# HACETTEPE UNIVERSITY

## Department of Computer Engineering

BBM204

Programming Lab.

Programming Assignment 1

| Date | Name | Number |
|---|---|---|
| 24/03/2021 | Murat Çelik | 21827263 |

# Analysis of Algorithms

## 1.Introduction

In computer science, a sorting algorithm is an algorithm that puts elements of a list in a certain order. In computer science, the computational complexity or simply complexity of an algorithm is the amount of resources required to run it. [1]

The number of (machine) instructions which a program executes during its running time is called its time complexity in computer science. This number depends primarily on the size of the program's input, that is approximately on the number of the strings to be sorted (and their length) and the algorithm used. So approximately, the time complexity of the program "sort an array of n strings by minimum search" is described by the expression c*n^2.

"c" is a constant which depends on the programming language used, on the quality of the compiler or interpreter, on the CPU, on the size of the main memory and the access time to it.[2]

Generally, we perform the following types of analysis;

- Worst-case− The maximum number of steps taken on any instance of size a.
- Best-case− The minimum number of steps taken on any instance of size a.
- Average case− An average number of steps taken on any instance of size a.

[1] https://en.wikipedia.org/wiki/Sorting_algorithm

[2] https://www.leda-tutorial.org/en/official/ch02s02s03.html

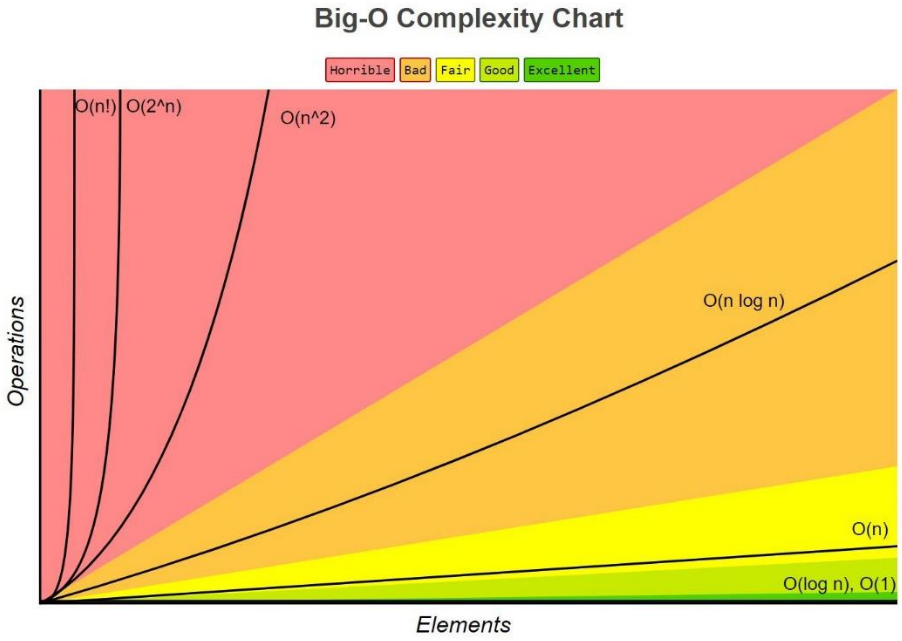


*Figure 1 Time Complexity Graph*

# 2.Problem

In this experiment, 5 sequencing algorithms will be analyzed:

- Comb Sort Algorithm
- Gnome Sort Algorithm
- Shaker Sort Algorithm
- Stooge Sort Algorithm
- Bitonic Sort Algorithm

1) A brief information will be given about algorithms.
2) The time we obtained will be given according to the number of inputs.
3) It will be proved whether it is stable or not.
4) The data will be visualized with graphics.
5) The final step will be the results and benefits from this experiment.

# 3. Sorting Algorithm

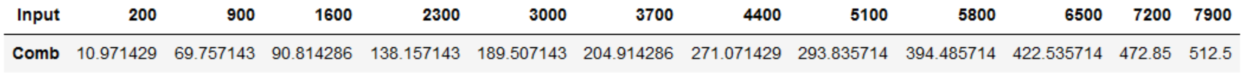
## *3.1 Comb Sort Algorithm*

Comb sort is a relatively simple sorting algorithm originally designed by Włodzimierz Dobosiewicz and Artur Borowy in 1980, later rediscovered by Stephen Lacey and Richard Box in 1991. Comb Sort improves on Bubble Sort by using gap of size more than 1. The gap starts with a large value and shrinks by a factor of 1.3 in every iteration until it reaches the value 1.[3]

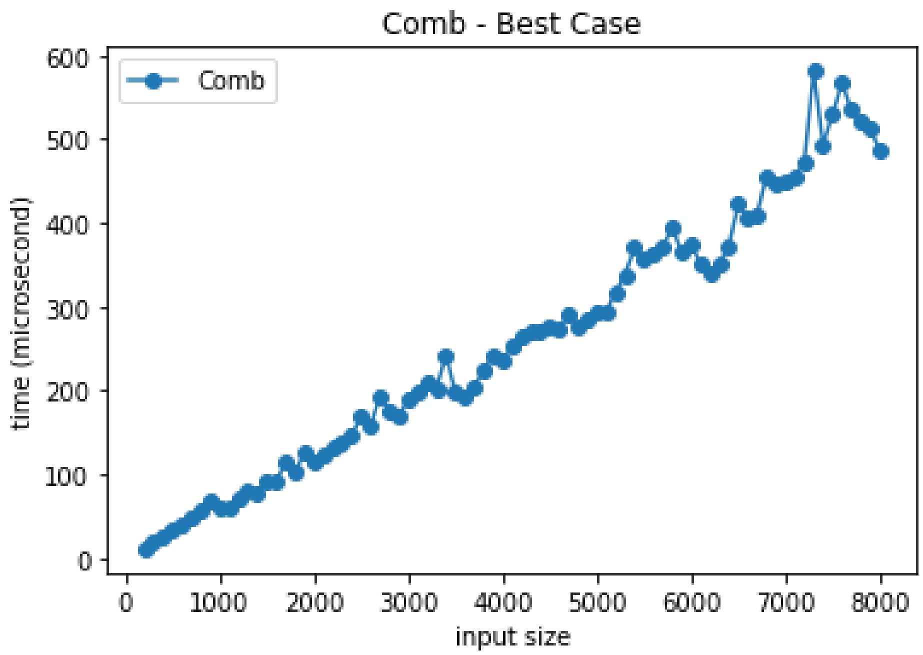
- Worst-case performance O ( n^2 )
- Best-case performance Θ ( n * log(n) )
- Average performance Ω ( n^2 / 2^p ) ,where p is the number of increments
  - Space Complexity O(1)

### Data: (Unit microsecond)

(Sorted Array) => Best Case

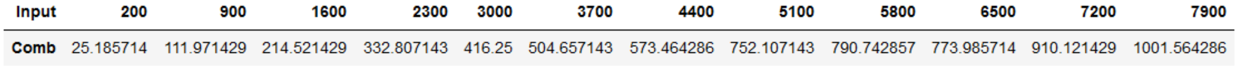


| Input | 200 | 900 | 1600 | 2300 | 3000 | 3700 | 4400 | 5100 | 5800 | 6500 | 7200 | 7900 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Comb** | 10.971429 | 69.757143 | 90.814286 | 138.157143 | 189.507143 | 204.914286 | 271.071429 | 293.835714 | 394.485714 | 422.535714 | 472.85 | 512.5 |

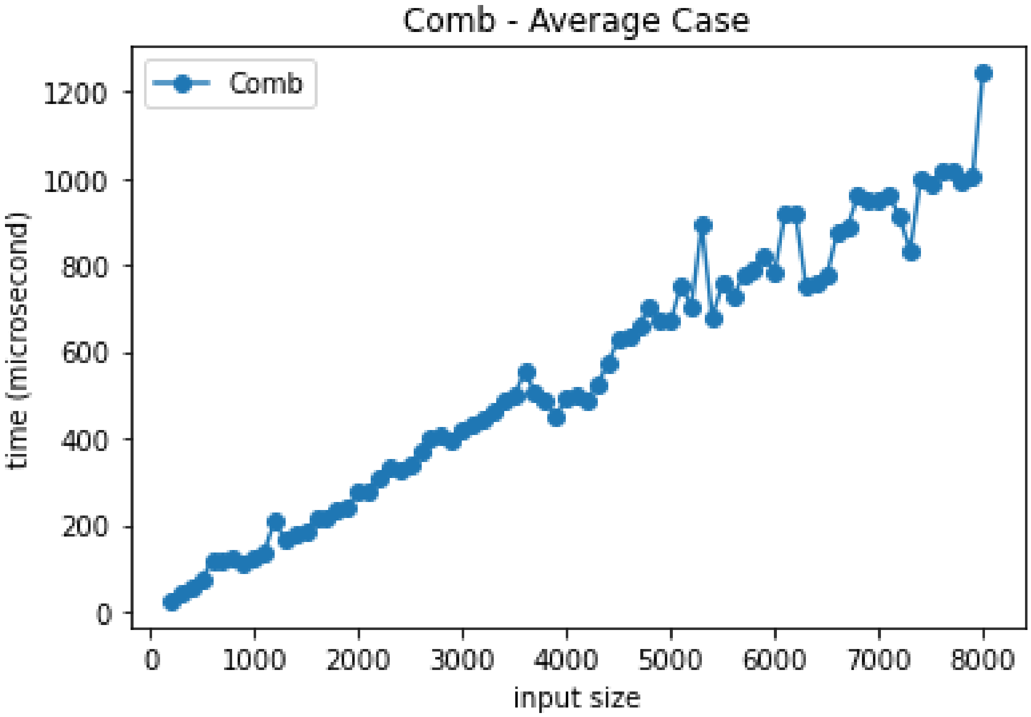


[3] https://en.wikipedia.org/wiki/Comb_sort

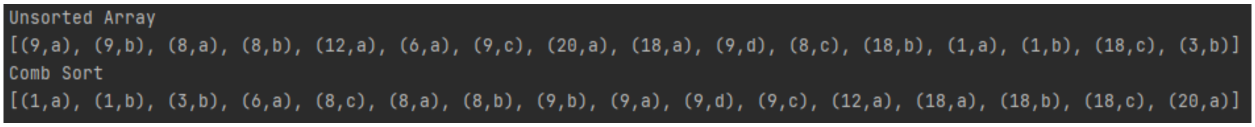
## (Unsorted Array) => Average Case



| Input | 200 | 900 | 1600 | 2300 | 3000 | 3700 | 4400 | 5100 | 5800 | 6500 | 7200 | 7900 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Comb** | 25.185714 | 111.971429 | 214.521429 | 332.807143 | 416.25 | 504.657143 | 573.464286 | 752.107143 | 790.742857 | 773.985714 | 910.121429 | 1001.564286 |



## Stable: No



```
Unsorted Array
[(9,a), (9,b), (8,a), (8,b), (12,a), (6,a), (9,c), (20,a), (18,a), (9,d), (8,c), (18,b), (1,a), (1,b), (18,c), (3,b)]
Comb Sort
[(1,a), (1,b), (3,b), (6,a), (8,c), (8,a), (8,b), (9,b), (9,a), (9,d), (9,c), (12,a), (18,a), (18,b), (18,c), (20,a)]
```

## Note:

When we look at Figure 1, it can be said that my average graph looks like the "n * log (n)" graph. Again, this can be said for our best graph. It surprised me that the Bubble sort algorithm was used for this algorithm and the gap value was determined to be 1.3. I observed that the experiment result was not stable. I can say that it is efficient compared to others. In addition, space complexity is efficient as memory usage is low.

## *3.2 Gnome Sort Algorithm*

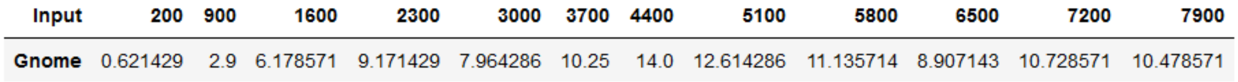
Gnome sort (dubbed stupid sort) is a sorting algorithm originally proposed by Iranian computer scientist Hamid Sarbazi-Azad (professor of Computer Science at Sharif University of Technology) in 2000.[4]

The gnome sort is a sorting algorithm which is similar to insertion sort in that it works with one item at a time but gets the item to the proper place by a series of swaps, similar to a bubble sort. It is conceptually simple, requiring no nested loops. [5]

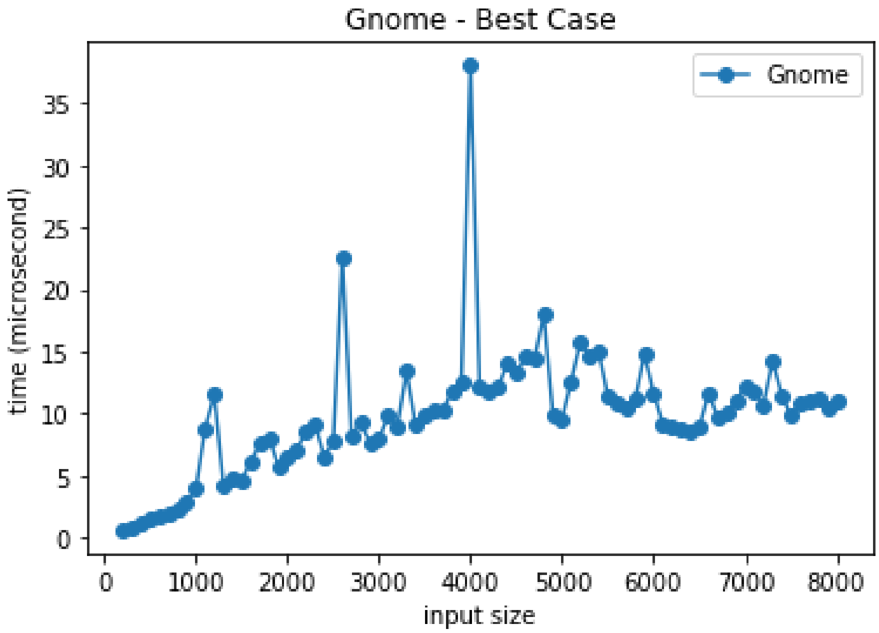
- Worst-case performance O ( n^2 )
- Best-case performance Θ ( n )
- Average performance Ω ( n^2 )
    - ○ Space Complexity O(1)

Data: (Unit microsecond)

(Sorted Array) => Best Case



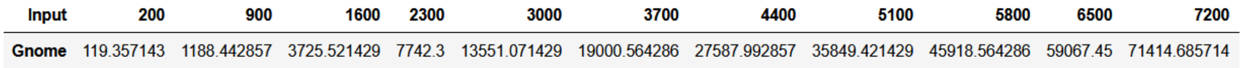
| Input | 200 | 900 | 1600 | 2300 | 3000 | 3700 | 4400 | 5100 | 5800 | 6500 | 7200 | 7900 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Gnome | 0.621429 | 2.9 | 6.178571 | 9.171429 | 7.964286 | 10.25 | 14.0 | 12.614286 | 11.135714 | 8.907143 | 10.728571 | 10.478571 |



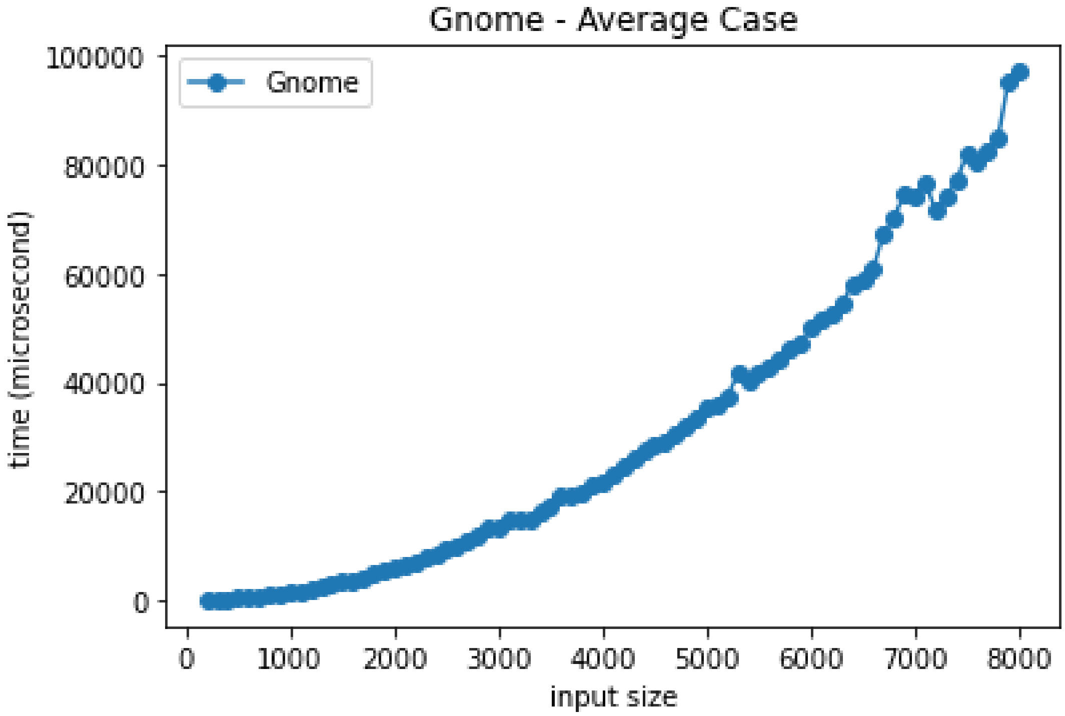
[4] https://en.wikipedia.org/wiki/Gnome_sort

[5] https://www.geeksforgeeks.org/gnome-sort-a-stupid-one/

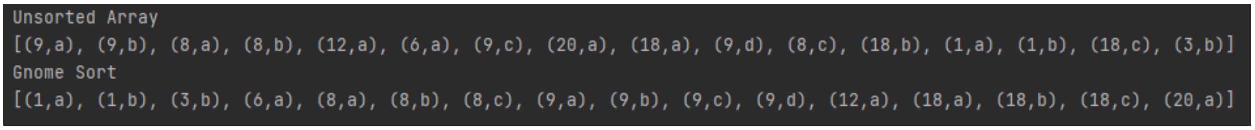
(Unsorted Array) => Average Case



| Input | 200 | 900 | 1600 | 2300 | 3000 | 3700 | 4400 | 5100 | 5800 | 6500 | 7200 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Gnome | 119.357143 | 1188.442857 | 3725.521429 | 7742.3 | 13551.071429 | 19000.564286 | 27587.992857 | 35849.421429 | 45918.564286 | 59067.45 | 71414.685714 |



## Stable: Yes



```
Unsorted Array
[(9,a), (9,b), (8,a), (8,b), (12,a), (6,a), (9,c), (20,a), (18,a), (9,d), (8,c), (18,b), (1,a), (1,b), (18,c), (3,b)]
Gnome Sort
[(1,a), (1,b), (3,b), (6,a), (8,a), (8,b), (8,c), (9,a), (9,b), (9,c), (9,d), (12,a), (18,a), (18,b), (18,c), (20,a)]
```

## Note:

When we look at Figure 1, it can be said that the average graph looks like "n * log (n)" or "n ^ 2" graph. It can be said that there is "n" graph for our Best graph. I would say it has a simple algorithm. I observed that the experiment result was stable. I can say that it is moderately efficient compared to others. I use it for a simple series, but not for a mixed series. It can be said that it is efficient with O (1) in terms of memory.

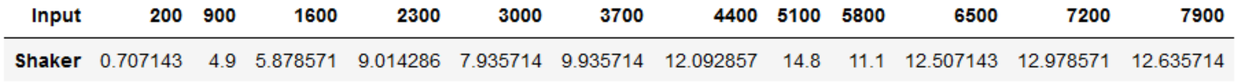
## *3.3 Shaker Sort Algorithm*

Cocktail shaker sort, also known as bidirectional bubble sort, cocktail sort, shaker sort (which can also refer to a variant of selection sort), ripple sort, shuffle sort, or shuttle sort, is an extension of bubble sort. The algorithm extends bubble sort by operating in two directions. [6]

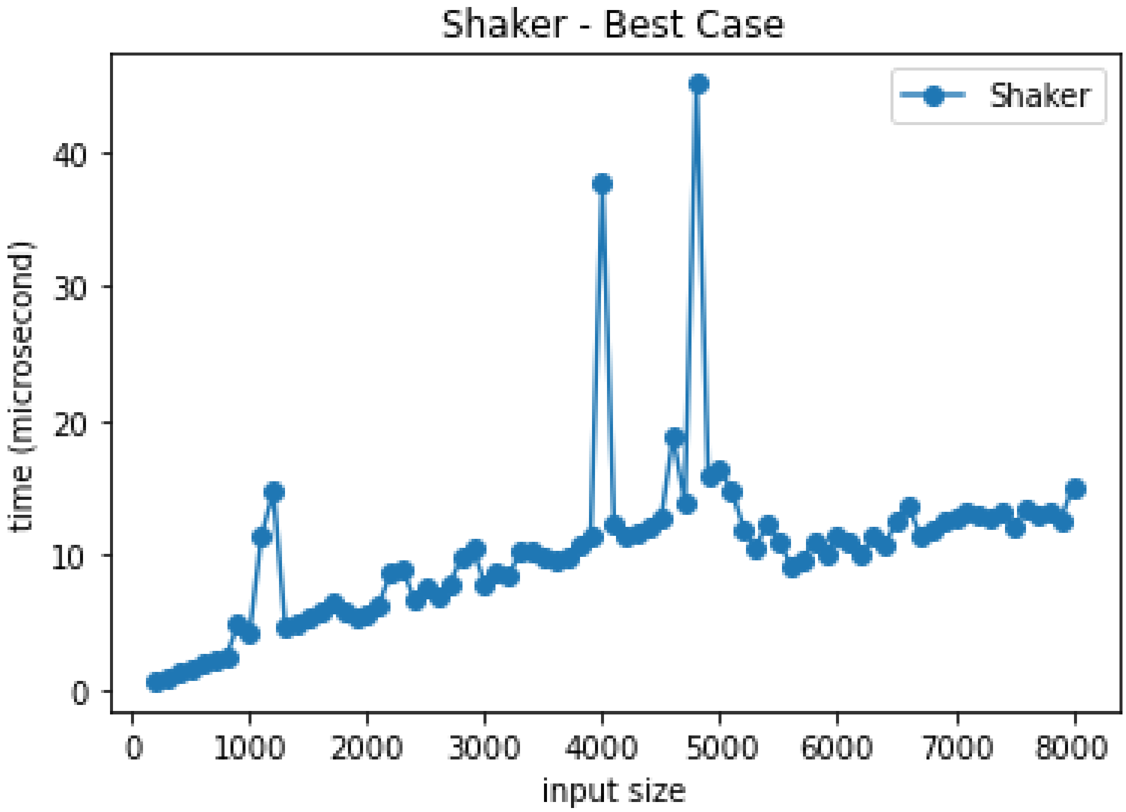
- Worst-case performance O ( n^2 )
- Best-case performance Θ ( n )
- Average performance Ω ( n^2 )
    - ○ Space Complexity O(1)

## Data: (Unit microsecond)

(Sorted Array) => Best Case

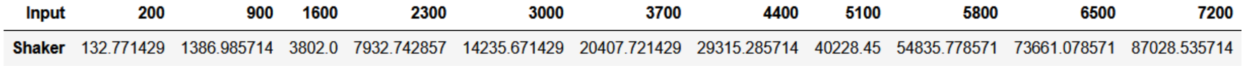


| Input | 200 | 900 | 1600 | 2300 | 3000 | 3700 | 4400 | 5100 | 5800 | 6500 | 7200 | 7900 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Shaker | 0.707143 | 4.9 | 5.878571 | 9.014286 | 7.935714 | 9.935714 | 12.092857 | 14.8 | 11.1 | 12.507143 | 12.978571 | 12.635714 |

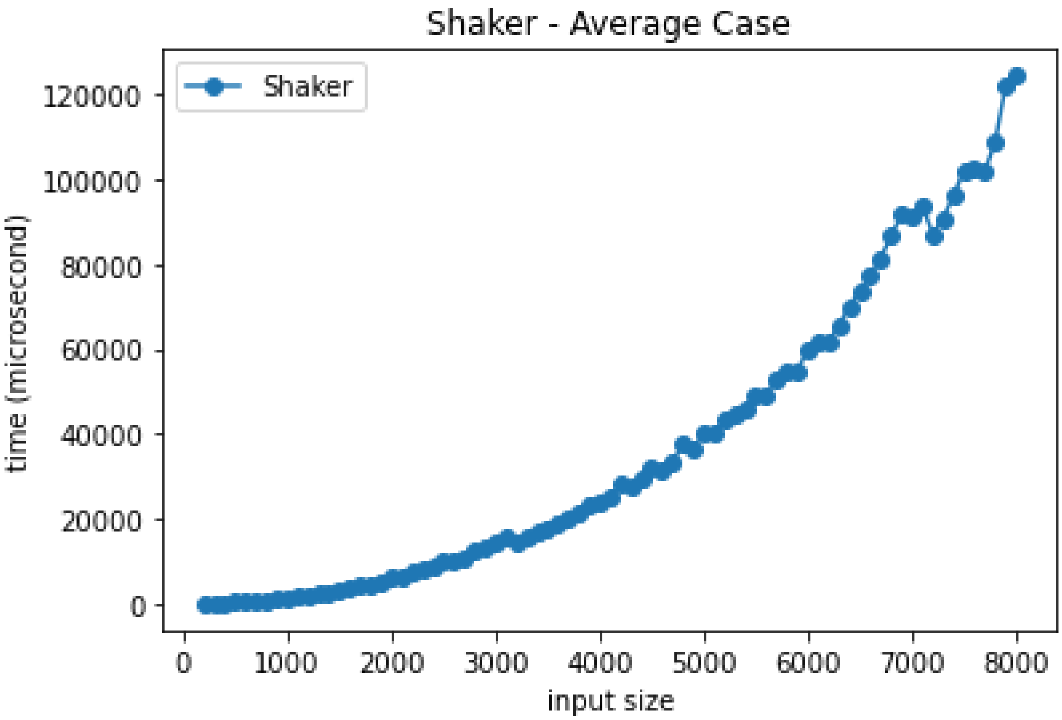


[6] https://en.wikipedia.org/wiki/Cocktail_shaker_sort

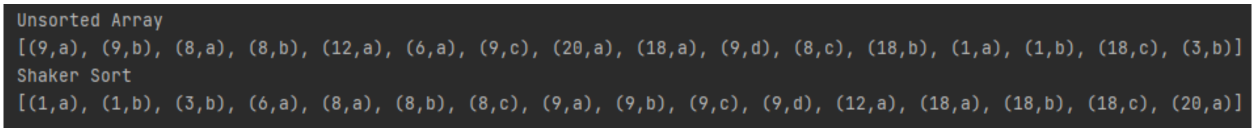
## (Unsorted Array) => Average Case



| Input | 200 | 900 | 1600 | 2300 | 3000 | 3700 | 4400 | 5100 | 5800 | 6500 | 7200 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Shaker | 132.771429 | 1386.985714 | 3802.0 | 7932.742857 | 14235.671429 | 20407.721429 | 29315.285714 | 40228.45 | 54835.778571 | 73661.078571 | 87028.535714 |



## Stable:  Yes



```
Unsorted Array
[(9,a), (9,b), (8,a), (8,b), (12,a), (6,a), (9,c), (20,a), (18,a), (9,d), (8,c), (18,b), (1,a), (1,b), (18,c), (3,b)]
Shaker Sort
[(1,a), (1,b), (3,b), (6,a), (8,a), (8,b), (8,c), (9,a), (9,b), (9,c), (9,d), (12,a), (18,a), (18,b), (18,c), (20,a)]
```

## Note:

When we look at Figure 1, it can be said that the average graph looks like "n * log (n)" or "n ^ 2" graph. It can be said that there is a "n" graph for our Best graph. This algorithm is an extension of bubble sort. Effective for small values at the end of the list. I observed that the experiment result was stable. I can say that it is medium-high efficient compared to others. I might prefer bubble sort instead. I would say that the algorithm is memory-efficient.

## *3.4 Stooge Sort Algorithm*

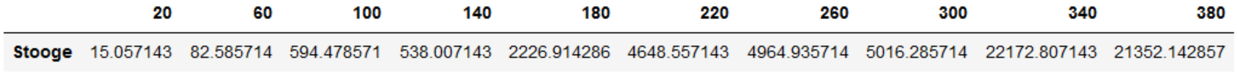
Stooge sort is a recursive sorting algorithm. The algorithm is defined as follows:

- If the value at the start is larger than the value at the end, swap them.
- If there are 3 or more elements in the list, then:
  - Stooge sort the initial 2/3 of the list
  - Stooge sort the final 2/3 of the list
  - Stooge sort the initial 2/3 of the list again[7]

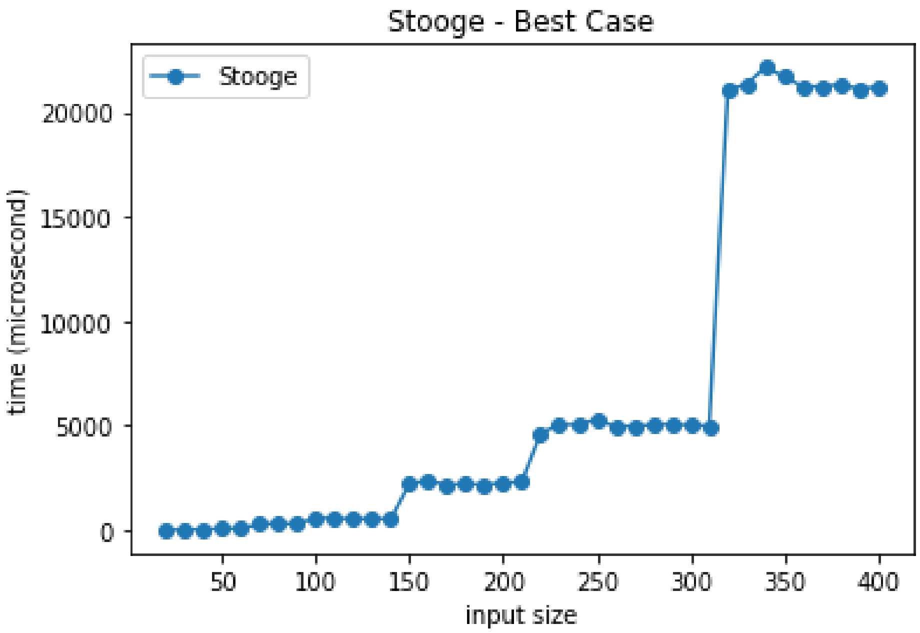
- Worst-case performance O(n^(log3/log1.5)) / O(n^2.709)
  - Space Complexity O(n)

# Data: (Unit microsecond)

(Sorted Array) => Best Case

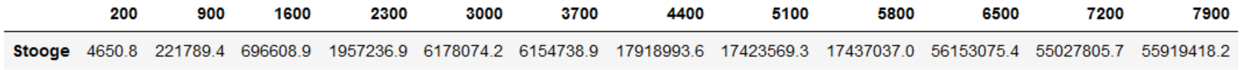


| | 20 | 60 | 100 | 140 | 180 | 220 | 260 | 300 | 340 | 380 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Stooge** | 15.057143 | 82.585714 | 594.478571 | 538.007143 | 2226.914286 | 4648.557143 | 4964.935714 | 5016.285714 | 22172.807143 | 21352.142857 |

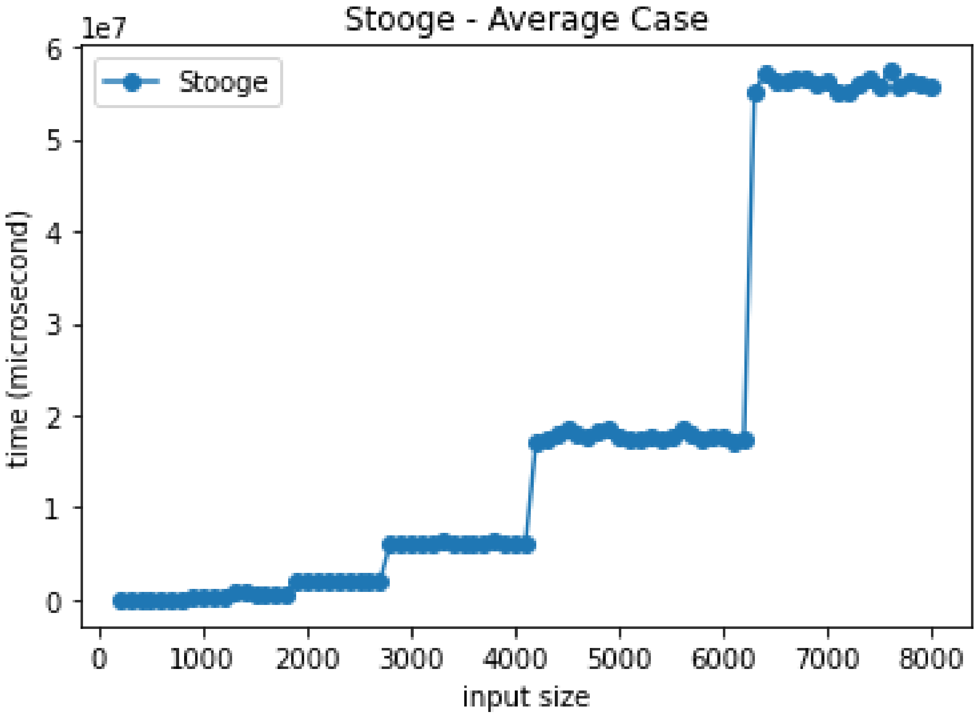


[7] https://en.wikipedia.org/wiki/Stooge_sort

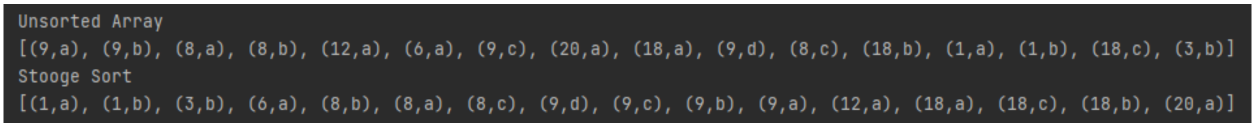
## (Unsorted Array) => Average Case



| | 200 | 900 | 1600 | 2300 | 3000 | 3700 | 4400 | 5100 | 5800 | 6500 | 7200 | 7900 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Stooge** | 4650.8 | 221789.4 | 696608.9 | 1957236.9 | 6178074.2 | 6154738.9 | 17918993.6 | 17423569.3 | 17437037.0 | 56153075.4 | 55027805.7 | 55919418.2 |



## Stable:  No



```
Unsorted Array
[(9,a), (9,b), (8,a), (8,b), (12,a), (6,a), (9,c), (20,a), (18,a), (9,d), (8,c), (18,b), (1,a), (1,b), (18,c), (3,b)]
Stooge Sort
[(1,a), (1,b), (3,b), (6,a), (8,b), (8,a), (8,c), (9,d), (9,c), (9,b), (9,a), (12,a), (18,a), (18,c), (18,b), (20,a)]
```

## Note:

When we look at Figure 1, it can be said that my average graph looks like the "n ^ 2" graph. It is observed that he makes some jump due to algorithm and computer. It can be said that there is a "n ^ 2 or n ^ 2.7" graph for our Best graph. This algorithm is a recursive algorithm. It can be said that it is quite inefficient. I observed that the experiment result was not stable. I can say that it is very inefficient compared to the others. I would not choose. Again, it is one of the reasons why I do not prefer it because of space complexity. It is quite inefficient.

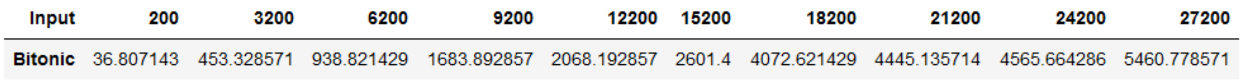
## *3.5 Bitonic Sort Algorithm*

Bitonic mergesort is a parallel algorithm for sorting. It is also used as a construction method for building a sorting network. The algorithm was devised by Ken Batcher.[8]

The number of comparisons done by Bitonic sort are more than popular sorting algorithms like Merge Sort [ does O(nLogn) comparisons], but Bitonice sort is better for parallel implementation because we always compare elements in predefined sequence and the sequence of comparison doesn't depend on data. Therefore it is suitable for implementation in hardware and parallel processor array.[9]

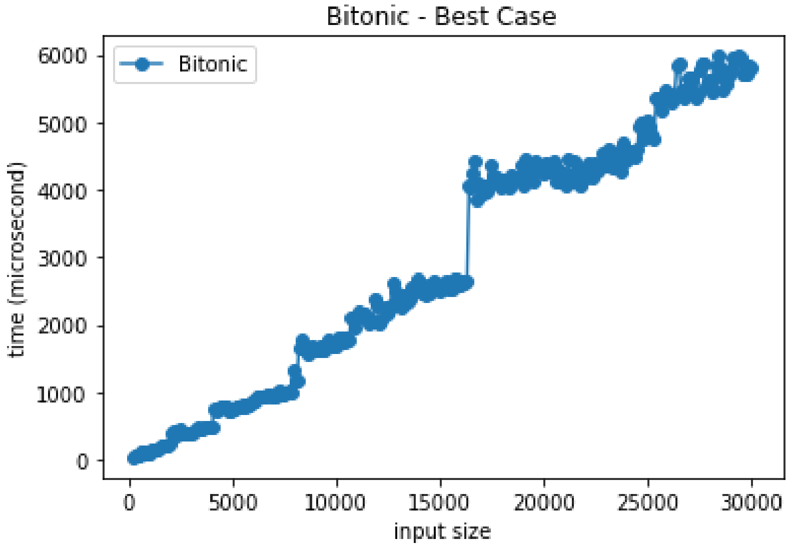
- Worst-case performance O ( log^2(n))
- Best-case performance Θ ( log^2(n) )
- Average performance Ω ( log^2(n))
  - Space Complexity O(n*log(n)^2)

Data: (Unit microsecond)



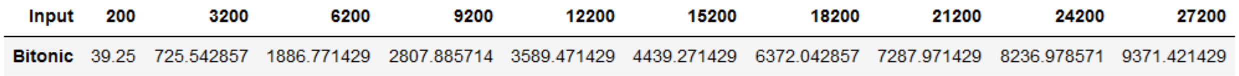
| Input | 200 | 3200 | 6200 | 9200 | 12200 | 15200 | 18200 | 21200 | 24200 | 27200 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Bitonic** | 36.807143 | 453.328571 | 938.821429 | 1683.892857 | 2068.192857 | 2601.4 | 4072.621429 | 4445.135714 | 4565.664286 | 5460.778571 |

(Sorted Array) => Best Case

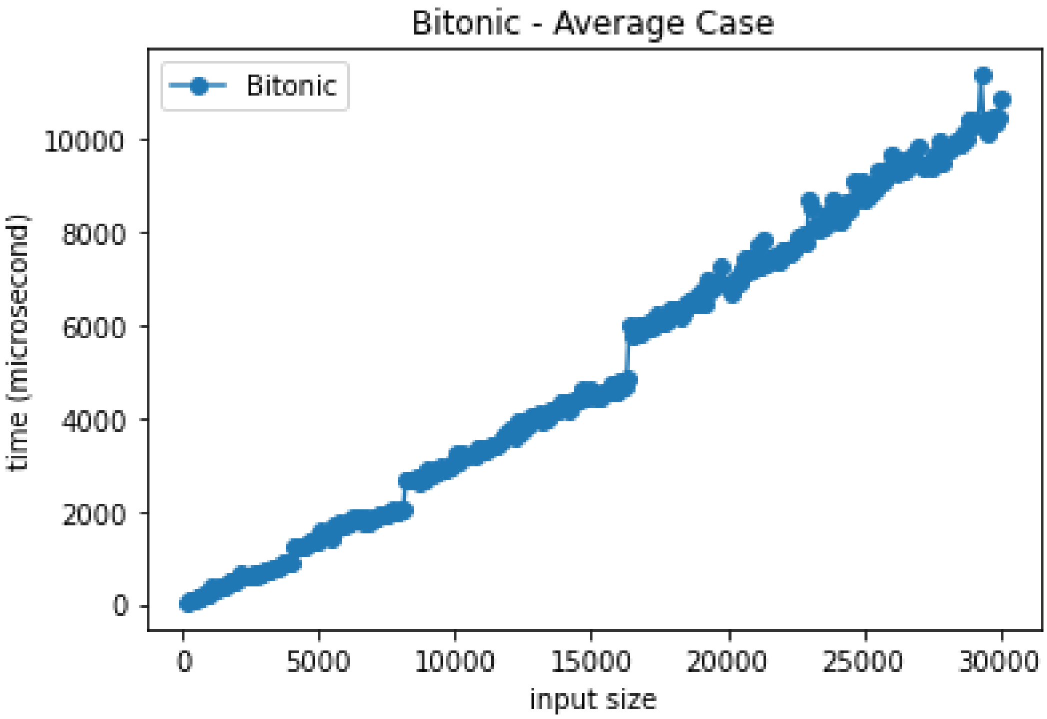


[8] https://en.wikipedia.org/wiki/Bitonic_sorter
[9] https://www.geeksforgeeks.org/bitonic-sort/

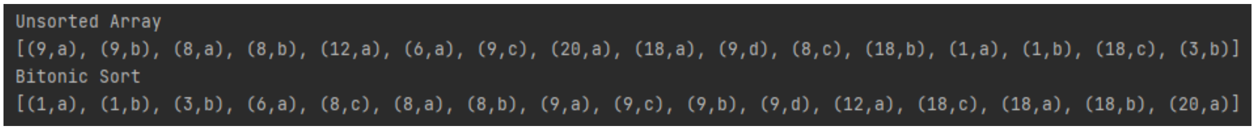
(Unsorted Array) => Average Case



| Input | 200 | 3200 | 6200 | 9200 | 12200 | 15200 | 18200 | 21200 | 24200 | 27200 |
|---|---|---|---|---|---|---|---|---|---|---|
| Bitonic | 39.25 | 725.542857 | 1886.771429 | 2807.885714 | 3589.471429 | 4439.271429 | 6372.042857 | 7287.971429 | 8236.978571 | 9371.421429 |



Stable:  No

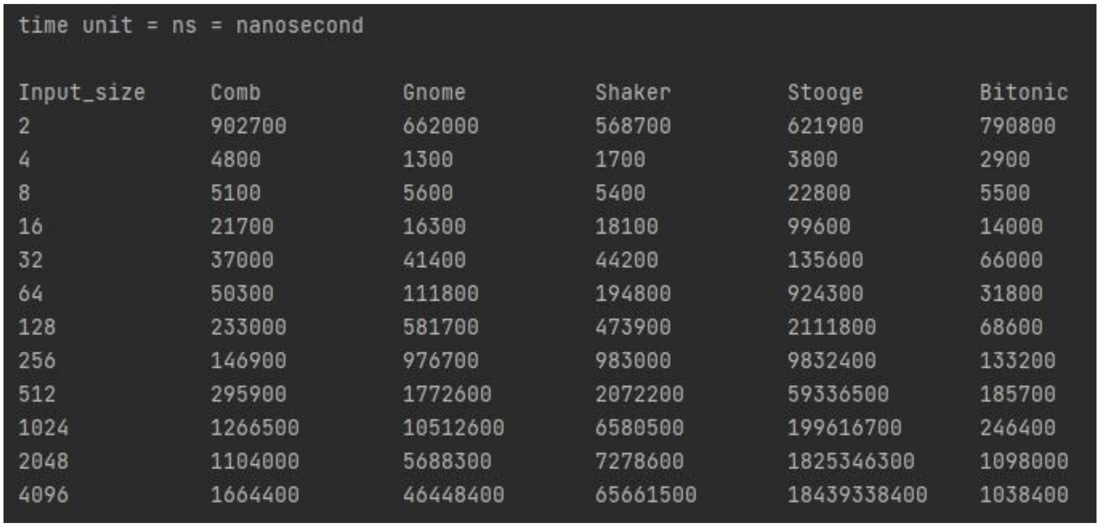


```
Unsorted Array
[(9,a), (9,b), (8,a), (8,b), (12,a), (6,a), (9,c), (20,a), (18,a), (9,d), (8,c), (18,b), (1,a), (1,b), (18,c), (3,b)]
Bitonic Sort
[(1,a), (1,b), (3,b), (6,a), (8,c), (8,a), (8,b), (9,a), (9,c), (9,b), (9,d), (12,a), (18,c), (18,a), (18,b), (20,a)]
```

## Note:

When we look at Figure 1, it can be said that my average and best graph is similar to the "n" graph. An algorithm that only works at powers of 2. It can be said that it is very efficient. I observed that the experiment result was not stable. The algorithm needs more memory than others, I see the reason for this at the speed of the algorithm. I expected it to be more efficient in my experiment, but I still prefer it because of its efficiency.

# 4. Discussion and Result



```
time unit = ns = nanosecond

Input_size     Comb           Gnome          Shaker         Stooge         Bitonic
2              902700         662000         568700         621900         790800
4              4800           1300           1700           3800           2900
8              5100           5600           5400           22800          5500
16             21700          16300          18100          99600          14000
32             37000          41400          44200          135600         66000
64             50300          111800         194800         924300         31800
128            233000         581700         473900         2111800        68600
256            146900         976700         983000         9832400        133200
512            295900         1772600        2072200        59336500       185700
1024           1266500        10512600       6580500        199616700      246400
2048           1104000        5688300        7278600        1825346300     1098000
4096           1664400        46448400       65661500       18439338400    1038400
```

*Figure 2*

*time comparison each of the algorithms with the same input*

In this assignment, I observed that the knowledge of theory does not make sense unless it is tested in experiment and that differences may arise between theory and practice. These are weak but noticeable differences. The reason for this may be the programming language we use, our computer speed, random sequences, numerical errors. In order to minimize these errors, the following can be done; choosing Java as the programming language, stopping every other program while the program is running, taking the average of a few test results.

In this experimental analysis, I saved the outputs to the ".csv" file. I processed these with the help of Python's Numpy, Pandas and Matplotlib libraries. I converted the data from nanoseconds to microseconds for convenience. I used my data by taking the average of 15 different data sets. The most accurate result is observed with too many input values and too many data sets.

As a result of the experiment, I observed that the Stooge algorithm was the most inefficient. Shaker and Gnome were close to each other. Comb and Bitonic algorithms were the most efficient of all. Bitonic may be preferred for powers of 2 and Comb for others. Of course, Shaker or Gnome should be used for stable state.

Stable status is important in ordering our data with more than one variable. For this reason, I tried all of them using the same array. As a result I observed that the Shaker and Gnome algorithms are stable.

As a result of this experiment, I had the opportunity to learn 5 different sorting algorithms with the subject of time complexity. I had the chance to research the properties of these algorithms and test them with each other.

## Last Note:

I would say that the worst-case in this experiment is almost similar to the average-case. Although it is difficult to create the worst case scenario specifically for algorithms, we have the opportunity to learn worst-cases from algorithm analysis. As seen in the experiment, there are places where we can think of average-cases as worst-cases.

## *5. References*

- Wikipedia - *https://en.wikipedia.org/wiki/Main_Page*
- GeeksforGeeks - *https://www.geeksforgeeks.org/*
- Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne - *https://algs4.cs.princeton.edu/home/*
- Tudorcodes - *https://www.tudorcodes.com/blog/stooge-sort/*