Hacettepe University

Department of Computer Engineering

BBM204 Programming Lab.

Spring 2021

Programming Assignment 1

Name: Ömer Kaan

Surname: Vural

ID: 21987537

Subject: Analysis of Algorithms

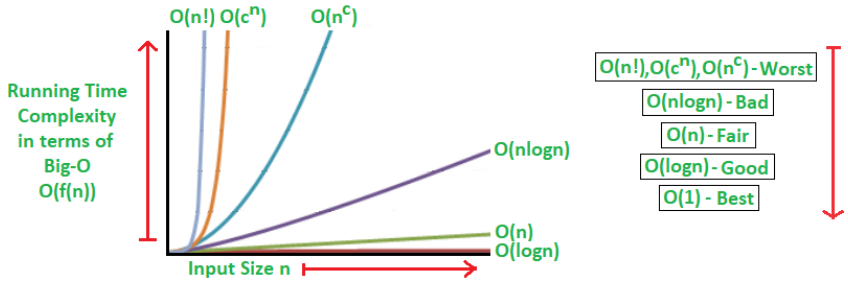
Submission Date: 10.03.2021

Deadline: 24.03.2021 23:59

Programming Language: Java

Introduction

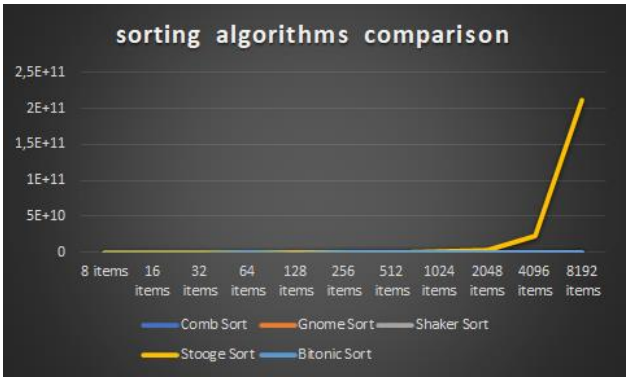
In this assignment, I am responsible for comparing some sorting algorithms in average case and worst case.. Given sorting algorithms are “Comb Sort, Gnome Sort, Shaker Sort, Stooge Sort, Bitonic Sort” algorithms. I have to compare running times of these algorithms with using different array size. In this way I can see that their complexities affect the running time through ascending array size and they can be plotted as a graph to see clearly. In a nutshell we should utilize different sorting algorithms in different cases which can be serve different fields that needs more searching operation or sorting operation or increase operation etc. Thus we can produce more efficient algorithms that have time and effort advantages. Lastly one of the our purpose is to observe asymptotic complexity of the sorting alogirthms.

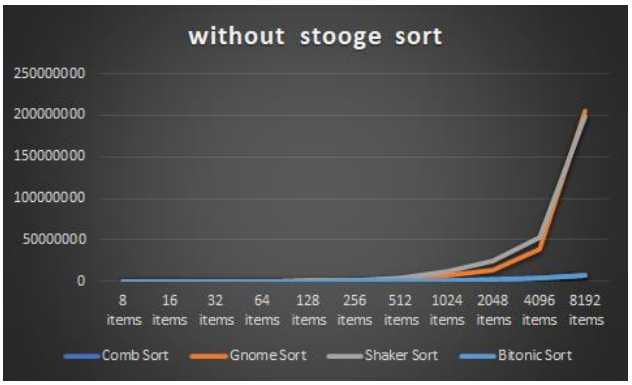


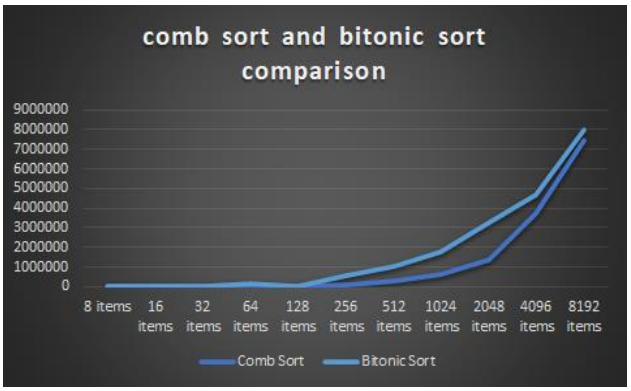
Comparing Sorting Algorithms

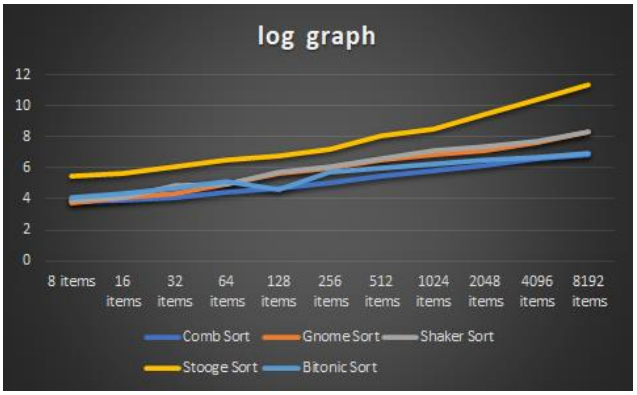
In this part we have four different graphs representing the relation between the input size and execution times. I’ve seperated stooge sort because of it’s irregularity and that it’s curve blocks to see other sorting algorithm curves clearly.

In the beginning we can simply state that all sorting algorithms are just about same running times. However, when the input size increase, all sorting algorithm’s running times start to increase exponentially. In this point, the crucial thing is that their exponential ratios are different . As seen in the first graph that exponantial increase of stooge sort’s running time are hugely larger than the other’s. Even we can’t clearly see the running times of other algorithms. So if we go to second graph we can see that the exponential increases of the running times of gnome and shaker sort algorithms are larger than comb sort and bitonic sort algorithms’s. After that, if we look at the third graph to compare combsort and bitonic sort, we can state that their running times react about same to increasing array size. Finally in the last graph I made a logarithm graph of the first graph to see comparison of all sorting algorithms in a different way and It proves above determinations that shows: stooge rt > gnome rt ≃ shaker rt > comb rt ≃ bitonic rt (r.t = running time).









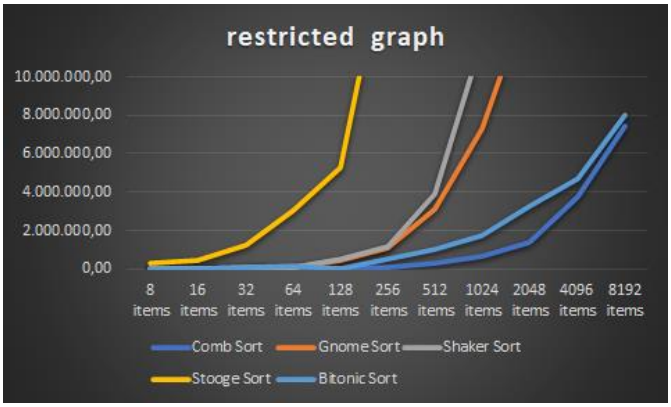
Strategy

In this assignment, I designed my experiment with using random numbers in changing array sizes to compare running times of sorting algorithms. I created an array for five different algorithms with 10000 bound that is enough my max size 8192. I've used Random class to do that. Also I've created an descending array to provide a worst case comparison. After all I created sorting algorithm objects an send arrays to them in order to measure running times with using nanoTime method. I placed starting time before the sorting function and ending time after the sorting function for every sorting algorithm.

General Analysis

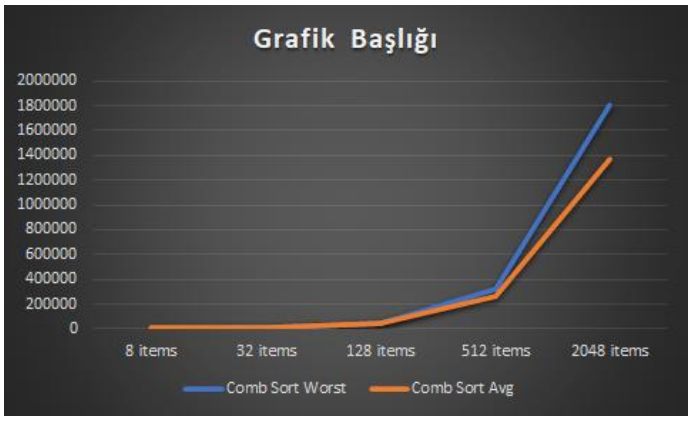
I've realized that the shaker sort and gnome sort algorithms are stable. What it means is stable algorithms don't do extra comparisons as not stable algoirithms. For Instance when some small number comes before a big number, stable algorithms leaves them as it is without doing any comparison. Regardless of the algorithms type, this state cost us more running time and process amount. I've obtained this conclusion with using an array of objects which have two attributes. I've sent these objects to array randomly and sorted that array by all sorting algorithms. Regardless of running times and logic of algorithms, stable ones have sorted the array without comparing the objects in which are already sorted position.

In terms of time complexity, we can sort the algorithms as Stooge Sort > Shaker Sort ≈ Gnome Sort > Comb Sort ≈ Bitonic Sort as we can see in the graph. If we examine the worst and best algorithm on this list, the last step of stooge sort, which causes a repetition of previous step have stooge sort make inefficient. However, Comb Sort algorithm does not make unnecessary comparison as Stooge, also it has a efficient constants which provides min step size almost perfect. You can see the time complexitiy information at algorithms own title in detail. In a nutshell, the time complexities on average case is " Bitonic Sort (log^2.n) < Comb Sort(n^2 / 2^p) < Gnome Sort (n^2) < Shaker Sort(n^2) < Stooge Sort(n^2,71).



COMB SORT

In the combsort algorithm the key point is a constant which is "1.3". As long as that our array size divided by the constant is smaller than 1 (it is called gap), we keep comparing the elements to the elements which is far from chosen index as the gap. Average time complexity of the combsort is O(n^2/2^p) and worst case time complexity is O(n^2). In the worst case, we have to find max comparison state which is descending order of an array. With the 8 size of an array, the worst case results is more than doubled of an average case. (avg:3100ns, worst:7700ns)

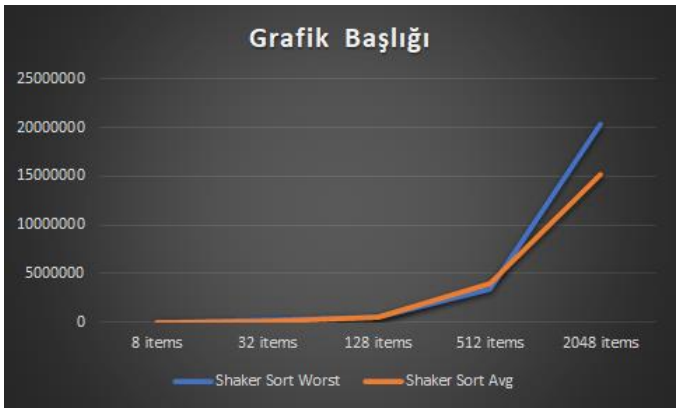


GNOME SORT

In the gnome sort, main logic is comparing the next element to the element in which position you are. If the next element is bigger than your current element, you just keep moving. But if the next element is smaller than yours, you swap them and keep moving to the backwards as long as finding the correct position of the swapped element. Average time complexity of the gnome sort is O(n^2) also the worst case time complexity is O(n^2) but there is a different coefficient on them. The worst case is clearly the descending order of an array which causes us maximum comparison. Finally the practical comparison results of the worst case and average case time complexity with 8 size of an array is avg: 3700ns, worst: 7300ns.

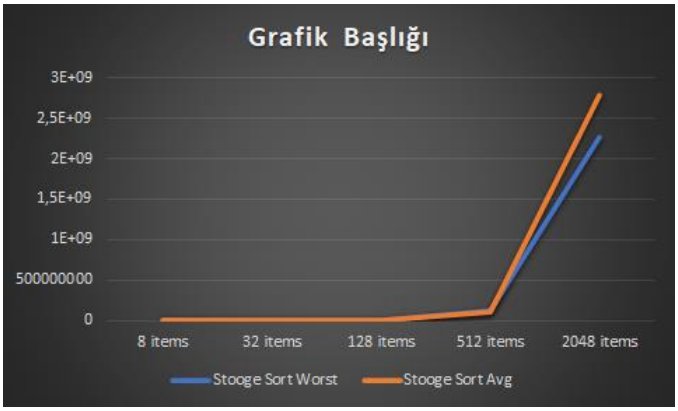
SHAKER SORT

In the shaker sort, we iterate the array one by one. While we doing this if our current position is smaller than the next element, our current position will be the bigger one. If not we swap the elements and keep moving until we reach end of a list. At the end of a list our largest element will be there through using this process. After we place largest element, we iterate opposite to place the smallest element of a list. This time if our current position is smaller than the next element we swap the elements and keep moving. If not we don't swap and keep iterating. Every time we decide a position of an element which is on starting side or ending side, we close their pointers to each other and when the starting pointer become bigger than ending pointer our sorting process ends. Also the worst case of shaker sort is the same as average case O(n^2) because we must keep iterating until starting and ending points change sides. So It does not important that our array partially sorted.



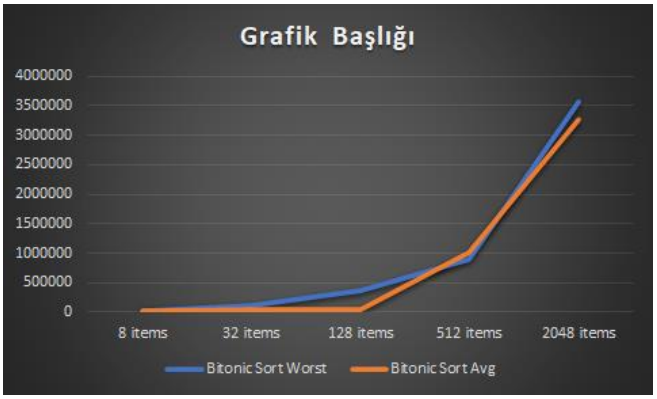
STOOGE SORT

In the stooge sort, we get our array's first 2/3 part and last 2/3 part to sort partially and recursively. Parts we got also sorting recursively with it's first 2/3 part and last 2/3 part and so on until the hole array is sorted. The crucial thing is in there is when this process ends the stooge sort algorithm controls is the first 2/3 part is still sorted after the last 2/3 part of a list sorted. Because last 2/3 part and first 2/3 part has common interval. So while the stooge sort algorithm sorting the last 2/3 part, perhaps there is also some changes happened first part. This hole recursive and unefficient process cost us a huge time complexity, which is the slowest one. Stooge sort average time and worst time complexity is $O(n^{\log 3/\log 1.5})$.



BITONIC SORT

In the bitonic sort, the first thing to be done is converting array to a bitonic sequence which is the first half of array is on ascending order and last half of array is on descending order. Every side by side duos make a bitonic sequence. If we split a half of an array we try to create a first half ascending and last half descending, with using that duos. The main idea in here is keep combining and replacing that duos until reach an ascending half of a bigger bitonic sequnce namely ascending sorted array. Also because it is about duos and dividing half operations the bitonic sort works on only power of two size arrays. For the worst case we can use opposite our main wish. If we want to sort the array ascending order we can create an array that it's first half descending and second half ascending. For Instance with using eight size of array average running time is 7600 ns altough worst case running time is 13000 ns. Further the bitonic sort average time complexity and worst time complexity is O(log^2.n).



WORST CASE STATUS

	8 items	32 items	128 items	512 items	2048 items
Comb Sort	3900	8400	41100	327000	1803200
Gnome Sort	4400	81600	590000	6994900	26098100
Shaker Sort	4000	164700	515500	2767500	22348400
Stooge Sort	261900	1104300	4754300	129144700	2266662200
Bitonic Sort	12400	108500	357200	901700	3564700

In the worst case condition for Gnome and Shaker sorting algorithms, descending order gives us worst values than the average case does. For Bitonic sorting alogrithm because the main thing is bitonic sequence I create an array that first half descending and other half ascending, which represent opposite of the ascending bitonic sequnce. Also I've tried to find a worst case for combsort but every random array gives me close results so I couldn't find a real worst case for combsort. Finally the worst case complexities:

- Comb Sort O(n^2)
- Gnome Sort O(n^2)
- Shaker Sort O(n^2)
- Stooge Sort O(n^2,71)
- Bitonic Sort O(log^2(n))

AVERAGE CASE STATUS

	8 items	16 items	32 items	64 items	128 items	256 items	512 items	1024 items	2048 items	4096 items	8192 items
Comb Sort	6600	8800	12300	25467	51066	106800	269367	659933	1366600	3729567	7467400
Gnome Sort	5800	11433	24100	82000	414967	1061400	3129100	7329667	14278667	38966400	205703833
Shaker Sort	8800	13300	69267	89933	525667	1141833	3928733	11888867	25278400	53430767	199606300
Stooge Sort	274867	445767	1215900	3070200	5311233	17160967	114730033	332633033	2784713167	23295815367	2,12E+11
Bitonic Sort	12600	22333	52533	131967	40900	529100	1009033	1752200	3255967	4701633	8010100

In average case status, we can clearly see that comb sort and bitonic sort algorithm more efficient than others. Because combsort constant provide us randomness in all case and that makes it adventageous in terms of finding worst order of an array for it. Also in Bitonic sorting algorithm, bitonic sequence provides us well randomness in addition our random array and saves us making redundant compares too. However, in stooge sort, we make redundant compares because trying sort the first part in the end of the process. Lastly the average case complexities:

- Comb Sort O(n^2) -Shaker Sort O(n^2) -Bitonic Sort O(log^2(n))
- Gnome Sort O(n^2) -Stooge Sort O(n^2,71)