HACETTEPE UNIVERSITY

Computer Science

A REPORT

ON

Programming Assignment 1

Analysis of Algorithm

SUBMITTED BY

Samet MOLLAOĞLU (21827704)

March 24, 2021

Chapter 1

PROBLEM DEFINITION

In this assignment, we will analyze given different algorithms and compare their running times with ideal time complexities. We measured running times of the algorithms which are given us and discuss about the results. Hereby end of this assignment, we will be able to understand which sorting algorithm is suitable for solving any situations efficiently and we will learn what stability is in sorting algorithms and its advantages and disadvantages.

Chapter 2

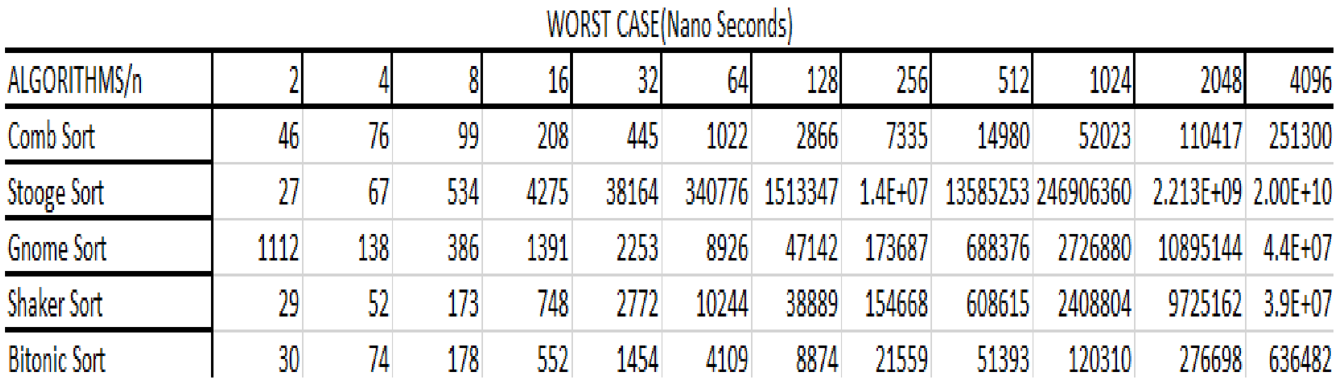
CODE IMPLEMENTING

In my main part of the code, I call the sort function to calculate with sorting algorithm classes and print to console average and worst cases results of desired sizes of lists on every given kind of the sorting algorithms. Also, I checked the stabilities of the sorting algorithms by using stable_print and stable_control functions. Actually, main part of the code is sort function. It takes 3 parameters list size, which sorting algorithm and sortingKinds means average or worst case will be calculated in there. For every time execution of this function is repeating in loop 1000 times to take the average of each elapsed time to get closer results. And also, I took advantage of enum variables to get rid of code redundancy.

Chapter 3

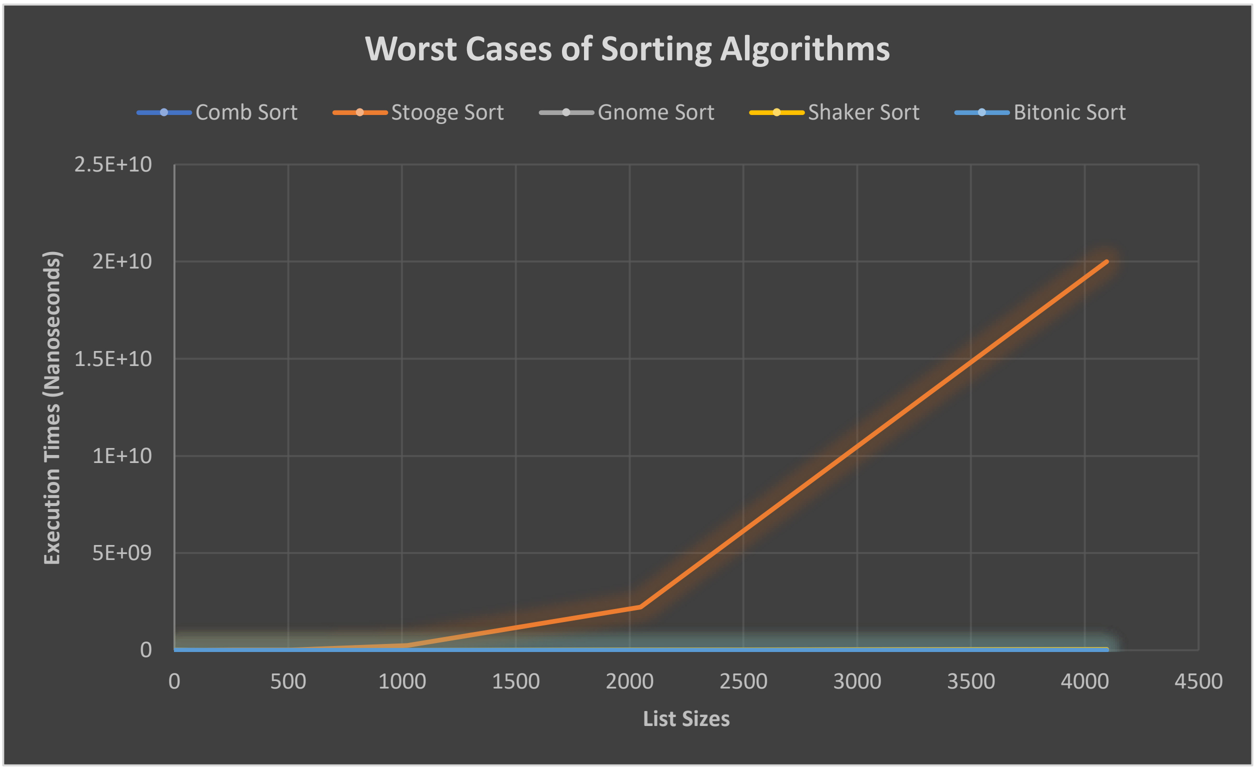
COMPLEXITY GRAPHS OF SORTING ALGORITHMS

3.1 *Worst Case Time Complexities*



WORST CASE(Nano Seconds)

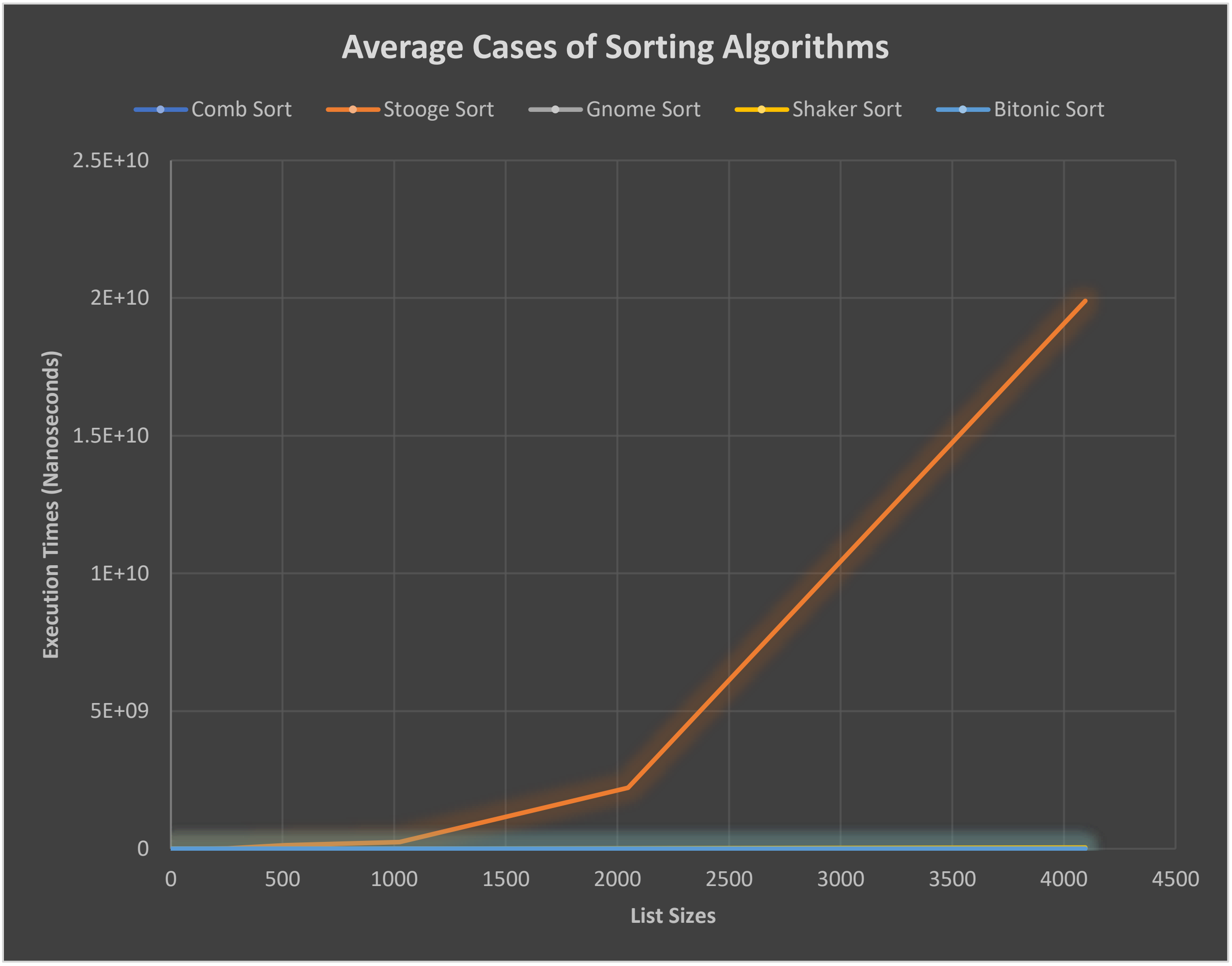
ALGORITHMS/n	2	4	8	16	32	64	128	256	512	1024	2048	4096
Comb Sort	46	76	99	208	445	1022	2866	7335	14980	52023	110417	251300
Stooge Sort	27	67	534	4275	38164	340776	1513347	1.4E+07	13585253	246906360	2.213E+09	2.00E+10
Gnome Sort	1112	138	386	1391	2253	8926	47142	173687	688376	2726880	10895144	4.4E+07
Shaker Sort	29	52	173	748	2772	10244	38889	154668	608615	2408804	9725162	3.9E+07
Bitonic Sort	30	74	178	552	1454	4109	8874	21559	51393	120310	276698	636482



3.2 *Average Case Time Complexities*

AVERAGE CASE(Nano Seconds)

ALGORITHMS/n	2	4	8	16	32	64	128	256	512	1024	2048	4096
Comb Sort	404	182	441	861	1649	3216	6709	15776	36781	86201	201468	439171
Stooge Sort	120	624	610	5460	38991	340636	1514668	1.4E+07	122422770	245731733	2.211E+09	1.99E+10
Gnome Sort	120	109	277	878	1721	5378	19987	98759	405435	1588470	6293861	2.5E+07
Shaker Sort	147	275	338	1283	4270	11505	43982	170310	606345	2335761	9968409	4.5E+07
Bitonic Sort	366	288	610	1650	4927	15217	33109	43794	79714	187153	433061	1001651

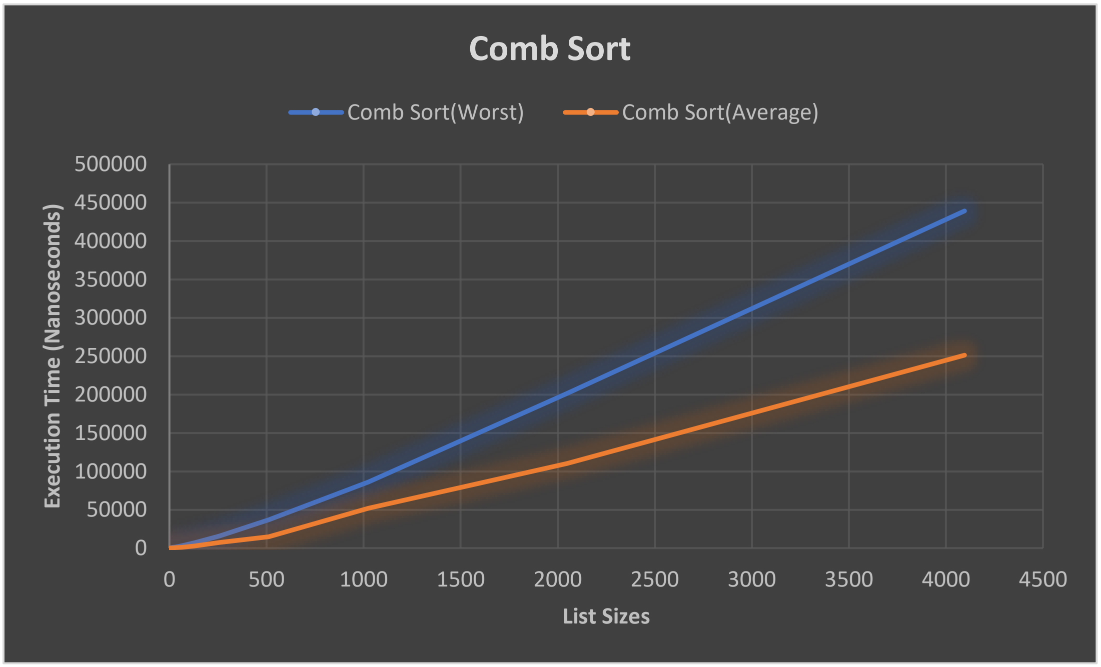


Chapter 4

DETAILED ANALYSES ABOUT SORTING ALGORITHMS

4.1 <u>Comb Sort Algorithm</u>

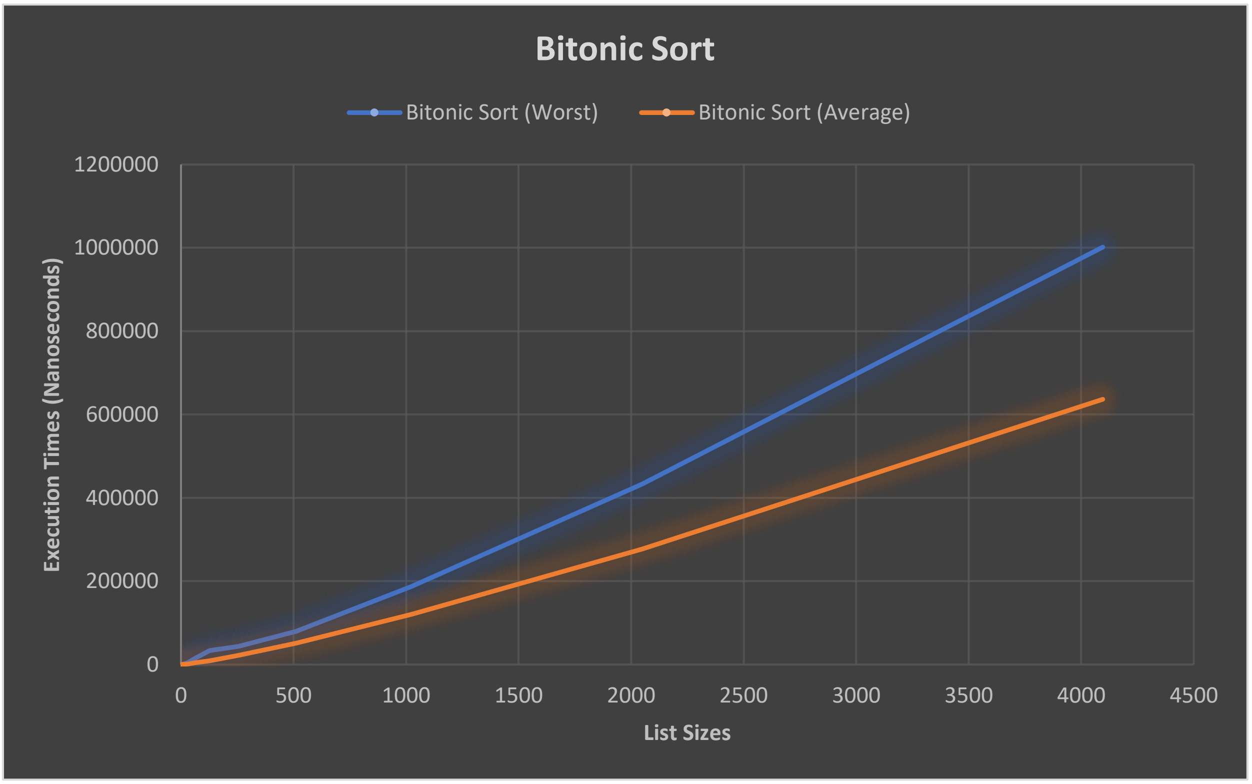
This sorting algorithm are the advanced form of the bubble sort. The difference between them is gap of bubble sort is always 1 but comb sort's gap is initially size of list and shrinks by 1.3 in every iteration until this gap variable reaches to the 1. So that the number of inversions counts of comb sort is less than the bubble sort. This algorithm is swapping the values that need to change, firstly determines a gap value and swap elements that have gap space between them throughout the list.



I defined random list and descending order list to sort. According to my observation descending order list sorting gives better result than random defined list sorting. The average and worst-case complexities are O(n^2) in comb sort. Proof of that complexities is a bit tricky but has been proved using a method based on Kolmogorov complexity. Also, space complexity of this algorithm is O(1) in all experiment.

4.2 <u>Bitonic Sort Algorithm</u>

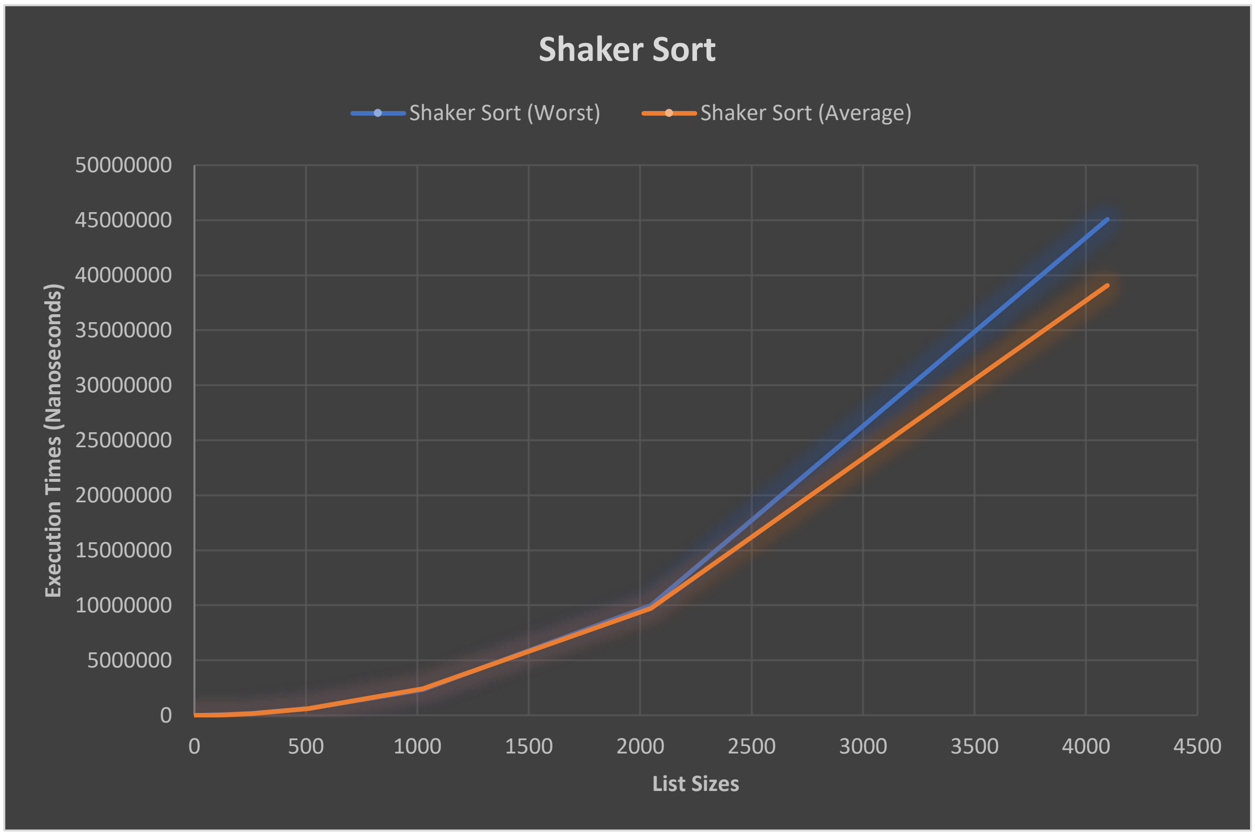
Main explanation of bitonic sort algorithm, firstly forming unsorted list to bitonic sequence means two sub-list in our main list that first half is ascending order, and second half is decreasing order. Then we can compare first element of the first half with first element of second half, then second element of first half with second element of second half and we swap the elements if an element of first half is bigger than element of second half and so on. We repeat this process in that sub-lists recursively, until the lengths of bitonic sequences are one. When we get bitonic sequences with size of one our list will be sorted.



Bitonic sort has O(log ^{2} n) time complexity for all cases. Also has O(nlog ^{2} n) space complexity. This algorithm does O(nlog ^{2} n) comparisons. An important thing about this sorting algorithm is although comparisons complexity is more than that other common algorithm, bitonic sort gives better performance to sort because elements of list are compared in main sequence means there is no need to define another list to comparing.

4.3 *Shaker Sort Algorithm*

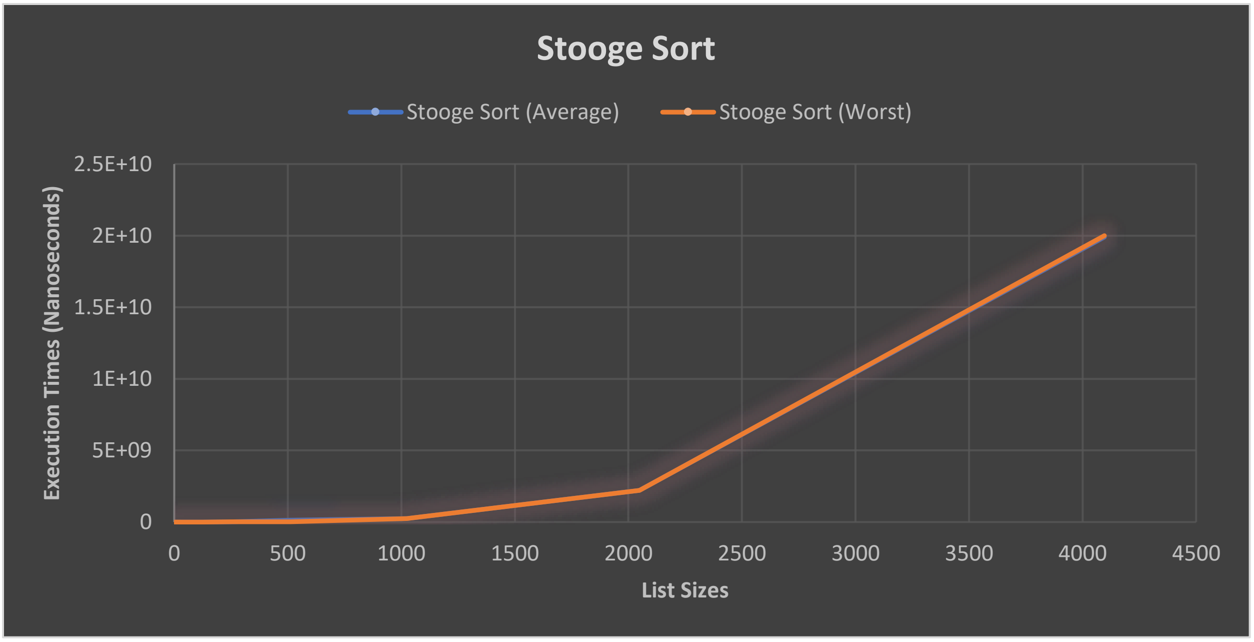
Shaker sort, it can also be called upper version of bubble sort. Because shaker sort traverses through both directions in given list but the bubble sort traverses through just one direction.



Time complexities in shaker sort, the worst and the average case are the same $O(n^2)$ because it traverses whole list to find correct position to every element. On the other hand, the best case of this sorting algorithm is O(1) because if all of the elements are in correct positions, there is no need to traverse the list more than one. According to my observations descending order created list sorting gives better result than random order created list. In additionally shaker sort has O(1) space complexity.

4.4 *Stooge Sort Algorithm*

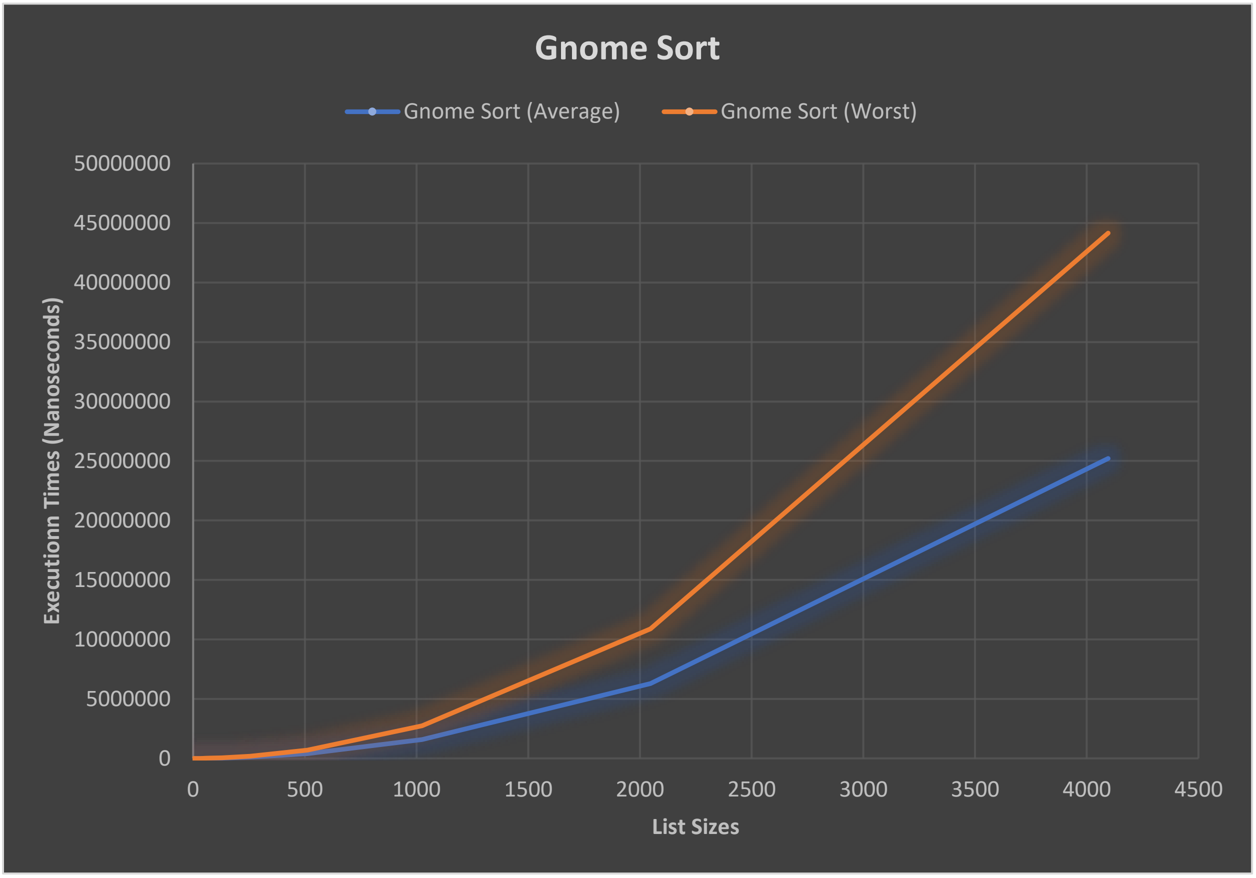
This sorting algorithm is a recursive sorting algorithm. It divides the list into two part after swap the element at first index with last element if necessary. These parts are overlapping means that first part is left 2/3 of the list and the other one is right 2/3 of list. Firstly, stooge sort process is applied to the first part of list, then same process applied to second part. Finally, again stooge sort process applied to first part of list to ensure. This sorting steps are applied recursively to the list until whole list will be fully sorted.



Stooge sort is known as an inefficient sorting algorithm. It has a bad time complexity according to other sorting algorithm. But difference of this algorithm from the others is that sorting overlapping sections in the list. All cases of this sorting algorithm are the same O(n ^ log (3) /log (1.5)). I tested this algorithm by using two differently created list. Firstly, I sort descending order list ,after that I sort another list of random numbers. End of this process I saw a negligible difference between their time complexity. In additionally stooge sort has O(n) worst case space complexity.

4.5 *Gnome Sort Algorithm*

Gnome sort algorithm is similar to the insertion sort and bubble sort. The main difference from insertion sort that we swap just adjacent items in gnome, but insertion sort is also can swap with non-adjacent element if necessary. Their similarity is that they iterate the list thoroughly one by one, also gnome sort has same similarity with bubble sort. The difference from the bubble sort is that when gnome sort found the element which in incorrect position, it continues until it finds the correct index for that element, but the bubble sort just swaps for one time.

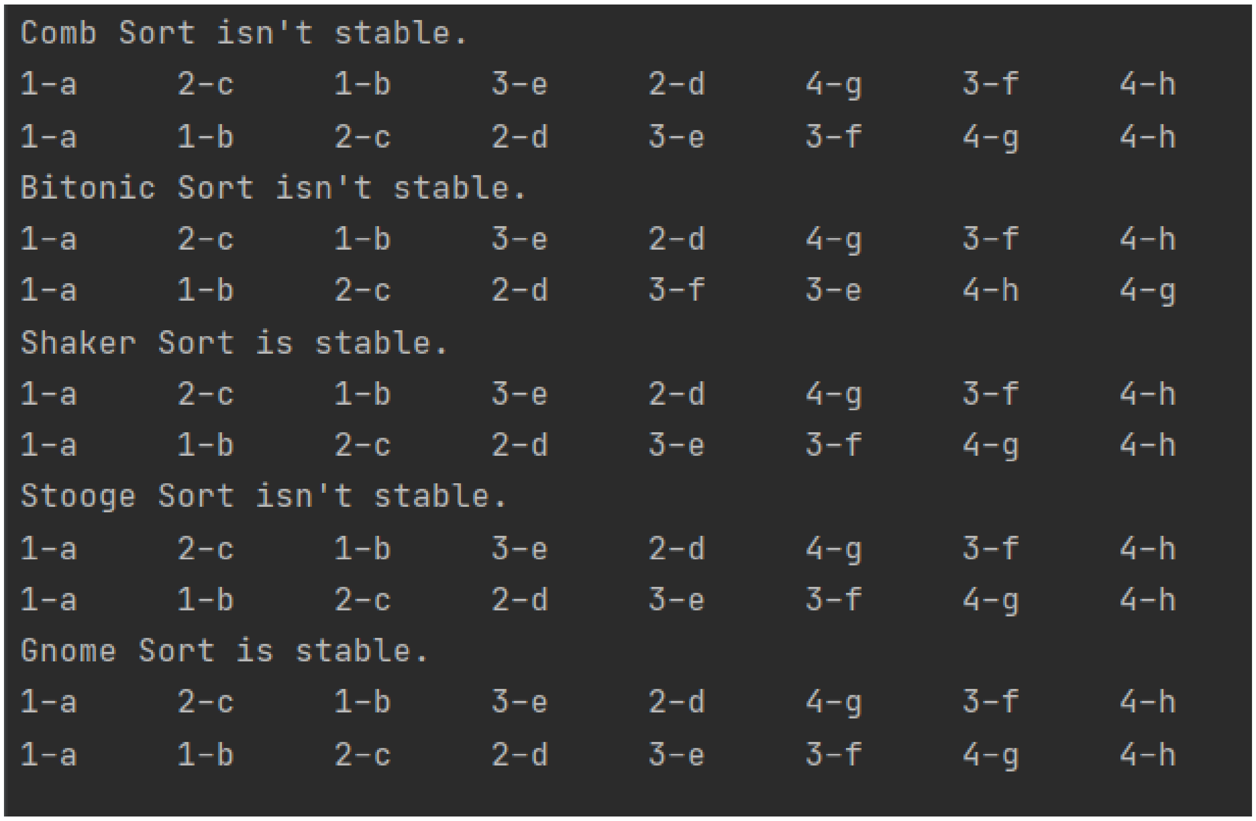


Gnome sort has O(n^2) to worst and average cases, O(n) to best case. I defined two list, randomly created one and ascending order one. At the end of the sorts, I saw that the sort of the list of numbers in descending order gives better results than list of randomly ordered numbers.

Chapter 5

Observations About Stability

Stable sorting algorithms retains the order of identical items in end of the sorting. For example, you have a list of the objects and each object has two attributes to be sorted separately means you have to sort that list by with both attributes, respectively. When you finished first sort your list will be sorted by first attribute after that end of the sort by second attribute, you will see that list is sorted by both attributes but in unstable sorting algorithms when the code is sorting by the second attribute, it may break the order of the first attribute. To control stability of algorithms, firstly I created unsorted list and pass it as a parameter of stable_control function. In this function I sort that list by given type of sort algorithms and print the output by stable_print function to see the result. Thereby I get an idea about advantages and disadvantages of stability. According to my inferences from my output, shaker and gnome sorting algorithms are stable and the other one's comb, bitonic and stooge sorts are unstable so order of identical items can changeable.



```
Comb Sort isn't stable.
1-a     2-c     1-b     3-e     2-d     4-g     3-f     4-h
1-a     1-b     2-c     2-d     3-e     3-f     4-g     4-h
Bitonic Sort isn't stable.
1-a     2-c     1-b     3-e     2-d     4-g     3-f     4-h
1-a     1-b     2-c     2-d     3-f     3-e     4-h     4-g
Shaker Sort is stable.
1-a     2-c     1-b     3-e     2-d     4-g     3-f     4-h
1-a     1-b     2-c     2-d     3-e     3-f     4-g     4-h
Stooge Sort isn't stable.
1-a     2-c     1-b     3-e     2-d     4-g     3-f     4-h
1-a     1-b     2-c     2-d     3-e     3-f     4-g     4-h
Gnome Sort is stable.
1-a     2-c     1-b     3-e     2-d     4-g     3-f     4-h
1-a     1-b     2-c     2-d     3-e     3-f     4-g     4-h
```