Algorithm Design and Analysis

FINAL PROJECT REPORT

Comparison Of

Three Weird Algorithms

Prepared by:

Baal's Abandoned Children

(Jocelin, Nicholas & Tiffany)

Odd semester 2nd year 2022

Computer Science Program

Binus University International

Table of Contents

Project name: Comparison Of Algorithms

Group members:

1. Tiffany Widjaja - 2502059360 (L3AC)

2. Nicholas Valerianus Budiman - 2502055596 (L3AC)

3. Jocelin Wilson - 2501963330 (L3AC)

I. Introduction/Background

A sorting algorithm is an algorithm made up of a series of instructions that takes an array as input, performs specified operations on the array, and outputs a sorted array. We chose to do a comparison of algorithms for our final project so that we can study the difference between their efficiencies such as the run time, and memory consumption, and find out in what situation would it be most optimal to use them. The three algorithms we will be comparing are all sorting algorithms however there is a twist to them as they are not the most popular algorithms. In fact, these are the strangest sorting algorithms that we didn't know existed. The three sorting algorithms are:

- Stalin
- Sleep
- Cocktail

These strange algorithms are selected based on the criteria we created:

1. It is not often heard of or implemented
2. Unusual names (Subjective Opinion)

3. Algorithms created for jokes

II. Problem

There are plenty of algorithms available for programmers to choose from however many occasionally use algorithms that are famously known to be the most efficient or have the best time complexity. Other newer algorithms lack publicity and thus may get forgotten behind or not even analyzed.

Factors such as efficiency like memory usage, time complexity, run time, and simplicity are undeniably important for us to consider before deciding to implement the algorithm on a problem. This is precisely why we believe it is important for us to do comparisons of the 3 previously mentioned algorithms, in hopes of bringing new insight to programmers into what the algorithms are capable of. Why these algorithms are not so popular or not often used may be found in the discussion of our research.

III. Proposed Solution

The algorithms we have decided to compare are Stalin sort, sleep sort, and cocktail sort. These algorithms will all be coded in C++. Our group would like to test out the advantages and disadvantages of the three strangest algorithms for the difference. Therefore we are doing a comparison between the Stalin, Sleep, and Cocktail method. This has to be done to make sure that the most effective and efficient algorithm can be used on the problems for good optimization and efficient time and memory usage. In this project, we will compare and study the algorithms' run time, memory usage, pros, cons, time complexity, and code simplicity so that other fellow

programmers and ourselves included can have a good idea of what algorithms to use in different case scenarios for optimal solutions in the future.

IV. Measurement

Since we are comparing algorithms we would have to measure the run time of the algorithm measured in ms. We will also have to measure the amount of physical memory as well as the peak memory taken when running each algorithm in bytes. This is done to see which out of all the three sorting algorithms performed the best (most efficient) when it comes to run time and memory taken. We will be using 5 different sizes to measure which algorithm is most suited for sorting which array. These different-sized arrays will consist of nearly sorted and randomly sorted numbers. The array size would be 1000, 10000, 15000, 30000, and 50000. The measurement of run time and memory will run 5 times. Afterward, we will calculate the mean of the run times by adding all run times and dividing it by 5. This is done to find the average to avoid any data anomalies.

We used java code to create a partially sorted array which is done by generating a random number for the first element of the array and performing a mathematical equation on that number and performing a different mathematical equation that results in a similar number to every number after each element. Through this we create an array where each element is barely unsorted, then we recheck whether it worked properly or not by looking through the array.

```
public static void main(String[] args) {
    int [] array = new int[15000];    // Almost sorted Array of 1

    array[0] = (int)(Math.random () * 10) + 1; //set a first elem

    for (int a = 1; a < array.length; a++) {
        array[a] = array[a-1] + (int)(Math.random() * 12) - 2; //
    }
```

V. Concept of the Algorithm

The weird sorting algorithms are:

- Stalin

- Sleep

- Cocktail

The Stalin sorting algorithm is a sorting algorithm that sorts arrays and numbers by eliminating numbers that are not positioned correctly. This occurs repeatedly until the numbers are sorted in order. For example, we have an unsorted array: {1,2,4,3,5,8,7,10,9,13,11,12}. After implementing Stalin sort, the array turns to {1,2,4,5,8,10,13}. As you can see it eliminates all the numbers that are not placed in order. As you can see it eliminates all the numbers that are not placed in order. However, as this is not really implementable as it removes numbers and you'll only be left with one number in a case of a reversed array, there is another version for the Stalin sort called the "Safe Stalin Sort". The safe version, unlike the non-safe one, doesn't remove any numbers and is implementable. Like the non-safe version, it traverses the elements one by one, but if the next element is out of order, the next element instead will go to the beginning and

repeat until the array is sorted. We will be implementing the Safe Stalin Sort, instead of the non-safe version.

Sleeping sort, also known as the king of laziness, is a sorting algorithm that involves time. Sleeping sort creates a thread for every element in the array and puts the element to "sleep" and will wake up according to their value. So the element having the least amount of sleeping time wakes up first and the number gets printed and then the second last element and so on. The largest element wakes up after a long time and then the element gets printed at the end. Due to the elements being dependent on the time, this doesn't allow negative numbers in the array. The downside of this is that this algorithm doesn't produce a correct sorted output every time. This generally happens when there is a very small number to the left of a very large number in the input array.

Finally, there is the cocktail algorithm, which is another variation of the bubble sort. The algorithm gets its name from the way bubbles sort by repeatedly passing through the array, comparing adjacent elements, and swapping them if it is not sorted, which is similar to the way a bartender mixes cocktails by shaking them. The first stage is traversing the array starting from the left all the way to the right, just like Bubble Sort. During the loop, adjacent items are compared and if the value on the left is greater than the value on the right, then values are swapped. At the end of the first iteration, the largest number will reside at the end of the array. The second stage goes through the array in the opposite direction; starting from the item just before the most recently sorted item, it goes back to the start of the array. Here also, adjacent items are compared and are swapped if required. With this array example (2, 3, 4, 5, 1), bubble sort requires four traversals while Cocktail sort requires only two traversals. This algorithm is also known as the happy hour sort, ripple sort, and shuttle sort.

VI. The Flow

1. Prepare the coding algorithm and 5 different sizes of nearly sorted and random arrays.

2. Find the time and space complexity of algorithms.

3. Record the run time and physical memory with tables and test the arrays 5 times, score out the average.

4. Comparison between memory, run time, and code simplicity of the algorithms.

5. Identifying the pros and cons of each algorithm.

6. Identifying optimal situations when such an algorithm is used.

7. We conclude whether the algorithms can be applied or not and in which situation the algorithms are best.

VII. Results

Stalin Sort

STALIN SORT (Partially)

RUNS	SIZE	RUNTIME(MS)	CURRENT MEMORY	PEAK MEMORY
1	1000	357	4202496	4206592
2	1000	377	4190208	4194304
3	1000	351	4202496	4206592
4	1000	361	4186112	4190208
5	1000	329	4186112	4190208
	Average:	355	4193484.8	4197580.8
1	10000	26426	4390912	4395008
2	10000	26701	4382720	4386816
3	10000	26620	4382720	4386816
4	10000	30304	4337664	4382720
5	10000	26898	4395008	4384550
	Average:	27389.8	4377804.8	4387182
1	15000	96690	6230016	6234112
2	15000	155114	4444160	4448256
3	15000	93319	4448256	4452352
4	15000	93957	4448256	4452352
5	15000	143729	4457213	4469237
	Average:	116561.8	4805580.2	4811261.8

STALIN SORT(Random)

RUNS	SIZE	RUNTIME(MS)	CURRENT MEMORY	PEAK MEMORY
1	1000	597	4206592	4210688
2	1000	607	4202496	4206592
3	1000	587	4202496	4206592
4	1000	591	4182016	4186112
5	1000	542	4194304	4198400
	Average:	584.8	4197580.8	4201676.8
1	10000	154274	4452352	4456448
2	10000	163551	4460544	4464640
3	10000	155469	4452352	4456448
4	10000	145679	4357098	4464895
5	10000	68066	4329472	4333568
	Average:	137407.8	4410363.6	4435199.8
1	15000	154474	4456448	4460544
2	15000	593149	4481024	4485120
3	15000	163803	4460544	4464640
4	15000	180105	4448256	4452352
5	15000	1913791	4063232	4067328
	Average:	601064.4	4381900.8	4385996.8

Sleep Sort

SLEEP SORT (Partially)				
RUNS	SIZE	RUNTIME(MS)	CURRENT MEMORY(bytes)	PEAK MEMORY(bytes)
1	1000	3422	4636672	20316160
2	1000	3421	4747264	20312064
3	1000	3457	4960256	20357120
4	1000	3412	4792320	20164608
5	1000	3434	4694016	20287488
	Average:	3429.2	4766105.6	20287488
1	10000	35157	12619776	168644608
2	10000	35248	12218368	167530496
3	10000	35251	11984896	167124992
4	10000	35148	12107776	167976960
5	10000	35125	12148736	168087552
	Average:	35185.8	12215910.4	167872921.6
1	15000	106449	25128960	493666304
2	15000	106437	27455488	493830144
3	15000	106644	25370624	493236224
4	15000	106479	25542656	494194688
5	15000	106409	25468928	494358528
	Average:	106483.6	25793331.2	493857177.6
1	30000	186679	28041216	163287040
2	30000	188399	39071744	173277184
3	30000	184115	38809600	172703744
4	30000	170428	42221568	182853632
5	30000	170880	34516992	173260800
	Average:	180100.2	36532224	173076480

SLEEP SORT (Random)				
RUNS	SIZE	RUNTIME(MS)	CURRENT MEMORY(bytes)	PEAK MEMORY(bytes)
1	1000	1040	4751360	20099072
2	1000	1040	4718592	20062208
3	1000	1046	4730880	20115456
4	1000	1043	4763648	20217856
5	1000	1044	4694016	20279296
	Average:	1042.6	4731699.2	20154777.6
1	10000	3204	4792320	15507456
2	10000	3443	4722688	14962688
3	10000	3154	4730880	14974976
4	10000	3187	4657152	14372864
5	10000	3221	4767744	15237120
	Average:	3241.8	4734156.8	15011020.8
1	15000	16034	15544320	236912640
2	15000	15773	16207872	245575680
3	15000	16087	15568896	238067712
4	15000	15742	15355904	245161984
5	15000	15974	16146432	240152576
	Average:	15922	15764684.8	241174118.4
1	30000	365969	36884480	214315008
2	30000	365706	38985728	216928256
3	30000	368807	37588992	218697728
4	30000	383965	33177600	222220288
5	30000	364588	29302784	203378688
	Average:	369807	35187916.8	215107993.6

Cocktail Sort

COCKTAIL SORT (Partially)

RUNS	SIZE	RUNTIME(MS)	CURRENT MEMORY	PEAK MEMORY
1	1000	28	3235840	3235840
2	1000	33	3231744	3231744
3	1000	4	3235840	3235840
4	1000	28	3235840	3235840
5	1000	37	3231744	3231744
	Average:	26	3234201.6	3234201.6
1	10000	508	3301376	3301376
2	10000	438	3309568	3309568
3	10000	448	3297280	3297280
4	10000	660	3305472	3305472
5	10000	672	3305472	3305472
	Average:	545.2	3303833.6	3303833.6
1	15000	817	3346432	3346432
2	15000	913	3342336	3342336
3	15000	942	3338240	3338240
4	15000	770	3342336	3342336
5	15000	890	3346432	3346432
	Average:	866.4	3343155.2	3343155.2
1	30000	1880	3469312	3469312
2	30000	1986	3465216	3465216
3	30000	1861	3469312	3469312
4	30000	1422	3461120	3461120
5	30000	2127	3465216	3465216
	Average:	1855.2	3466035.2	3466035.2
1	50000	4589	3624960	3624960
2	50000	5138	3624960	3624960
3	50000	4498	3624960	3624960
4	50000	5469	3624960	3624960
5	50000	5420	3629056	3629056
	Average:	5022.8	3625779.2	3625779.2

COCKTAIL SORT (Random)

RUNS	SIZE	RUNTIME(MS)	CURRENT MEMORY	PEAK MEMORY
1	1000	5954	3235840	3235840
2	1000	6057	3235840	3235840
3	1000	5208	3235840	3235840
4	1000	5765	3235840	3235840
5	1000	4536	3235840	3235840
	Average:	5504	3235840	3235840
1	10000	450244	3297280	3297280
2	10000	466558	3305472	3305472
3	10000	480377	3305472	3305472
4	10000	422027	3305472	3305472
5	10000	456552	3305472	3305472
	Average:	455151.6	3303833.6	3303833.6
1	15000	1041002	3338240	3338240
2	15000	1015040	3342336	3342336
3	15000	1006960	3342336	3342336
4	15000	1025891	3346432	3346432
5	15000	1000227	3342336	3342336
	Average:	1017824	3342336	3342336
1	30000	4501050	3461120	3461120
2	30000	3790061	3465216	3465216
3	30000	3829977	3469312	3469312
4	30000	3803590	3461120	3461120
5	30000	3828456	3469312	469312
	Average:	3950626.8	3465216	2865216
1	50000	11018918	3624960	3624960
2	50000	12750523	3616768	3616768
3	50000	10472972	3620864	3620864
4	50000	11112507	3620864	3620864
5	50000	10938495	3629056	3629056
	Average:	11258683	3622502.4	3622502.4

VIII. Discussion

Average Runtime				
Array Type	Size	Stalin	Sleep	Cocktail
Partially	1000	355.00	3,429.20	26.00
Random	1000	584.80	6,642.80	5,504.00
Partially	10000	27,389.80	35,185.80	545.20
Random	10000	137,407.80	97,450.40	455,151.60
Partially	15000	116,561.80	106,483.60	866.40
Random	15000	601,064.40	130,251.20	1,017,824.00
Partially	30000		180,100.20	1,855.20
Random	30000		369,807.00	3,950,626.80
Partially	50000			5,022.80
Random	50000			11,258,683.00

- Stalin can't do large arrays; Cocktail can do the largest.

- Cocktail works the fastest in a smaller size but also works worst in large size.

- Cocktail works the fastest in partially sorted arrays.

- Sleep works the fastest in a random array with a bigger array size.

Average Current Memory				
Array Type	Size	Stalin	Sleep	Cocktail
Partially	1000	4,193,484.80	766,105.60	3,234,201.60
Random	1000	4,197,580.80	4,470,374.40	3,235,840.00
Partially	10000	4,377,804.80	12,215,910.40	3,303,833.60
Random	10000	4,410,363.60	20,660,224.00	3,303,833.60
Partially	15000	4,805,580.20	25,793,331.20	3,343,155.20
Random	15000	4,381,900.80	26,303,692.80	3,342,336.00
Partially	30000		36,532,224.00	3,466,035.20
Random	30000		35,187,916.80	3,465,216.00
Partially	50000			3,625,779.20
Random	50000			3,622,502.40

Average Memory Peak				
Array Type	Size	Stalin	Sleep	Cocktail
Partially	1000	4,197,580.80	20,287,488.00	3,234,201.60
Random	1000	4,201,676.80	12,940,083.20	3,235,840.00
Partially	10000	4,387,182.00	167,872,921.60	3,303,833.60
Random	10000	4,435,199.80	100,809,932.80	3,303,833.60
Partially	15000	4,811,261.80	493,857,177.60	3,343,155.20
Random	15000	4,385,996.80	142,810,316.80	3,342,336.00
Partially	30000		173,076,480.00	3,466,035.20
Random	30000		215,107,993.60	3,351,838.72
Partially	50000			3,625,779.20
Random	50000			3,622,502.40

Stalin takes up more memory at the beginning similar to Cocktail sort, while Sleep takes the least, but as it progresses, Sleep takes more memory with a significant difference. The current memory of Stalin and Cocktail increases steadily without any big jumps. For larger sizes of

arrays, Cocktail uses the least amount of memory. Cocktail is also shown to be better in a random array memory-wise.

Although we initially planned to do until 50000 sized nearly sorted array and randomly sorted array, we stopped measuring the run time due to the IDE not being able to handle it. The IDE just completely stopped working (crashed) and we have to restart it. That explains the missing values in our table results.

Time Complexity:

- Stalin best case: O(N)
- Stalin worst case: O(N^2)
- Cocktail best case: O(N)
- Cocktail worst case: O(N^2)
- Sleep sort best and worst case: O(NlogN + max(input))

Code Simplicity:

1. Safe Stalin
2. Sleep sort
3. Cocktail Shaker

IX. Conclusion & Recommendation

Conclusion

Cocktail is the most optimal at partially sorted and in small size of array compared to all three in runtime and can also handle larger sizes of arrays with more memory efficiency, but the code

complexity is more complex than Cocktail or Stalin. Stalin is just in between Cocktail and Sleep Sort which is not worse but also is not the best. The downside of Stalin is that Stalin is unable to handle large data. Sleep sort takes the most memory. No functional use for sleep sort as it sleeps based on the value of x element in the array.

Recommendation

In data bigger than 30,000 or smaller than 1,000 partially sorted arrays, Cocktail is recommended to be used as it is optimal for smaller data that's partially sorted and can handle larger data. Safe Stalin, although applicable, is not recommended when handling large data but can still be applied for smaller arrays. Preferably random generated arrays with the size of 1000 or less since it has the fastest run time compared to Cocktail sort and Sleep sort.

X. References

https://www.geeksforgeeks.org/cocktail-sort/

https://stackoverflow.com/questions/12324822/sleep-sort-in-c-11

https://www.geeksforgeeks.org/sleep-sort-king-laziness-sorting-sleeping/

https://www.geeksforgeeks.org/cocktail-sort/

https://stackoverflow.com/questions/12324822/sleep-sort-in-c-11

https://www.geeksforgeeks.org/sleep-sort-king-laziness-sorting-sleeping/

XI. Files & Links

Github:

https://github.com/Pandalmation/Strange-Sorting-Algorithms

Result and Comparison:

https://docs.google.com/spreadsheets/d/17vI5OkTi4h-QEMBL3i5t328LhxlrtpqteaA14sj0glk/

Presentation:

https://www.canva.com/design/DAFXRUXOTAE/DRNC98j5fRYEhYcoqKPLQw/view?utm_content=DAFXRUXOTAE&utm_campaign=designshare&utm_medium=link&utm_source=publishsharelink

XII. User Manual

Download the folders in the Github Link:

https://github.com/Pandalmation/Strange-Sorting-Algorithms.git
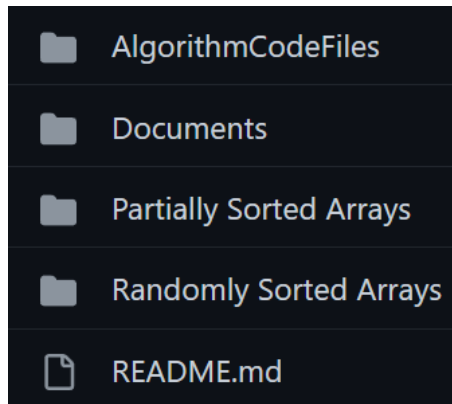
The AlgorithmCodeFiles consists of the weird sorting files which are the Stalin Sort in the Extra folder, Cocktail Sort, Safe Stalin Sort, and Sleep Sort which was compared to one another. The memuse.h file is used for measuring the current physical memory use and the peak memory used physically.
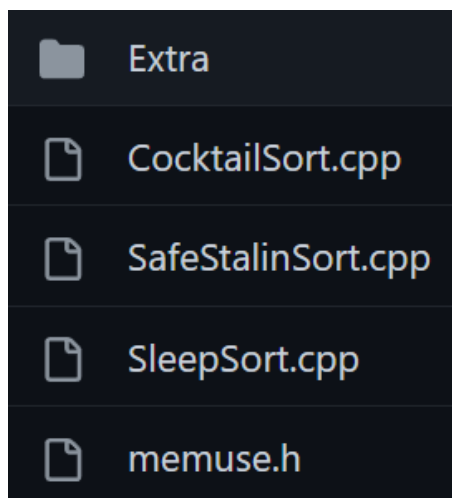
The Partially Sorted Arrays and the Randomly Sorted Arrays folder consist of 5 different sizes of arrays to test different cases. The first one has arrays sorted partially, while the second one has arrays sorted randomly.
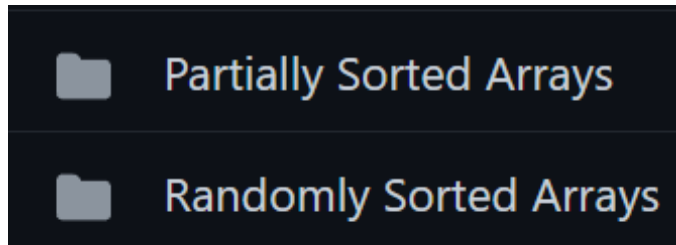
These are the 5 different sizes of arrays according to how sorted they are with 10000, 10000, 15000, 30000, and 50000 as the array size.

So to use it, pick one of the sorting algorithms and open it in the IDE, for example, CocktailSort.cpp for Cocktail Sort.

Scroll down to line 65 and find the array which is set in default to 1000 partially sorted array.

Pick the array type and size, for example randomly sorted array with 15000 as the size. Copy the .txt file.
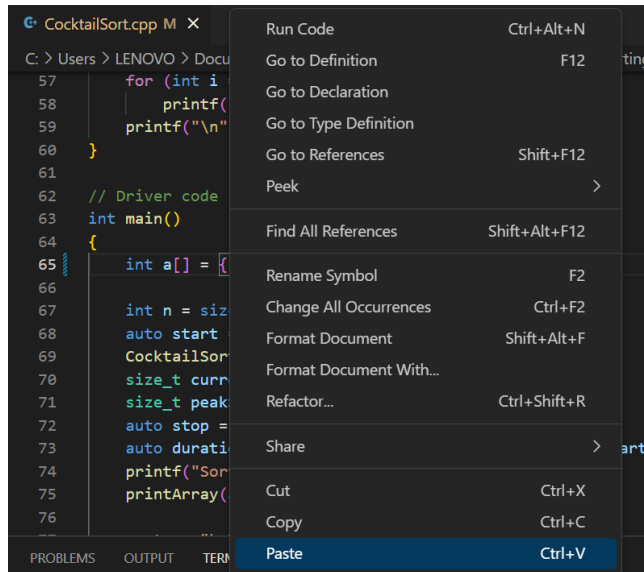
Replace the previous array with the array of your choosing.

Run it and observe the result where the time taken, current memory, and peak memory can be seen.


```
TERMINAL
  96 14996 14997 14998

Time taken by function: 1259453 microsecond

Current Memory: 3383296
Peak memory: 3383296
```

The Document folder consists of files for documentation purposes. It consists of the results of the runtime and memory, the presentation(first project milestone, second project milestone, and the final presentation), and the report (first project proposal, second project proposal, and the final report).

ADA FINAL RUNTIME MEMORY.xlsx

Algorithm Final Presentation.pdf

Algorithm Final Report.pdf

Algorithm Project Milestone 1.pdf

Algorithm Project Milestone 2.pdf

Project Proposal Milestone 1.pdf

Project Proposal Milestone 2.pdf