Hacettepe University

Computer Science and Engineering Department

Spring 21’

BBM204 Course Assignment 1 Report

Alihan KARATATAR

21904324

Problem Definition

In this assignment we are expected to implement and measure time complexity of 5 different sorting algorithms which are Gnome Sort, Stooge Sort, Comb Sort, Cocktail Sort and Bitonic Sort. Before start discussion section, you can find brief information about these algorithms in below.

Gnome Sort

The algorithm finds the first place where two adjacent elements are in the wrong order and swaps them. It takes advantage of the fact that performing a swap can introduce a new out-of-order adjacent pair next to the previously swapped elements. It does not assume that elements forward of the current position are sorted, so it only needs to check the position directly before the swapped elements. It's average running time and worst running time are both $O(n^2)$. It is stable sorting algorithm.

Stooge Sort

Stooge sort is a recursive sorting algorithm. First, the algorithm check whether first element is greater than last element or not. If it is, swap them. In 3 element list, this provides ascending order. But if there are more than 3 elements, it recursively calls itself for initial 2/3 part of list, then final 2/3 part of list then again 2/3 part of list. If the list length is not divided by three as integer, the result is rounded up. It's bad time complexity and nearly $O(n^{2,71})$. It is not stable sorting algorithm.

Comb Sort

Comb sort algorithm works like K-Sort algorithm and Bubble Sort algorithm. But Comb Sort arranges first k value as 1.3. It applies integer division, so floating parts become not significant. After that, it changes element k to k over 1.3 till k is smaller than or equal to 1. Its time complexity is O(n2) at worst case, O(n2 /2p), where p is the number of increments, O(nlogn) at best case. It is not stable sorting algorithm.

Shaker Sort

Shaker sort algorithm simply goes first index to last index and then last index to first index and apply Bubble Sort in both direction till list is sorted. It is known as Bidirectional Bubble Sort. Its time complexity is O(n) at best case, O(n2) at average and worst case. It is stable sorting algorithm.

Bitonic Sort

Bitonic Sort is a classic parallel algorithm for sorting for using bitonic sequences which is firstly increases then decreases. It divides list in two parts sort them separately and merge them. it has O(log2n) time complexity in every cases. It is not stable sorting algorithm.

My Solution Approach

I created a ListElement class with have 2 attributes, value and order. It is also comparable by orders. By doing this I am planning to reset and reuse my lists every time after each time I sorted them. After that I created a ListObject class for both checking stability and calculating best, average and worst-case scenarios. In the constructor method of this class, if stability check is true it creates a bounded random array and append it. It is bounded because time constraint. If stability check is false it just creates howManyElement sized random array and append it with temp array. I also said that I arrange scenarios for 3 cases. In switch case part, I assign a status attribute for make changing. For example, if case 1 it is for best-case scenario and it send to algorithms sorted arrays. If case -1 it is for the worst-case scenario and it send to algorithms reverse sorted array. If nothing selected, which is case 0, it generates a random array, and this is for the average case. Under constructor method I create a checkStability method. It first check whether list[I-1] value and list[I] value or not. If it is, it checks whether list[I-1] order is greater than list[I] order, and again if it is, that means our algorithm is not stable. So, it returns false. İf not, it returns true.

In Algorithms class, I implemented all the algorithms that we are asked to. I mainly use pseudo codes, which are provided in assignment pdf, and code them. I separate them another class to increase readability of my code.

In Main class, HOW_MANY_SIZE keep how many different size array there would be. I mean, if it is 4, by starting 2^6, it goes 4 steps further like 2^7 ,2^8 ,2^9 and 2^{10}. START_SIZE keeps starting point which is 2^6. END_SIZE keeps the value of highest array size. And the last one is HOW_MANY_SAMPLES keeps the repeating times of sorting experiment.

In here first, I check the stabilitiy of algorithms and keep the results in stable array, then I print it. I got true, false, false, true, false result for the algorithms gnome, stooge, comb, shaker and bitonic sort respectively. That means gnome and shaker sort are stable while the rests are not stable, which is correct. You can see the result by executing my code.

After that I corrected stabiliyCheck to false for performing sorting algorithms and calculating execution time on my arrays for 3 case. I know it has been asked only average and worst-case scenarios, but I wanted to implement all scenarios for better comparison.

My results looks pretty confusing in terminal screen so I decided to print them into a text file with a well-shaped format. Later, I copy my result into a csv file for plotting operations. I attached my .txt and .csv file which I use for drawing plots. You can check it in any necessary situation. In the experiment, I arrange HOW_MANY_SIZE as 8, which means I check for all 64-128-256-512-1024-2048-40-96-8192 size arrays. Besides, HOW_MANY_SAMPLES arranged as 32. Which means I execute the algorithm for 32 time to get more accurate result. I keep and print the duration of each run time, and also average of them.

Discussion Results

****I implement my code with starting highest array size, so array sizes are looked reversed. That is because, I realized that when I did this, I did increase the performance of my computer****

****There is also one issue with comb sort and bitonic sort. For those algorithms giving descending order list does not provide worst-case scenario. Yet, since we are applying comparison among these 5 algorithms giving same input for same case provides better accuracy****

Best Case:

Algorithms/n	8192	4096	2048	1024	512	256	128	64
Gnome	23816	9416	4031	2594	1437	719	466	353
Stooge	1,30951E+11	14534797491	1615405581	179463044	45775216	5105728	579928	258291
Comb	247366	114772	48903	21503	10025	4472	2072	988
Shaker	19422	9931	4041	2106	1197	678	459	316
Bitonic	1197063	531856	228166	100494	42378	18072	7688	3228

Average Case:

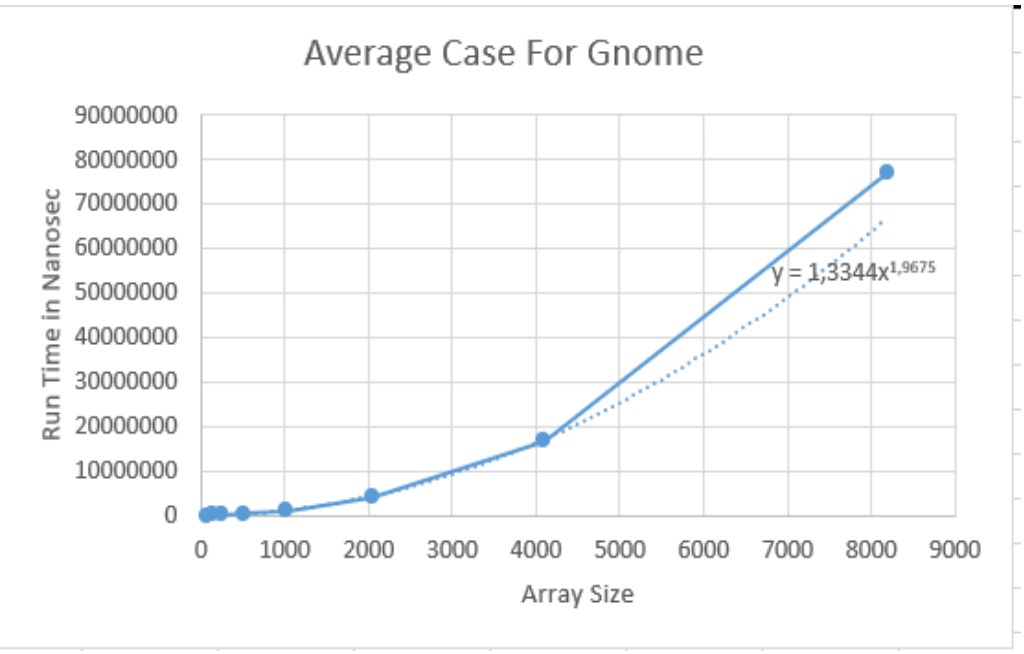
Algorithms/n	8192	4096	2048	1024	512	256	128	64
Gnome	77011269	16791353	4042266	997200	298684	66706	18703	5337
Stooge	131648038491	14555094909	1619063234	180723041	46015063	5153638	599122	262788
Comb	973638	395619	173572	72559	31628	14416	6538	2963
Shaker	82856750	17264369	4043625	1042666	290388	84772	24491	7638
Bitonic	2258509	1007016	457763	202109	84163	35606	15241	6216

Worst Case:

Algorithms/n	8192	4096	2048	1024	512	256	128	64
Gnome	144363856	31164897	8073922	2040825	509344	131150	33709	9231
Stooge	1,3113E+11	14567631297	1623118856	181581409	46178178	5195909	601059	264156
Comb	322684	142778	60731	25666	11694	5228	2375	1147
Shaker	91836756	22818803	5678059	1413772	355116	90831	24134	6334
Bitonic	1196509	525778	229706	99084	42303	17978	7644	3206

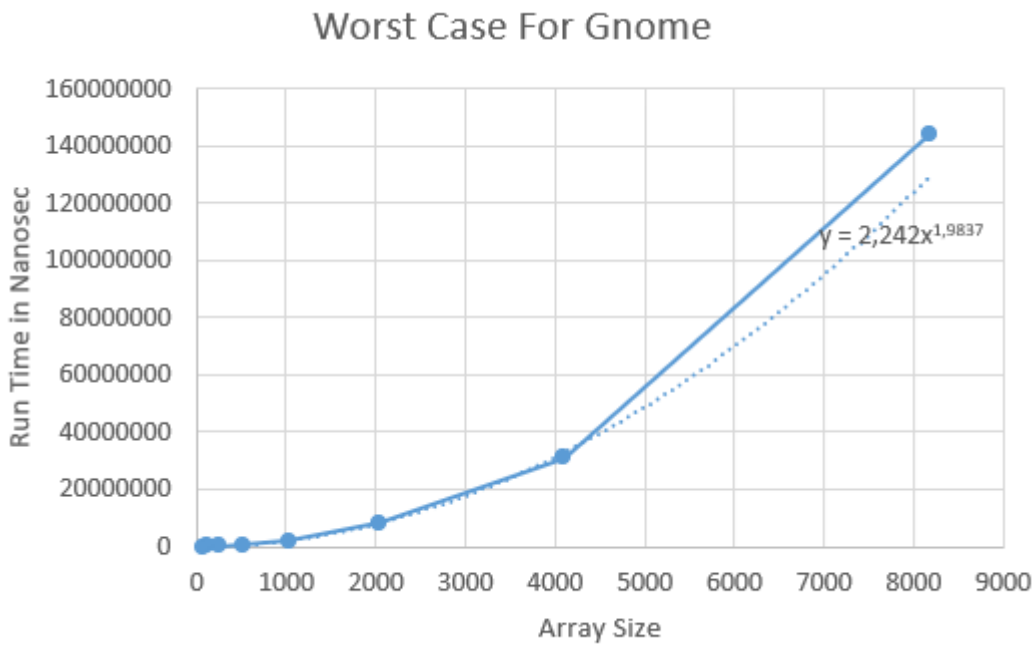
It can be clearly seen that, while array size decreasing, for each sorting algorithm execution times are decreasing also. By the way, time measurements are in nanoseconds.

Gnome Sort:



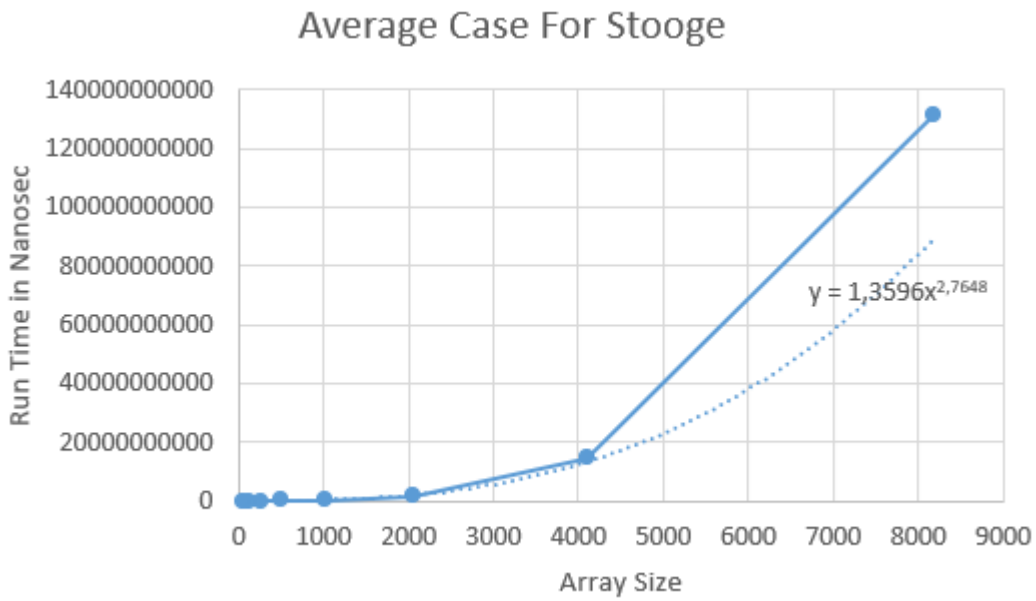
In below there are 2 charts for gnome sort. One of them is average case. It can be seen that how running tine increases due to increasing input size. I also add a trend line and equation for this line. It is written that $x^{1,9675}$, which is so close to x^2. And we have, $O(n^2)$ time complexity for gnome sort in average case. That means, our experiment has pretty good results.

Again, we have so close to the $O(n^2)$ time complexity in our equation line for worst case. So, again our implementation gives nearly same results.



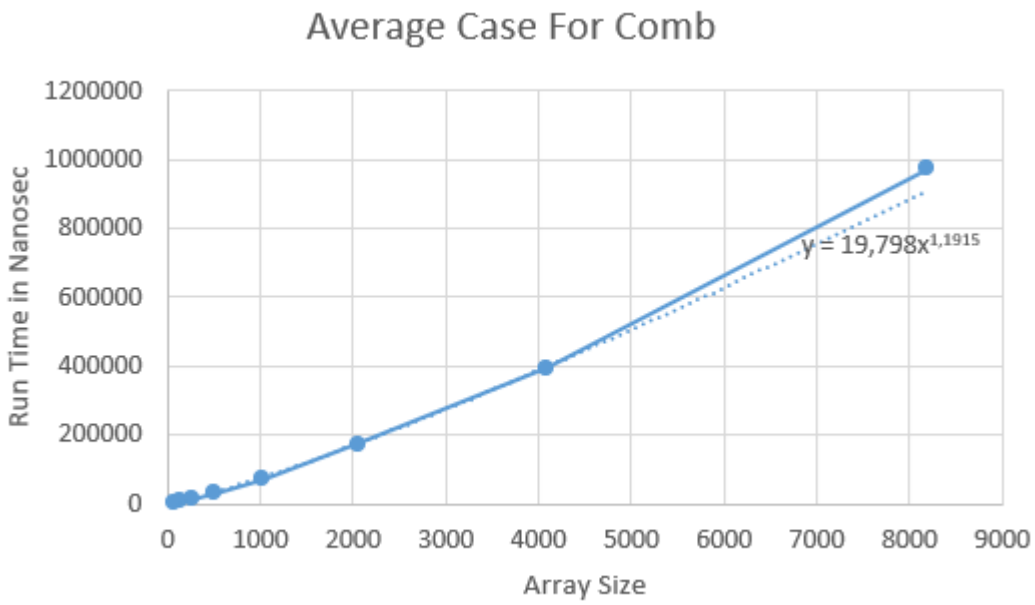
Stooge Sort:

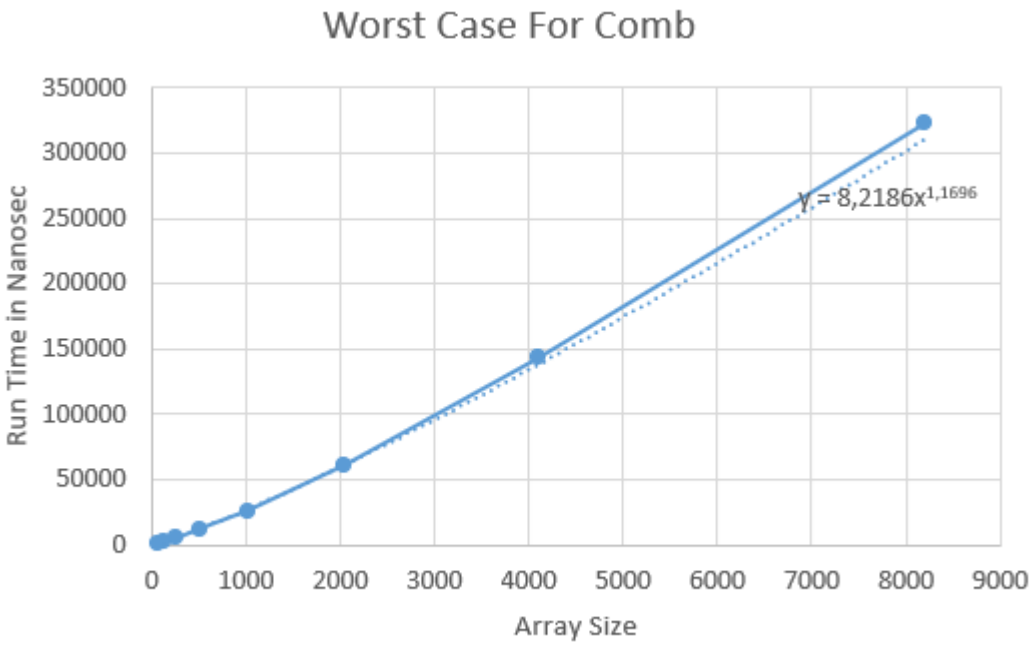
This is the slowest algorithm that applied in this assignment. Theoretically it has $O(n^{2,71})$ for both cases. When we are look at our trend line equation, we can observe that we got pretty close result with the theory.



Comb Sort:

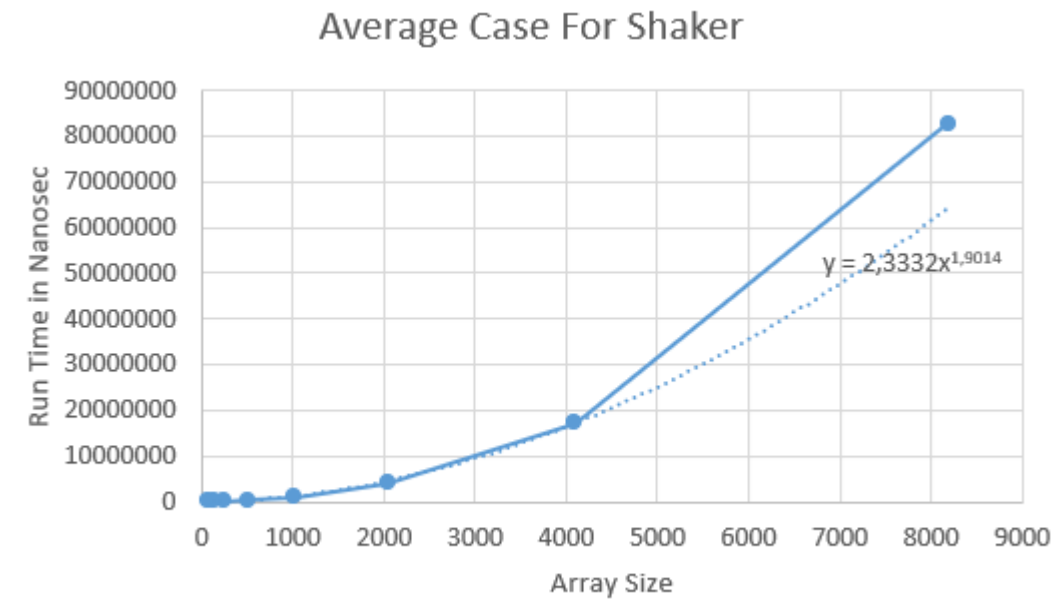
Theoretically its time complexity is $O(n^2)$ at worst case, $O(n^2/2p)$, where p is the number of increments. But as I said at the beginning of the discussion part, descending ordered list does not provide worst-case scenario for comb sort. Yet our average result is again close to theoretical result.

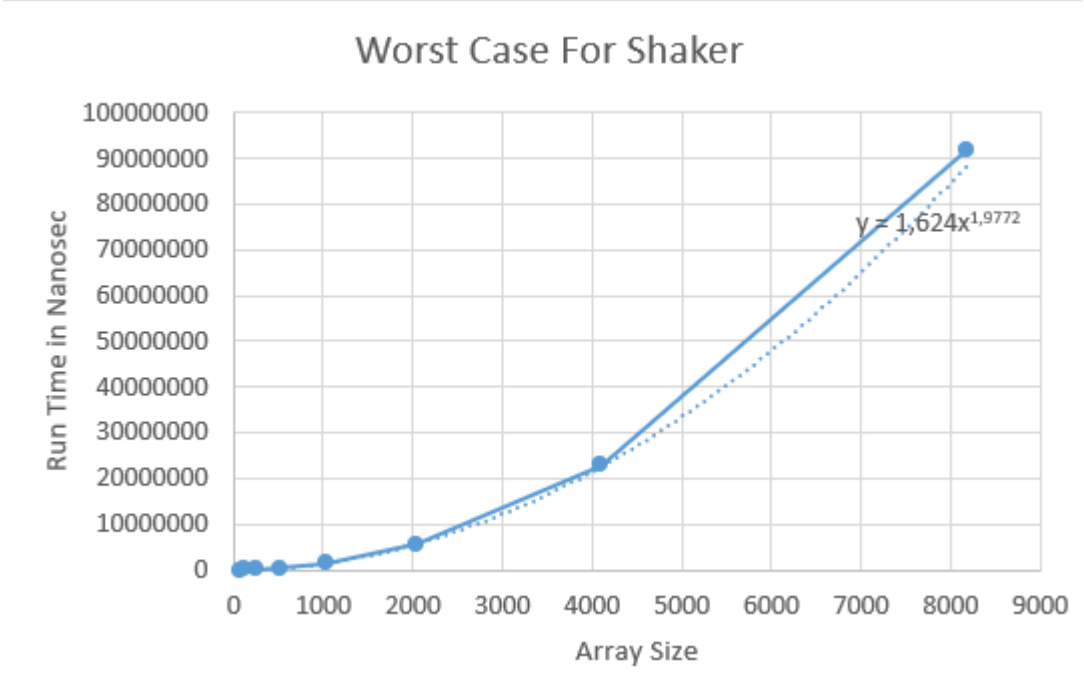




Shaker Sort:

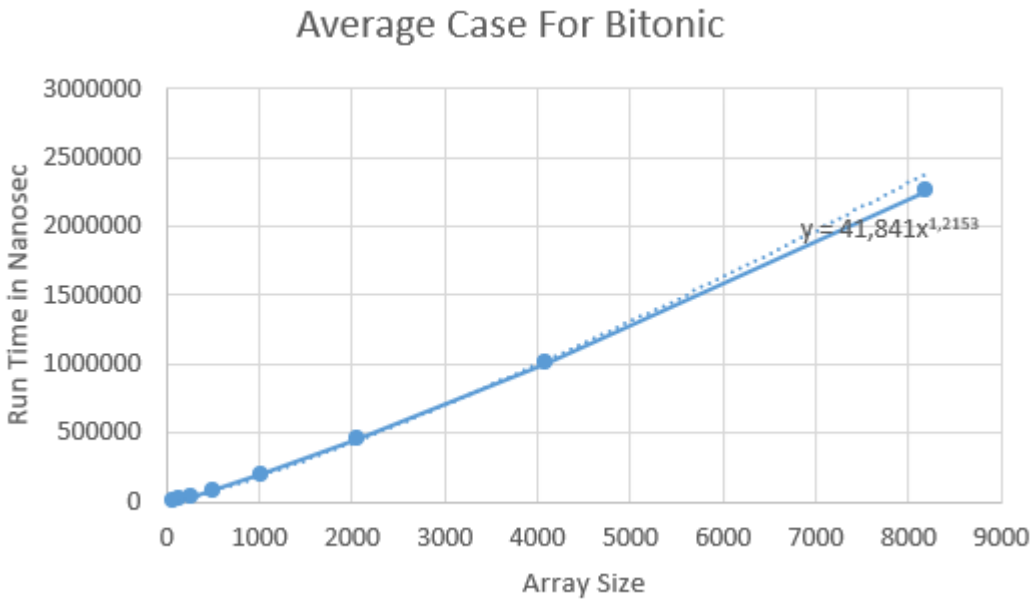
Theoretically its time complexity $O(n^2)$ at average and worst cases. When we check the trend line equations in both cases we are again so close to theoretical results. It is $n^{1,9014}$ for average and $n^{1,9772}$ for worst cases.

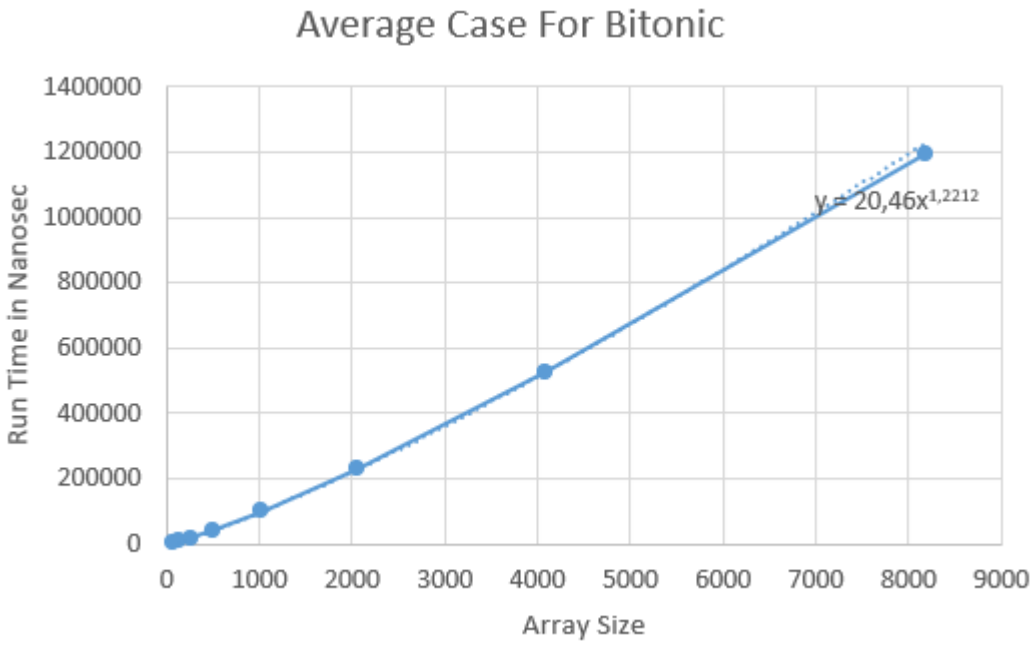




Bitonic Sort:

Theoretically it has O(log2n) time complexity. And we have good result here.





References

Liang, Daniel Y. Introduction to Java programming, 10th Edition, 2015. Pearson.

https://en.wikipedia.org/wiki/Gnome_sort

https://en.wikipedia.org/wiki/Stooge_sort

https://en.wikipedia.org/wiki/Comb_sort

https://en.wikipedia.org/wiki/Cocktail_shaker_sort

https://en.wikipedia.org/wiki/Bitonic_sorter#How_the_algorithm_works