THE COMPARISON OF SORTING ALGOROTHMS IN A LOCATION FINDING PROGRAM

İbrahim Türkmen 150140002, Kürşat Yaşar 150130064, Baran Kaya 150130032

This is the final report of the Gray Owls Group`s project. The project is about the creation of a program that finds the locations of local establishments and measures the distance between these locations and the user. The resulting array of type float and length between 0 and 1000 is then sorted using either Insertion Sort, Randomized Quick Sort or Merge Sort. The Gray Owls Group proposed a series of experiments to ascertain which algorithm would be the most suitable for our program. In the experiments we sent 20 arrays, the first consisting of 50 elements and each array larger by 50 than the last, into each of the sorting algorithms and recorded the time it took to sort the arrays. The results of the experiments prove that Merge Sort is the best option when the array size is smaller than 675, if larger then Quick Sort is faster, and in all cases Insertion Sort is the slowest. The program has been completed and the project has been successful, our group wishes that society would benefit from an easy-to-use and convenient program. The technical details and background information are provided in this paper.

GRAY OWLS GROUP
163 Charming str, Dinosaur Ave, Uskudar, Istanbul

Contents

Introduction

This document was written for the purpose of reporting the completion of the Gray Owls` project to all of our fellow engineers and programmers who are interested in this field and to whomever else it may concern. To those who have forgotten or are not familiar with, the Gray Owls` project was to create a program on windows or android platforms that pinpoints the locations of establishments or hotspots that the user inputs and then to present these locations in the ascending order of distance from the user. The projects main goal is to benefit the community (really we are not doing this for the cash -_o) by providing free advertising to many overlooked establishments and to help people who are searching for a particular establishment. Our group was not certain on which sorting algorithm our program should use once it finds the locations of the desired establishments and calculates the distances from the user. Hence an experiment to determine which algorithm is the most suitable for this task was proposed. Even though; theoretically, there is only a few second difference between the sorting algorithms we believe that our customers, being the lazy and impatient creatures they are, would appreciate only the best. This few second gain will undoubtedly put our program at the top. The completion of the experiments and the project will stimulate global economy and bring in financial resources to our group and to our pockets.

The rest of the report will entail background information of our project including a literature review, the methods and conditions we used to conduct our experiments, the technical results of our experiments, an analysis of the results, a table inferring the work packages and responsibilities of each team member and a conclusion of the overall project.

Background

With the invention and globalization of the smart phone the Computer Age reached a new milestone, people can do nearly everything they can do on a PC with a small pocket sized gadget. As Zickuhr said, Phones, laptops and desktops are becoming more popular throughout the modern age and are owned by a majority of the population, with younger adults preferring laptops and cellphones for mobility (2011) [1]. With such a boom in convenience the opportunities to make money is nearly endless, one of such opportunities is to create a helpful app. According to Viswanathan, the invention of new mobile electronics and the creation of their OS have increased the number of app developers and caused industries to give importance to Mobile development and marketing (2017) [2]. From this we can conclude that program creation for mobile devices has a prominent place in our modern society. The Gray Owls Group would like to exploit these opportunities in the interest of contributing to society and the economy (definitely not because we want to commit identity theft or anything), so the only question remains is what kind of app should we aim for?

Amongst all the possible options like a game app, a music app, an app that tells you how pretty you are and so on, we decided to lean towards a post-modern approach and chose to create an establishment finding app. No one has developed an app such as this before and given how there is a direct correlation between a town’s obesity density and a town`s total number of smartphone addicts, we judged that an app that tells people how close they are to a restaurant or the like would certainly be appealing. Half way through our project we encountered a major problem:

Which sorting algorithm should we use to sort the distances of the local establishments from the user? In order to answer this question correctly we proposed a series of experiments to help us confirm the performance of Insertion Sort, Merge Sort and Randomized Quick Sort under the conditions unique to our program.

As mentioned before in our proposal report, sorting algorithms may perform differently for different input types; Alex Allain said (2011), "Many algorithms that have the same efficiency do not have the same speed on the same input." [3] Using integer or float type for the input array can change the sorting algorithms performance. Many CPUs can handle integer operations faster than the float operations. Today's CPUs' have IEEE-754 floating point standard for floats or doubles. Anderson, Hagerup, Nilsson and Raman stated that IEEE 754 floating-point standard's aim is sorting floating-point numbers like integers with using integer sorting subroutines and essentially all floating-point numbers sorts using multiple-integer sort operations (1995)[4]. Hence floating-point sorting operations use ALU at least twice; one for mantisa part and one for exponential part so these operations spend more time than integer operations. Generally using float-point numbers over integers increase the computing time for sorting therefore, float numbers operations take more time than the integer numbers operations. With respect to this information in order to fully grasp the correct algorithm to use we deemed it necessary to field test Insertion, Merge and Quick sort`s performance when the input type is a float.

According to Nayak, Wason and Mudgal while some sorting algorithms work well for specified situations, they may not be the most appropriate solution for other situations. Because of this all sorting algorithms are suitable for different types of problems. Some sorting algorithms suitable for huge number of data, some of them practical to work with less number of elements (2014). To find best sorting algorithm, problem should be well evaluated and suitable algorithm should be chosen. Size and order of data and input type are some of the most important factors in selecting appropriate sorting algorithm. Karunanithi stated that insertion sort is useful for small lists and mostly sorted lists. In analyzing of this algorithm, while computational complexity is O(n^2) for elements that are in reverse order, if the array is already sorted computational complexity is equal to O(n). Because merge sort uses the divide and conquer approach, size and order of array does not affect computational complexity(O(nlogn)). One disadvantage of merge sort is it needs twice as much additional memory. Unlike merge sort, Quick sort does not require any different memory space to sort. Although worst case computational complexity of quick sort is O(n^2), quick sort is the best sort between algorithms which have O(nlogn) time complexity and it is more efficient for huge data sets (2014). According to this information, insertion sort is more efficient for smaller arrays. Merge sort is more usable for the systems that does not have much memory space and quick sort algorithm is more suitable for big data sets.

Methods

As mentioned in the project proposal, the setup of the experiments is as follows:

- Operating System: Windows 10
- RAM: 6GB
- Compiler: Dev C++ 11, with option : -std 11
- Language: C++

- Use of the library ‘Chrono’ for accurate measurements (nanoseconds)

It goes without saying that under standard conditions the performance of these algorithms are already known, however the conditions that our program produces is unorthodox, therefor experiments to determine how each sorting algorithm would behave when the input type is float and when the array is mysteriously ordered like our program`s array, and which sorting algorithm would be most suitable for our program must be conducted.

We chose to test insertion sort because when the array is nearly ordered, then insertion sort overwhelms all others (O (n^2, θ (n^2), Ω (1)) We chose Merge Sort because regardless of the array the time would be θ(nlogn). Randomized Quicksort was also nominated because in the worst case it behaves as slow as insertion sort (although this case is very unlikely), regularly it behaves like merge sort and in the best cases in behaves like a linear or constant algorithm.

We predicted that insertion sort would be the worst of the three and that quick sort and merge sort would be nearly the same. The results of the experiment proved us to be half right.

We started our experiments by inputting the user`s coordinates to a program that calculates the distance, using Pythagoras Theorem, from the inputted coordinates to a desired number of predetermined hypothetical coordinates that represent the locations of the desired establishments and then sorts these distances using a specified sorting algorithm. First we considered only the first 50 coordinates, and then the first 100 and so on until we reached 1000.

Ibrahim Turkmen, Kurshat Yasar and Baran kaya used this method to determine the performance of Insertion Sort, Merge Sort and Quick sort respectively

Figure 1 shows the pseudocode for insertion sort, Figure 2 shows the pseudocode for quick sort and Figure 3 show the pseudocode for merge sort:

```
I → 2
While I < n
        Key → array[I]
        j → I – 1
        While j >= 1 && array[j] > j
                Switch array[j] with array[j+1]
                j → j-1
        I → I + 1
```

Figure 1. Pseudocode of Insertion sort. Arrow signs correspond to the equal sign.

```
Quicksort ( int b, int e )
If ( b == e )
return
r → <random number between b and e>
switch array[r] with array[e]
l, j → b
While j < e
        If array[j] <= array[e]
                Switch array[j] with array[l]
                l → l + 1
        j → j + 1
switch array[l] with array[e]
quicksort ( b, l-1 )
quicksort ( l+1, e )
```

Figure 2. Pseudocode for Randomized Quick Sort. Made by Baran Kaya

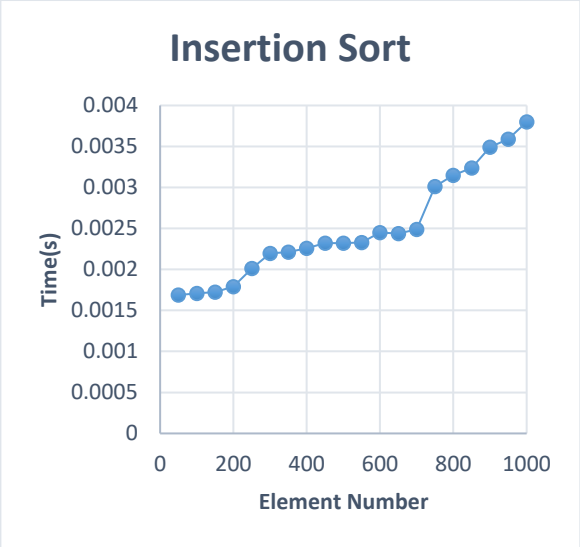
```
func merge( var a as array, var b as array )
    var c as array
    while ( a and b have elements )
        if ( a[0] > b[0] )
            add b[0] to the end of c
            remove b[0] from b
        else
            add a[0] to the end of c
            remove a[0] from a
    while ( a has elements )
        add a[0] to the end of c
        remove a[0] from a
    while ( b has elements )
        add b[0] to the end of c
        remove b[0] from b
    return c
end func
```

```
func mergesort( var a as array )
If ( p < r )
if ( n == 1 ) return a
    var l1 as array = a[0] ... a[n/2]
    var l2 as array = a[n/2+1] ...
a[n]
    l1 = mergesort( l1 )
    l2 = mergesort( l2 )
   merge( l1, l2 )
return
end func
```

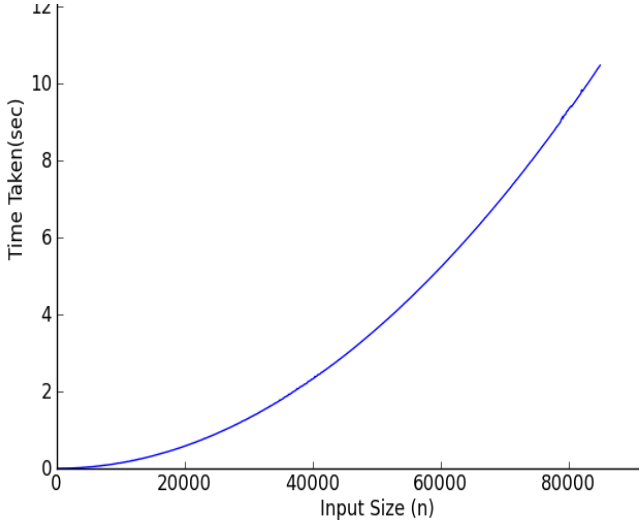
Figure 3. Pseudocode of Merge Sort. Made by Kürşat Yaşar

Once we obtained the results we made use of our brain cells to determine which sorting algorithm would be the most effective for our program.

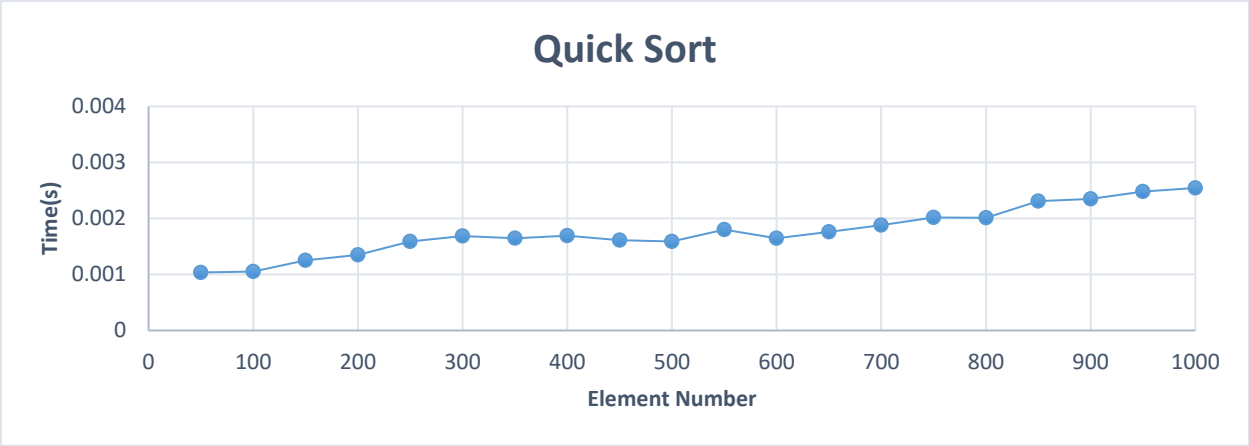
Results and Data



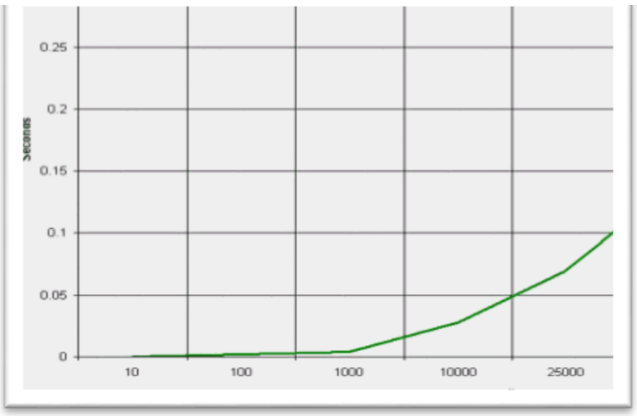
Graph 1 Insertion Sort by Ibrahim Turkmen. For more accurate perceptions see Table 1 on the attached Data Sheet documents.



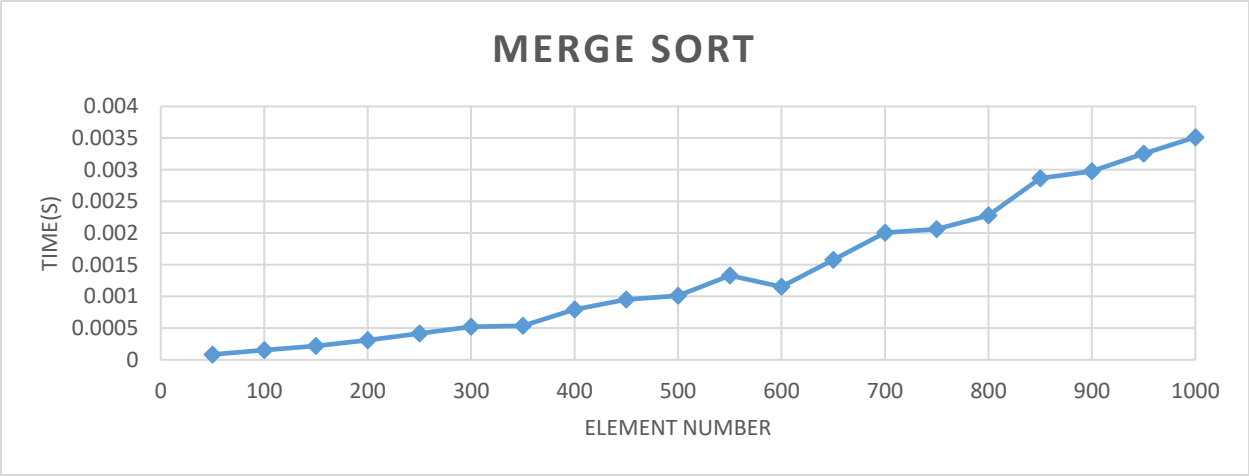
Graph 2 Insertion Sort when the input type is the integer type and when the array is in decreasing order.



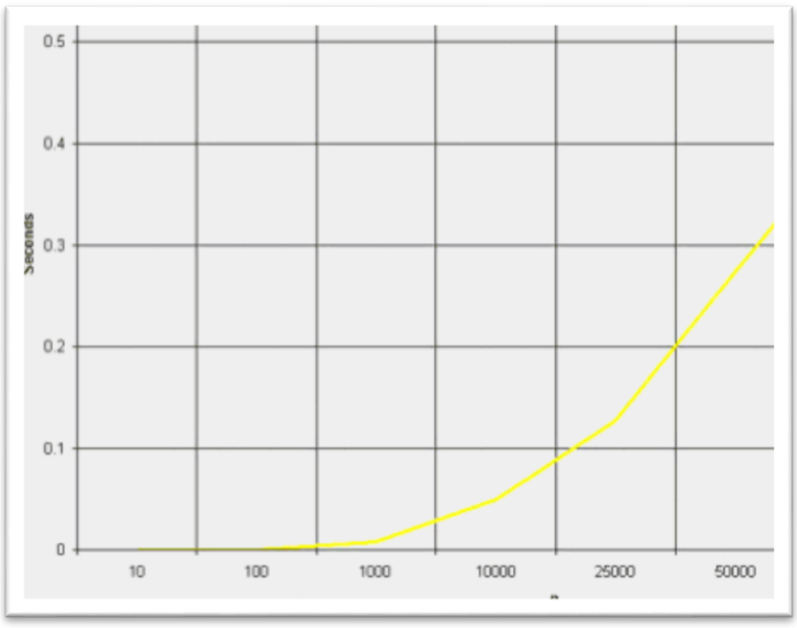
Graph 3 Quick Sort by Baran Kaya. See Table 1 on attached Data Sheet



Graph 4 Normal Quick Sort

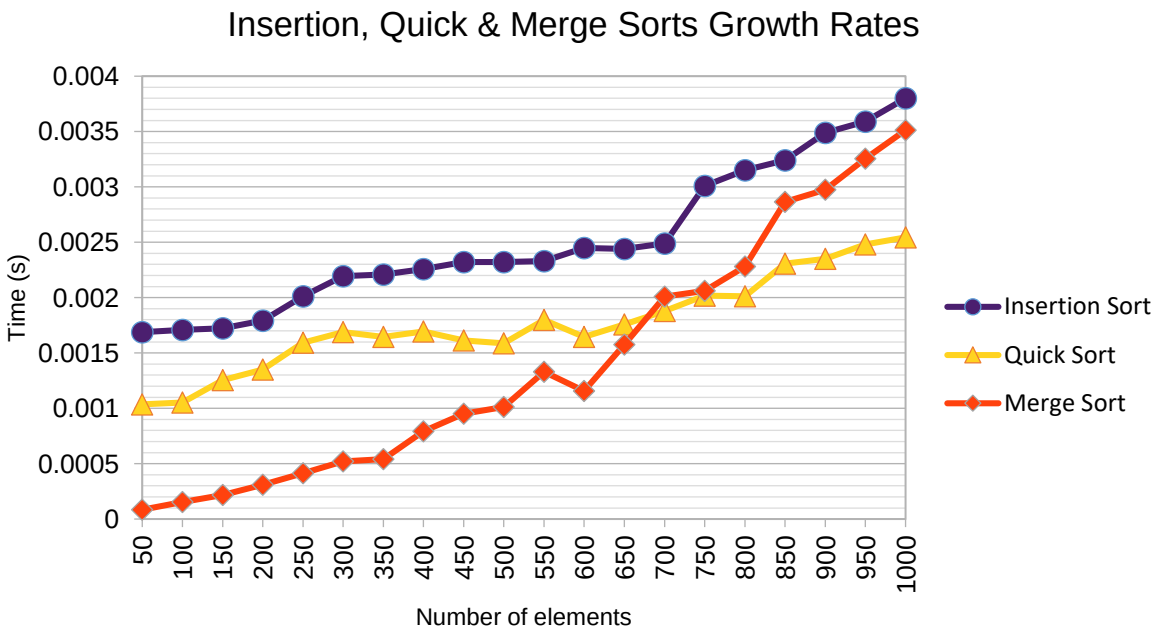


Graph 5 Merge Sort by Kursat Yasar.



Graph 6 Real Merge Sort

These graphs are project experiment results. Ibrahim made the insertion sort experiments (graph 1 and 2), Baran made the quick sort experiments (graph 3 and 4) and Kürşat made the merge sort experiments (graph 5 and 6). Ibrahim and Kürşat ran their program once to get the results, but Baran ran his quick sort code 5 times and calculated their average for more clear results.



Graph 7. Comparison of all the sorting algorithms.

Analysis of the experiment results

Under normal circumstances Merge Sort overtakes Quick Sort at any given input size, but as it can inferred from Graph 7 when the input size reaches a certain point (675) Quick Sort performs better than Merge Sort. This result was unexpected by us which was why conducting these experiments was definitely worthwhile. We believe the cause of this outcome to be because Merge Sort takes a longer time than usual to create subarrays since the input type (float) is larger than the integer type, Quick Sort however is not as limited by this change. As we expected Insertion Sort is the slowest of the three with around 2-5 times more time necessary to sort an array, which is disappointing.

From here we are torn between two options. The first option is to slightly modify the program to use Merge Sort when the array size is smaller than 675 and to use Quick Sort when otherwise. The second option is to rewrite the fundamentals of our program to produce a slightly ordered array and then repeat the experiments with these new program, hoping that Insertion Sort produces the best results. However, our team strongly believes that if we change the program too much then more time would be wasted on the pre-sorting phases.

Project Evaluation

The project overall was a success, the experiments were conducted ethically and swiftly, the experiment results provided fruitful insight and the program has been successfully created. The beta has been scheduled to launch at 32nd of July, 2017. Table 1 shows the schedule that our group displayed on our project proposal. The group stayed faithful to the schedule.

Figure 4. Shows the responsibilities that were assigned to one or multiple group members. The members MOSTLY fulfilled their assigned tasks without any whining or complaining. The

responsibilities allocated may slightly differ from what was earlier proposed due to some unexpected developments.

	Ibrahim Turkmen	Baran Kaya	Kürşat Yasar
22/03/2017	MEETING		
24/03/2017	PROPOSAL DEFENCE		
25/03/2017	SPRING BREAK		
12/04/2017	Insertion Sort experiment Start	Merge Sort experiment Start	Quick Sort experiment Start
14/04/2017	Progress Report	Progress Report	Progress Report
5/05/2017	PRESENTATION		
19/05/2017	FINAL REPORT		

Table 1. Gray Owls Group Activities

Experiment Setup	**---**	**Completed by Turkmen**
Insertion Sort Exp.	**---**	**Conducted by Turkmen**
Quick Sort Exp.	**---**	**Conducted by Kaya**
Merge Sort Exp.	**---**	**Conducted by Yasar**
Meetings	**---**	**Scheduled by Kaya**
Presentation	**---**	**Presented by Turkmen, Kaya and Yasar**
Experiment results	**---**	**Compiled and Documented by Kaya**
Methods of the Exp.	**---**	**Produced by Yasar and Tukmen**
Sustenance	**---**	**NOT PROVIDED BY KAYA!!**

Figure 4. Responsibilities that were allocated to each team member and whether or not they have been fulfilled.

The project did not proceed without hurdles or setbacks; one of our members, Kusat Yasar, was struck by misfortune when his leg was injured during one of his extreme physical endeavors debilitating him from working for several weeks, as if this was not unfortunate enough two mosquitos flew into Turkmen`s room while he was conducting his experiments and were tragically flattened like a pancake on a highway.

Despite these unforeseeable incidents, the experiments proceeded smoothly and the project has been successfully completed. The positive impact that it will make on society is awaited with anticipation and homicidal impatience.

The results of our experiments is currently being edited and is scheduled to be published by Mars Magazine on 23/11/2100 (as emphasized in the proposal, this is not a typo it is the soonest available date).

Conclusion

Thanks to our experiments we were able to conclude that insertion sort, merge sort and quick sort perform differently when the input type is a float and when the nature of the array is like how our program produces them. If we had not the glorious wisdom to propose an experiment then our program would have been a little slower if we had proceeded with insertion sort. The program has been completed and has been rated 4/5 stars by our mothers. Our Group is very satisfied with our hard work and is optimistic about the future.

References

[1]Zickuhr, K. (2011, February 3). *Generations and their gadgets.* Retrieved from:

http://www.pewinternet.org/2011/02/03/generations-and-their-gadgets/

[2]Viswanathan, P. (2017, February 23). *Is it really profitable to develop a Mobile app?* Retrieved from: https://www.lifewire.com/is-it-profitable-to-develop-a-mobile-app-2373416

[3] Allain, A. (2011). *Sorting Algorithm Comparison.* Retrieved from http://www.cprogramming.com/tutorial/computersciencetheory/sortcomp.html

[4] Anderson, A., Hagerup, T., Nilsson, S. & Raman, R. (1995). Sorting in Linear Time?. Las Vegas, Neveda, USA: ACM

https://www.deepdyve.com/lp/acm/sorting-in-linear-time-i00PCu1b7t?key=acm

[5]Nayak, P., Wason, R. & Mudgal, S. (2014). Sorting algorithms. *International Journal of Innovative Research in Technology*, *1*(11), p. 272

[6] Karunanithi, A. (2014). *A survey, discussion and comparisionof sorting algorithms* (Master's thesis). University of Umea, Umea, Sweden