HACETTEPE UNIVERSITY
Computer Science and Engineering Department

BBM204 : Software Practicum

Spring 2021, Section: 2

Programming Assignment I

Analysis of Algorithms

Mehmet Giray Nacakcı
Number: 21989009

Due : 24.03.2021

TA: Nebi Yılmaz

1. Software Design Notes

1.0 Problem Definition

Algorithmic Complexity Experiment

A system for testing execution-time of five different comparison-based sorting algorithms is to be designed. Objective is to empirically observe the theoritical complexities of these algorithms.

1.1 Design

Random input Generation

The objects to be sorted are Comparable Celebriti'es who has a name and a follower count. To conduct "random-input" tests, Celebrity objects are created and their attributes are assigned randomly.

Pseudo-code:

// int followerCount = Random.nextInt( );

// String name = random positive integers (< 28) are generated and interpreted as letter position in alphabet. e.g. 1 => A ; 4 => D

Testing Algorithms

For fair comparison, before every different sorting operation, celebrity list is copied and sorting is done on that independent copy.

1.2 Execution

Build command: javac *.java
Run command: java Main

User is asked to enter input size. After tests are conducted, chronometer results are printed. Then, user may continue testing different sizes as they wish.

```
Enter "0" to Terminate or Enter input size: 512

Sorting time for 512  Random inputs, in Microseconds:
Comb->  2210       Gnome-> 5269       Shaker->    3613       Stooge->    105441      Bitonic->   1203

(Worst case) Sorting time for 512  Descending-order sorted inputs, in Microseconds:
Comb->  344     Gnome-> 1439       Shaker->    5222       Stooge->    80477       Bitonic->   553
```

2. Algorithm Analyses

Memory requirements

Comb sort, Gnome sort and Shaker sort do not utilize extra memory. Auxilary space is O(1), total space is O(n).

Experiments and Comparison of Algorithms

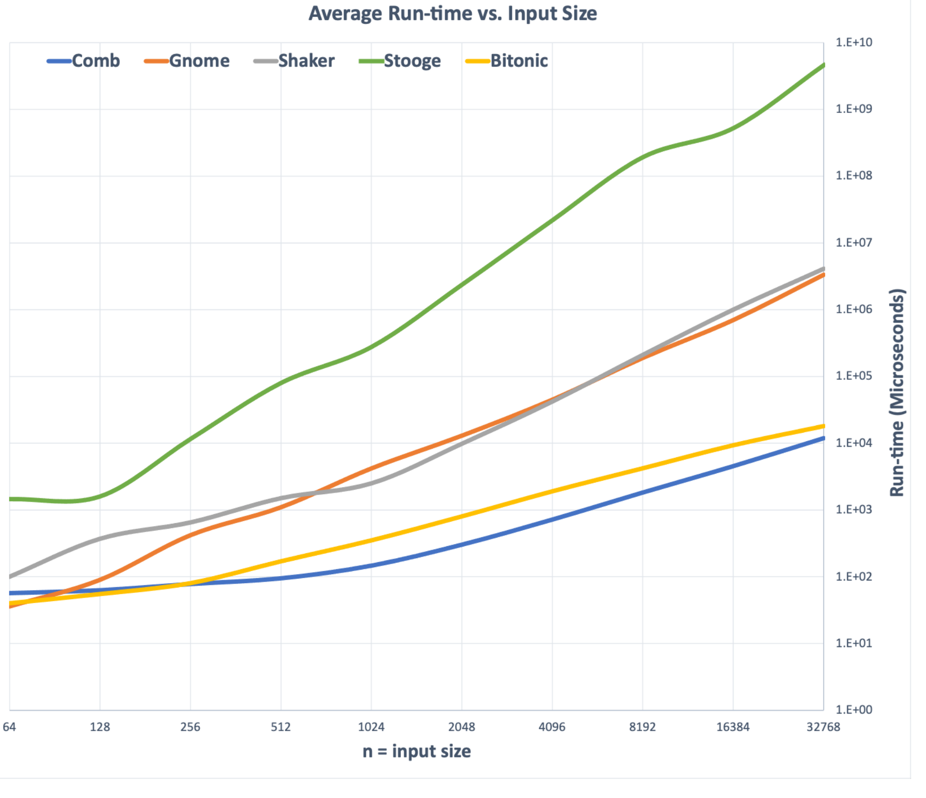
algoritms \ input size	64	128	256	512	1024	2048	4096	8192	16384	32768
COMB	57	63	78	95	147	303	720	1830	4541	11920
GNOME	36	90	420	1100	4200	13000	45000	190000	701874	3350000
SHAKER	100	370	650	1500	2500	9800	42000	210000	1000000	4100000
STOOGE	1454	1592	11530	79546	275392	2355250	21746725	191554618	521132982	4576162887
BITONIC	40	55	80	170	350	800	1900	4200	9300	18000

Table 1 : Average (of 30 tests, done on **random** inputs) Run-times of algorithms, in Microseconds

Random input (Average case) Tests

Shaker-sort goes in hand with Gnome-sort since they both have quadratic complexity.

Stooge-sort is visibly inefficient.



Descending-order input Tests

These tests were done based on an assumption that "**descending-order** sorted inputs would give **worst-case** results". However, empirical data suggested NO major change.

algoritms \ input size	512	1024	2048	4096	8192	16384	32768
COMB	300	320	350	400	920	1900	3920
GNOME	2000	6300	19000	80000	320000	-	-
SHAKER	2500	3500	15000	59000	240000	-	-
STOOGE	85000	282000	2550000	23100000	210000000	-	-
BITONIC	110	250	550	1300	2900	6000	12900

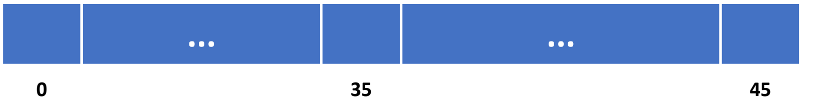
Table 2 : Worst (of 20 tests, done on **Descending-order** inputs) Run-times of algorithms, in Microseconds

2.1 Comb Sort

shrink factor = 1.3

gap = floor (gap / shrink factor)

EXAMPLE



For List size n = 46,

During the execution, the values "gap" will sequentially have, and how many comparisons this will lead to, is given below.

when gap is:	# comparisons happen	#comparisons Formulated	result (alternating)
35	45 – 35	n – n/1.3	= (0.3/1.3) n
26	45 - 26	n – (n – n/1.3)	= n / 1.3
20	45 - 20	n – (n – (n – n/1.3))	= (0.3/1.3) n
15	45 - 15	n – (n – (n – (n – n/1.3)))	= n / 1.3
11	45 - 11	...	= (0.3/1.3) n
8	45 - 8	...	...
6	45 - 6	...	...
3	45 - 3	...	...
2	45 - 2		
1	45 - 1		
1	45 - 1		
1	45 - 1		
...	as long as it is not sorted yet.		

Approx. $\log_{1.3}$n rows

up to **n** rows

Theoritical Complexity

For simplicity of calculations, let us assume that
gap = gap / shrink (without floor function)

Total average #comparisons = ($\log_{1.3}$n) . $\frac{\frac{n}{1.3}+\frac{0.3n}{1.3}}{2} = \frac{n}{2}\log_{1.3}n$ comparisons

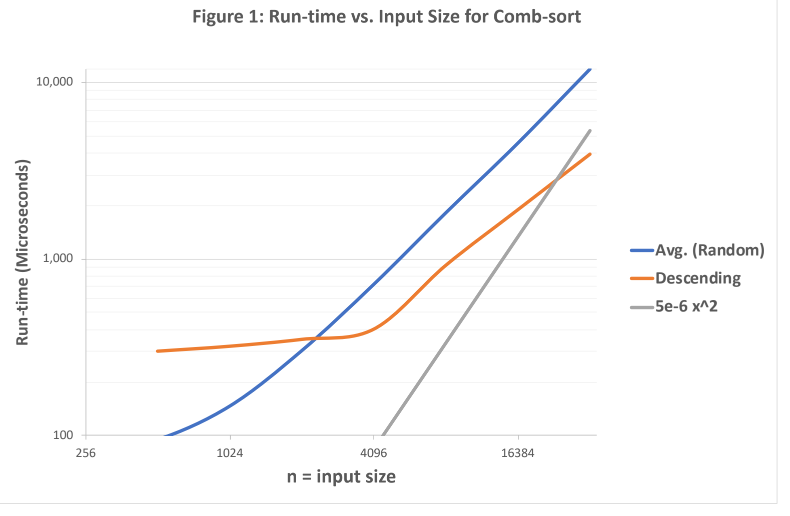
However, since execution can still go on for some time after gap reaches 1,

Max #comparisons $\sim \frac{n}{2}(n) = $ **$O(n^2)$**

Empirical Analysis

- Inputs being in Descending-order turned-out NOT to be the worst case condition.
- The curve obtained from experiment data, is estimated to asymptotically approach the function

$y = 0.000005 \; x^2$ which expectedly have $O(n^2)$ complexity.



2.2 Gnome Sort

Theoritical Complexity

Since list has n elements, maximum number of swaps is the maximum number of two-element combinations:

$$\binom{n}{2} = n(n-1)/\,2$$

Assuming for the worst, let us say, with every pass, we always traverse the whole list and do just one swap. That is $n - 2 + 1$ comparisons for every swap.

Then, max total number of comparisons would be $n(n-1)/2 \; * \; (n-1) \; = \; \mathbf{O(n^3)}$

Empirical Analysis

Assuming power law for cost function, $T_{gnome}(n) = c.n^b$

The growth rate for an example interval can be calculated by their ratio.

$$\frac{T(2n)}{T(n)} = \frac{c\,.\,nb\,.\,2^b}{c\,.\,nb} = 2^b$$

Examples:

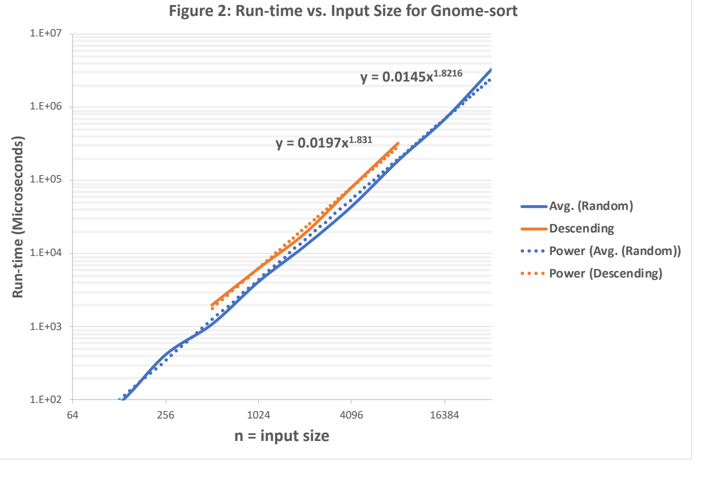
$T_{gnome}(1024) / T_{gnome}(512) = 4200 / 1100 = 3.81 = 2^{1.92}$

$T_{gnome}(16384) / T_{gnome}(8192) = 701874 / 190000 = 3.69 = 2^{1.88}$

$T_{gnome}(8192) / T_{gnome}(4096) = 190000 / 45000 = 4.22 = 2^{2.07}$

These examples demonstrates that, in fact, $\mathbf{O(n^2)}$ is a tighter bound for Gnome-sort.

Also, trendlines below are estimated to be power of **1.82** which are close to **quadratic**.



2.3 Shaker Sort

```
n = 8 EXAMPLE

0 1 2 3 4 5 6 7    indices

- - - - - - - -     8 elements
L>->->->->->->R     7 comparisons
L<-<-<-<-<-<R .     6 comparisons
. L>->->->->R .     5 comparisons
. L<-<-<-<R . .     4 comparisons
. . L>->->R . .     3 comparisons
. . L<-<R . . .     2 comparisons
. . . L>R . . .     1 comparisons
. . . . . . . .       Sorted (ascending)

< and > indicates direction of possible swaps
L and R shows the scope of comparison
- denotes not sorted element
. denotes sorted element and is in correct position
```

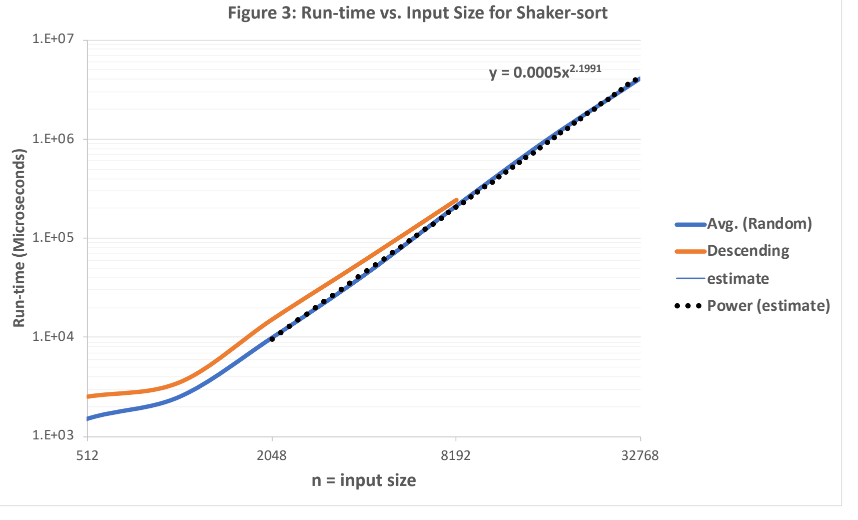
Theoritical Complexity

Shaker sort is a bidirectional bubble-sort. With every pass, number of remaining items to sort decrements by 1. Therefore, max number of total comparisons is

$$T_{shaker}(n) = (n-1) + (n-2) + \ldots\ldots + 2 + 1$$

$$= (n-1)(n) / 2$$

$$= (n^2 - n) / 2 \quad \sim \tfrac{1}{2}\ n^2$$



Empirical Analysis

Assuming power law for cost function, $T_{shaker}(n) = c.n^b$

The growth rate for an example interval can be calculated by their ratio.

$$\frac{T(2n)}{T(n)} = \frac{c\,.\,nb\,.\,2^b}{c\,.\,nb} = 2^b$$

Examples:

$T_{shaker}(16384) / T_{shaker}(8192) = 1000000 / 210000 = 2^{2.25}$

$T_{shaker}(32768) / T_{shaker}(16384) = 4.1 = 2^{2.03}$

In addition, the estimated power function on graph is $y = 0.0005\ x^{2.19}$

These polynomial powers 2.25 , 2.03 , and 2.19 are close to **quadratic** (2.0) complexity, as expected.

2.4 Stooge Sort

Theoritical Complexity

Recursive relation for cost:

- Every call does 1 comparison and then calls 3 calls.

$T(n) = 1 + 3\,T(2n/3)$

$$= 3^0 + 3^1 + 3^2 + \dots + 3^h = (3^{h+1} - 1) / (3 - 1) \quad \sim (3/2)\, 3^h$$

- 3^0 is for the root of recursion tree
- 3^h is for leaves of recursion tree
- **h** is height of recursion tree and equals to $\log_{3/2} n$ because a list segment is divided by 3/2 to obtain a 2/3 portion of it.

$$T(n) \sim (3/2)\, 3^h = \Theta\left(3^{(\log_{3/2} n)} \right) = \Theta\left(n^{(\log_{1.5} 3)} \right) = \Theta\left(n^{2.71} \right)$$

- Worst case comparison complexity will not differ from average case, because regardless of input content, all possible recursive calls will be made. Only difference will be the number of swaps.

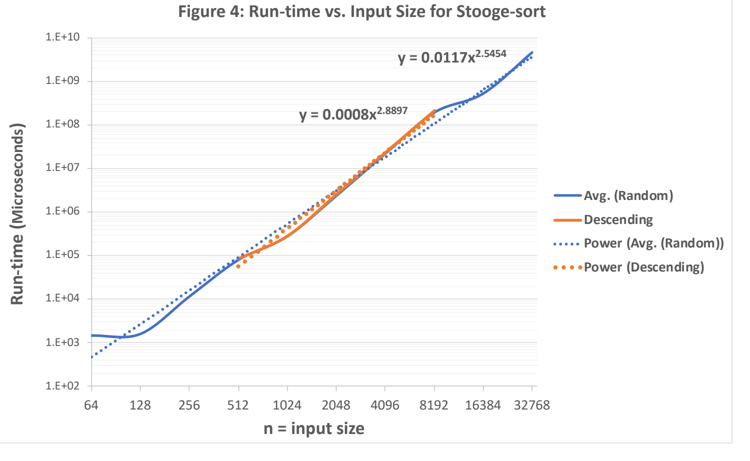
Empirical Analysis

Since the theoritical complexity is shown as

$$\Theta\left(\mathbf{n}^{2.71} \right),$$

Assuming power law $T_{stooge}(n) = c.n^b$,

The growth rate for an example interval can be calculated by their ratio.



$$\frac{T(2n)}{T(n)} = \frac{c\,.\,nb\,.\,2^b}{c\,.\,nb} = 2^b$$

- $T_{stooge}(8192) / T_{stooge}(4096) = 8.8 = \mathbf{2^{3.13}}$
- The function estimated from the average-case test data is $y = 0.0117\, \mathbf{x^{2.5454}}$
- The function estimated from descending-order test data is $y = 0.0008\, \mathbf{x^{2.8897}}$

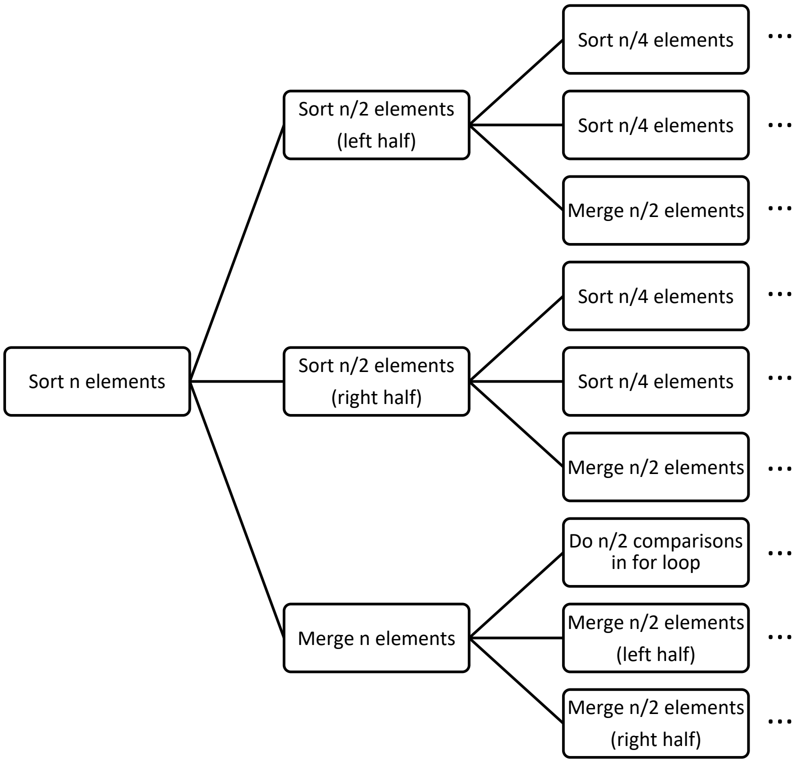
These polynomial powers 3.13 , 2.54 , and 2.88 are close to the theoritical (**2.71**) complexity.

Stooge-sort Space Complexity

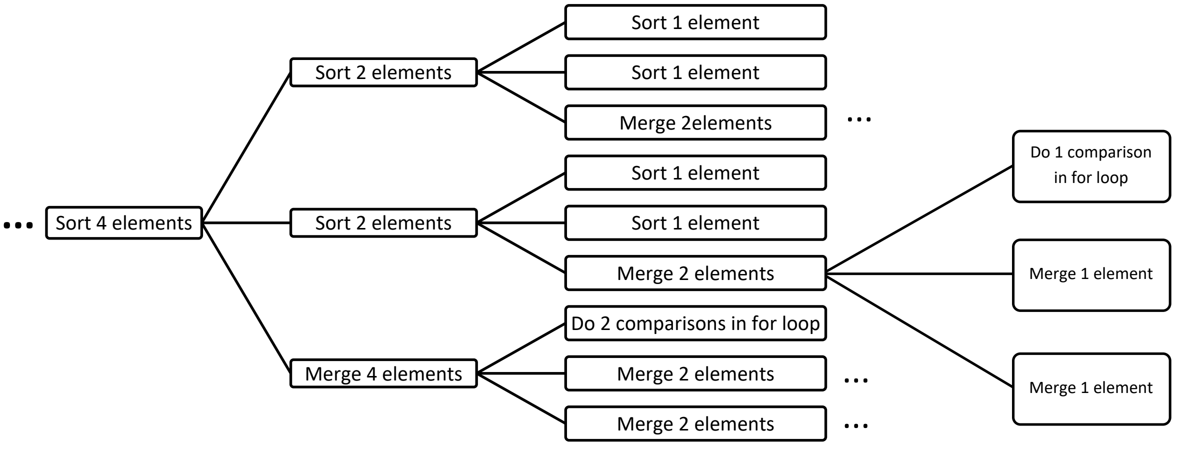
- Space used for input list $= O(n)$
- Auxilary spaced used for recursive calls is $O(n^{2.71})$ since every call does 1 comparison.

  Total space: $O(n) + O(n^{2.71}) = O(n^{2.71})$

2.5 Bitonic Sort



Model 1: Beginning of recursive calls



Model 2: Leaves of recursive tree

Theoritical Complexity

S(n) = #**comparisons** to Bitonic-Sort a list of length n

M(n) = #**comparisons** to Bitonic-Merge n elements

Recursive definitions of these functions:

$S(n) = 2\,S(n/2) + M(n)$

$M(n) = n/2 + 2\,M(n/2)$

Initial terms :

$\mathbf{S(2) = 1}$, $S(1) = 0$

$\mathbf{M(2) = 1}$, $M(1) = 0$, $M(4) = 4$

By Telescoping (back-substitution) method, first M(n) should be obtained as a function of n. Then it will be subsituted into S(n) and S(n) will be obtained as a function of n.

Telescoping M(n):

$M(n) = n/2 + 2\,M(n/2)$

$M(n) = n/2 + 2\,[\, n/4 + 2M(n/4)\,]$

$\quad = n/2 + 2n/4 + 4M(n/4)$

$M(n) = n/2 + 2\,[\, n/4 + 2\,[\,n/8 + 2M(n/8)\,]\,]$

$\quad = n/2 + 2n/4 + 4n/8 + 8\,M(n/8)$

$$M(n) = \underbrace{n/2 + n/2 + n/2 + \ldots}_{\mathbf{x}\ \text{many}} + \mathbf{2^x\,M(n/2^x)}$$

To substitue $M(2) = 1$ into the function, we define an x such that $n/2^x = 2$ and therefore

$$\mathbf{x = \log_2 \tfrac{n}{2}}$$

$M(n) = (n/2)\,x + 2^x\ M(2)$

$\quad = (n/2)\,(\log_2 \frac{n}{2}) + 2^{(\log_2 (n/2))}$

$\quad = (n/2)\,(\log_2 \frac{n}{2}) + n/2$

$\quad = (n/2)\,(\log_2 n - \log_2 2 + 1)$

$\mathbf{M(n) = (n/2)\,(\log_2 n)}$ is found.

Telescoping S(n):

$S(n) = 2\,S(n/2) + M(n)$

$\quad = 2\,S(n/2) + \frac{n}{2}(\log_2 n)$

$S(n) = 2\,\left[\,2\,S(n/4) + \frac{n}{4}(\log_2 \frac{n}{2})\,\right] + \frac{n}{2}(\log_2 n)$

$\quad = 4\,S(n/4) + \frac{n}{2}(\log_2 \frac{n}{2}) + \frac{n}{2}(\log_2 n)$

$= 4\left[2\,S(n/8) + \frac{n}{8}(\log_2 \frac{n}{4})\right] + \frac{n}{2}(\log_2 \frac{n}{2}) + \frac{n}{2}(\log_2 n)$

$= 8\,S(n/8) + \frac{n}{2}\,[\,\log_2 \frac{n}{4} + \log_2 \frac{n}{2} + \log_2 n\,]$

$= 8\,S(n/8) + \frac{n}{2}\,[\,\log_2 n - 2 + \log_2 n - 1 + \log_2 n\,]$

$= 2^y\,S(n/2^y) + \frac{n}{2}\,[\,y \log_2 n - (1+2+\ldots+(y-1))\,]$

$= 2^y\,S(n/2^y) + \frac{n}{2}\,[\,y \log_2 n - y(y-1)/2\,]$

To substitue $S(2) = 1$ into the function, we define y such that $n/2^y = 2$ and this implies $\mathbf{y = \log_2 \frac{n}{2}}$

In that case, S(n) =

$= (2^{(\log_2 (n/2))})\,S(2) + \frac{n}{2}\,(\log_2 \frac{n}{2})\,(\log_2 n + \frac{1-y}{2})$

$= \frac{n}{2}\,[\,1 + (\log_2 n - 1)\,(\log_2 n + 1 - \frac{\log_2 n}{2})\,]$

$= \frac{n}{2}\,[\frac{(\log_2 n)^2}{2} + \frac{\log_2 n}{2}]$

$$S(n) = \frac{n(\log_2 n)^2 + n\log_2 n}{4} = \boldsymbol{\Theta(n\,(\log_2 n)^2)}$$

comparisons

Worst case and average case has same number of comparisons since this is a divide-and-conquer algorithm; because all possible recursive calls will be made. Number of comparisons cannot change, but number of swaps can be maximized for worst case. Thus, worst-case is when each comparison results in a swap.

Bitonic-sort empirical analysis

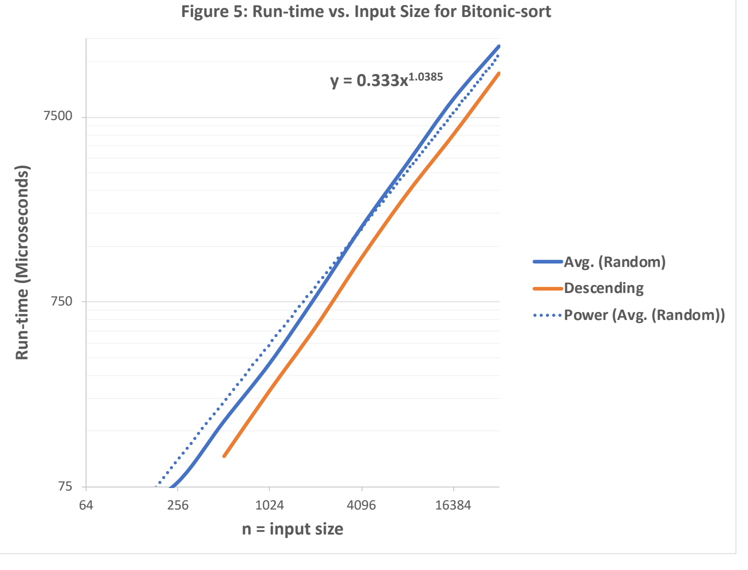
If complexity of Bitonic-sort is $T_{\text{bitonic}}(n) = \Theta(n\,(\log_2 n)^2)$, Then $\frac{T(2n)}{T(n)}$ should be $\sim 2^1$

$$\frac{T(2n)}{T(n)} = \frac{2n\,.\,(\log_2 2n)^2}{n\,.\,(\log_2 n)^2} = \frac{2\,(\log_2 n+1)^2}{(\log_2 n)^2} = 2\left(\frac{\log_2 n+1}{\log_2 n}\right)^2 = 2\left(1+\frac{1}{\log_2 n}\right)^2 \;\sim\; \mathbf{2} \;=\; \mathbf{2^1}$$

Now, let us test if empirical data complies with this hypothesis.

- $T_{\text{bitonic}}(32768) / T_{\text{bitonic}}(16384) = 18000 / 9300 = 1.93 = 2^{\mathbf{0.94}}$
- $T_{\text{bitonic}}(4096) / T_{\text{bitonic}}(2048) = 12900 / 6000 = 2.15 = 2^{\mathbf{1.10}}$
- The power function estimated from graph is $\mathbf{y = 0.333\ x^{1.0385}}$

Indeed, the powers 0.94, 1.10 and 1.03 are very close to the expected power: **1** .



Discussion

Giving inputs into the algorithm in Descending-order has increased the performance by a constant factor. That is because Bitonic-sort algorithm already tries to sort some parts of the list in descending order during its execution, for the sake of creating a "bitonic sequence".

3. Algorithmic Stability

Unsorted	
Follower Count	Celebrity Name
21	**A**
40	**B**
3	**C**
40	**D**
21	**E**
5	**F**
564	**G**
40	**H**

Sorted Follower Count	Sorted Celebrity Names				
	Comb	Stooge	Bitonic	Gnome	Shaker
3	**C**				
5	**F**				
21	**A**	**A**	**E**	**A**	
21	**E**	**E**	**A**	**E**	
40	**D**	**D**	**D**	**B**	
40	**B**	**B**	**H**	**D**	
40	**H**	**H**	**B**	**H**	
564	**G**				
	NOT STABLE	NOT STABLE	NOT STABLE	**STABLE**	**STABLE**

For stability tests, one-letter named Celebrities are created manually.

Discussion

Gnome-sort and Shaker-sort are the only algorithms that reserved both AE and BDH order. It is because they are variants of bubble sort, they compare only **adjacent** items, and they do NOT swap **equal** items and preserve their given ordering. Whereas, for instance Gnome-sort swaps items that have a lot of distance between them, and some equal items are seperated from each other, not likely to be put back in previous order. Furthermore, Bitonic-sort and Stooge-sort is likely to seperate the equal keyed items into seperate divisions due to their divide-and-conquer strategy.

Advantage of stability

When we sort celebrities by their follower count, we would also like to see the celebrities with same number of followers to be sorted by their names within each other. In situations like these, that sorting needs to be done with more than one criteria, first we sort items based on the lower priority attribute, then the higher priority attribute. The only case we get what we want is if the sorting algorithm was a stable one. Otherwise, the "order within" is not ensured to be preserved.

End of report.