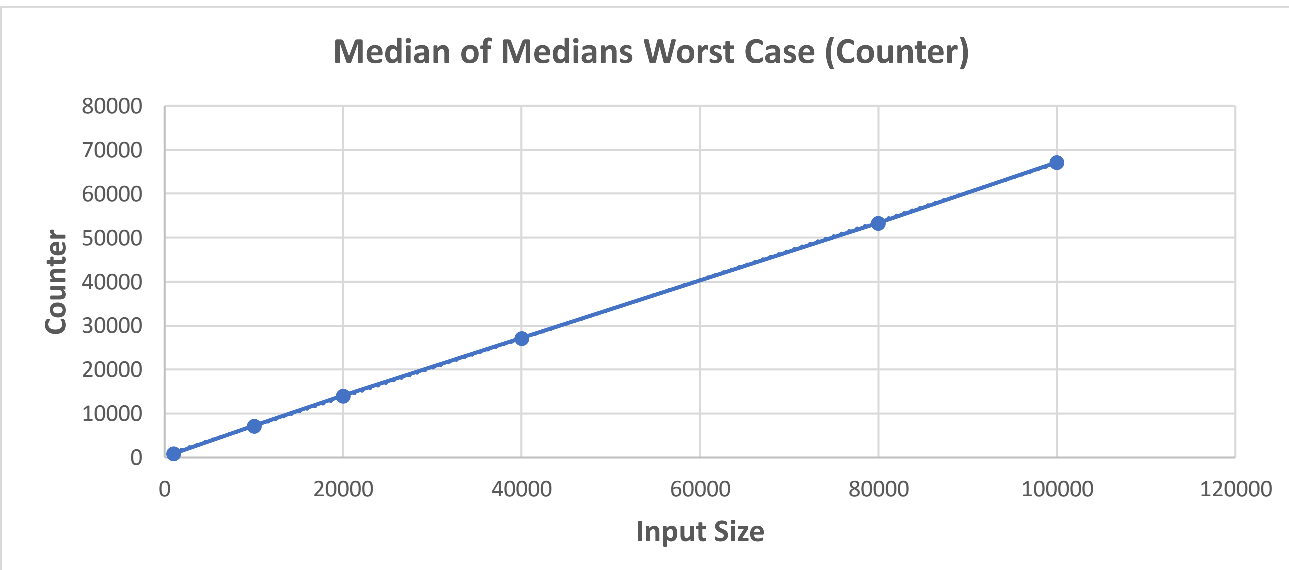


(Figure 6.4 Quick Select Median of Medians Average Case (Seconds))

Worst Case Input

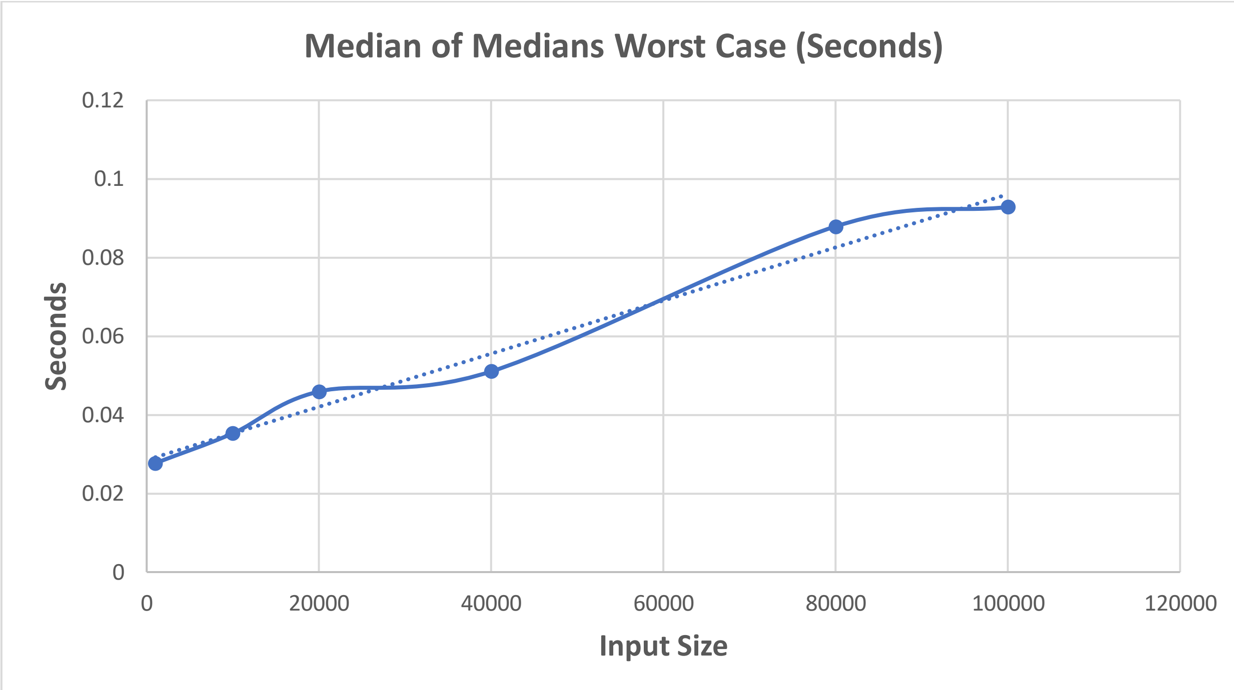
Input size	Median of Medians Worst Case (Counter)
1000	883
10000	7221
20000	14052
40000	27123
80000	53338
100000	67118

By the empirical results time complexity are approximately n for n input. So, we can say O(n) for empirical results and O(n) for theoretical results. And our findings meet theoretical expectations.



(Figure 6.5 Quick Select Median of Medians Worst Case (Counter))

Input size	Median of Medians Worst Case (Seconds)
1000	0.027704368
10000	0.035277755
20000	0.045926682
40000	0.05102347
80000	0.087912984
100000	0.092850784



(Figure 6.6 Quick Select Median of Medians Worst Case (Seconds))

Analyzing Results:

- Insertion sort has a simple implementation. It can be efficient for (quite) small data sets. It is also more efficient in practice than most other simple quadratic algorithms such as bubble sort selection sort. Its best case is nearly O(n). However, it is less efficient on list containing more numbers of elements. As the number of elements increases the performance of the program would be slow. The insertion sort is particularly useful only when sorting a list of few elements.
- Merge sort can be applied to files of any size. It is much faster than insertion and bubble sort for larger inputs. Because merge sort doesn't go through the whole list several times. It has a consistent running time, carries out different bits with similar times in a stage. However, it uses more memory space to store to sub elements of the initial split list. It goes through the whole process even list is sorted. It is not faster in comparative part in other sorting algorithms for smaller lists.
- Max Heap helps to find the greatest element in heap easily. Heap data structure efficiently use graph algorithms such as Dijkstra. The main advantage of using the heap is its flexibility. That's because memory in this structure can be allocated and removed in any particular order. However,

using heap is storing data on Heap is slower than it would take when using the stack. It also takes so much time to compare and then execute.

- Quick Select is also efficient as quicksort in practice. It has a good average-case performance but has poor worst-case performance it the pivot is selected as first element in the list. As with quicksort, quick select is generally implemented as an in-place algorithm, and beyond selecting the *k*th element, it also partially sorts the data. Quick select and its variants are the selection algorithms most often used in efficient real-world implementations. In quick select performance depends on choosing the pivot element. If good pivots are chosen, meaning ones that consistently decrease the search set by a given fraction, then the search set decreases in size exponentially and by induction (or summing the geometric series) one sees that performance is linear, as each step is linear and the overall time is a constant times this (depending on how quickly the search set reduces). However, if bad pivots are consistently chosen, such as decreasing by only a single element each time, then worst-case performance is quadratic: $O(n^2)$. This occurs for example in searching for the maximum element of a set, using the first element as the pivot, and having sorted data.
- Quick Select Median of Three is one of pivot selection strategy in quick select which yielding linear performance on partially sorted lists. Median of three helps in avoiding the worst-case complexity O(n^2). In the cases of already sorted lists this should take the middle element as the pivot thereby reducing the inefficiency found in normal quick select. Choosing the median of the first, middle and last element of the partition enables us to achieve faster results with the same complexity as quick select. It provides better pivot selection, especially when we do not know about input. Median of three pivot selection like standard quick select is also not a stable sorting algorithm for worst inputs.
- Quick Select Median of Medians can also be used as a pivot strategy in quick select, yielding an optimal algorithm, with worst-case complexity $O(n \log n)$. Although this approach optimizes the asymptotic worst-case complexity quite well, it is typically outperformed in practice by instead choosing random pivots for its average $O(n)$ complexity for selection and average $O(n \log n)$ complexity for sorting, without any overhead of computing the pivot. If one instead consistently chooses "good" pivots, this is avoided and one always gets linear performance even in the worst case. A "good" pivot is one for which we can establish that a constant proportion of elements fall both below and above it, as then the search set decreases at least by a constant proportion at each step, hence exponentially quickly, and the overall time remains linear. The median is a

good pivot – the best for sorting, and the best overall choice for selection – decreasing the search set by half at each step. Thus, if one can compute the median in linear time, this only adds linear time to each step, and thus the overall complexity of the algorithm remains linear.

Work-Share:

Research and Coding → Bilgehan Geçici – Anıl Şenay

Input Generation → Anıl Şenay

Testing and Recording Input Results → Bilgehan Geçici

Report → Bilgehan Geçici – Anıl Şenay

Generally, every group member has done their parts with share-screen method to inform other group members about the work done.