VISUALIZING SORTING ALGORITHMS

Souradeep Ash, Badal Kumar Singh, Pallab Mondal

Mentor - Dipta Mukherjee

The Department of Computer Science and Engineering

University Of Engineering and Management Jaipur

2021

Abstract

This paper discusses a study performed on animating sorting algorithms as a learning aid. Algorithm visualization technology graphically illustrates how algorithms work. A web-based animation tool was created to visualize sorting algorithms like: *Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Slow Sort, Tree Sort, Wave Sort, etc.* The animation tool would represent data as a Bar-graph, Pyramid, Dots, Dot-Spiral, many more and after selecting a data-ordering and algorithm, the user can run an automated animation or step through it at their own pace. This visualization can be used to explain how all data move to the proper position in order to be sorted in a display computer for education. The algorithm visualization indicates that students increase their motivation and ability to program variety of sorting in programming language they learn. During the testing step, the application is run and checked to confirm that it performs exactly what the author has intended and the students can learn selection sorting algorithm by studying the visualization

Introduction:

A. Sorting:

In computer science and mathematics, a sorting algorithm is an algorithm that puts elements of a list in a certain order [1]. The most-used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) that require sorted lists to work correctly; it is also often useful for canonicalizing data and for producing human-readable output. More formally, the output must satisfy two conditions: (1) The output is in non-decreasing order (each element is no smaller than the previous element according to the desired total order); and (2) The output is a permutation, or reordering, of the input.

Since the early of computing, the sorting problem has attracted a great deal of research, perhaps due to the complexity of solving it efficiently despite its simple, familiar statement. Sorting algorithms are prevalent in introductory computer science classes, where the abundance of algorithms for the problem provides a gentle introduction to a variety of core algorithm concepts, such as big O notation, divide and conquer algorithms, data structures, randomized algorithms, and case analysis.

B. Visualization:

Visualization is presenting data or information that used in science, engineering, medical, business, and others. Numerical simulations often produce data files that is contains data values. The data are converted to a visual form that is helpful for the user in analysing his/her problem. Visualization for science, engineering, and medical presents graphical form information presentation. And the term business visualization is used in connection with data sets related to business, management, social, industry, and other non-scientific areas.[2]

This repository is a demo of visualizing 26 types of Sorting Algorithms. It aims to make Sorting Algorithms easier for programmers to understand. Also, you can see the difference of Time Complexity between different sorting algorithms. Not only this you can find 12 kinds of visualizer effect like colored pillars, colored dots, colored circle of dots and many more and this make it different from any other sorting visualizer. This would make it distinct from the rest of the data as it is being shifted by the algorithm from its original, unordered position to its final ordered position.

BUBBLE SORT,INSERTION SORT,SELECTION SORT,QUICK SORT,DOUBLE SELECTION SORT,MERGE SORT,COUNTING SORT.GRAVITY SORT/BEAD SORT,BITONIC SORT,TREE SORT,INTRO SORT/INTROSPECTIVE SORT,HEAP SORT,RADIX SORT,SHELL SORT,COMB SORT, QUICK-MERGE SORT,WAVE SORT,GNOME SORT,COCKTAIL-SHAKER SORT,COCKTAIL-MERGE SORT,ODD-EVEN SORT,(BATCHER)ODD-EVEN MERGE SORT,SLOW SORT,STOOGE SORT,BOGO SORT,CUBE SORT.

Keeping in line with the example of sorting people by age, let's pretend that you have printed the age of each person on a separate index card. One way to go about organizing the cards is to first find the smallest age in the pile and bring it to the front. Then, find the next smallest and place it behind the already ordered first age. Eventually, you will end up with a pile of index cards that list the ages in ascending order. This method is exactly how Selection Sort works, where to sort a set of data, you select the smallest first, and then the next smallest and the next smallest. This algorithm is not very difficult to understand by word of mouth, but more abstract sorting algorithms, such as how Quick Sort requires moving data around a pivot point, may not be intuitive to read through.

Our main goal is to develop algorithm visualization of Sorting Algorithm. The visualization show how all data move to the proper position in order to be sorted. It can be easily understood by students how the algorithm should be implemented in coding.

Existing Work:

Over the year, there have been many studies and papers on the use if sorting algorithms as visual aids. Some are comprehensive views on how to create animations and perform statistical analysis, and others focus on different techniques aimed for increased understanding of a similar animation. There are lots of figure in researching animation like John T. Stasko, Andreas Kerren and many more.

The paper "Algorithm Animation" by A. Kerren and J. Stasko is step-by step guide to analysing the environment, means and available coding methods to use a sorting animation.[3] Many different types of software are listed to be used for animation, one of which was BLASA was created by Marc Brown In addition, the paper notes some limitations to its applicability that I have not undertaken.[4] For example, some factors that the study addressed were the availability of appropriate students, a trade-off of fairness regarding selective academic success, and procedure-specific questions for post-test design to promote quantitative results and many software are available for visualize the sorting algorithms like VISUALGO by Dr Steven Halim[5]. It invented in 2011 by him as a tool to help the students to better understand about algorithms, "Sorting Visualizer"by Clement Mihilescu[6], "sort-visualizer" by Ramiz Rahman[7], "Sound-of-Sorting" by Timo Bingmann[8], etc that inspire us lot to make this visualizer better.

Our Work:

Learning is a necessity in life, Nowadays, due to globalization and proliferation of technology, computer education in India becomes great demand for students. One of the favourite subjects in computer education is Data Structure and Algorithm (DSA), which is now held in many courses as main subject. You love algorithm because algorithm combining with data structure solving the problem and not only this in every sector in programing you need "Algorithm's" to solve problems faster and sorting are one of the key thing to sort an array in increasing or decreasing order to get the maximum and minimum value element . Accept from this, in education if you visualizing things than it is best to understand and it will recognise those things in needy time. What motivate me most to make this project is to know more about sorting algorithms and understanding there work like how they sort an array in different types like in Quick sort we chose a pivot and according to pivot we place the minimum value on left-side and maximum value on right-side of the array. On the other hand if we see Merge Sort we observe that it divide the array into two. We don't need any pivot in this case. Every sorting algorithm have there different ways to sort an array.

This project "Sorting Visualizer" is a very simple UI and it allows the user by selecting the sorting algorithm, select the speed, amount of blocks, size, different types of animation of the visualization using only HTML5, JavaScrip and Css.

Design:

The User-Interface:

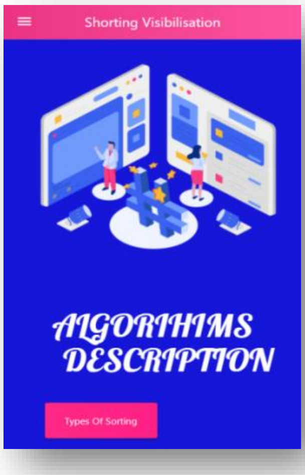
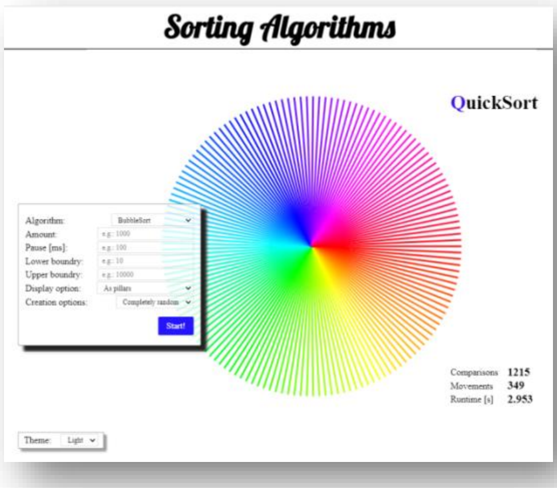


Figure 1: Representing   the user interface of sorting visualizer.

When user will open the project he can see a normal web page [Figure 1] where user can see one component "Types of Sorting" and accept from this user can see 3 lines on upper left corner of the web page, by clicking on that user can have two option "Visibilize" & "Creaters" if user click on the Visibilize user can direct to the main page where user can perform the project animation and see how sorting works visually.

How it Works:

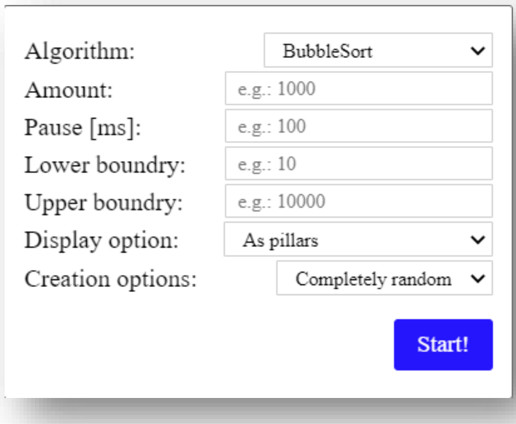
Even though the underlying back-end code went through a drastic refactor midway through the implementation, the overall design and layout of the components has remained the same. The interface has ten components: area, nine control buttons, and a theme changer Light/Dark button.



Algorithm:	BubbleSort
Amount:	e.g.: 1000
Pause [ms]:	e.g.: 100
Lower boundry:	e.g.: 10
Upper boundry:	e.g.: 10000
Display option:	As pillars
Creation options:	Completely random

Comparisons	1215
Movements	349
Runtime [s]	2.953

Theme: Light



Algorithm:	BubbleSort
Amount:	e.g.: 1000
Pause [ms]:	e.g.: 100
Lower boundry:	e.g.: 10
Upper boundry:	e.g.: 10000
Display option:	As pillars
Creation options:	Completely random

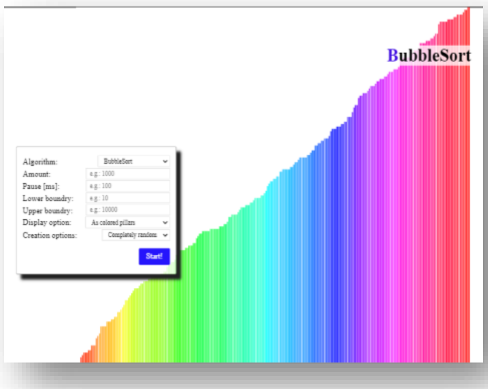
Figure 2: screenshot of webpage after loading.

Figure 3: user input for various sorting types.

The middle section that shows the colored circle is the canvas area and it change according to your choice using the drop and down button “Display option” and it change according to your choice like pillars, colored pillars, dots, colored dots, pyramid, colored pyramid, spiral of dots, colored spiral of dots, colored spiral of pillars, colored circle, colored circle of dots etc.

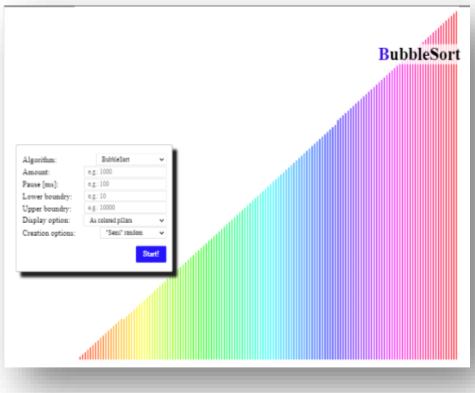
We can see there is a drop and down button named “Algorithm” can help you to chose 26 kinds of sorting algorithms named Bubble sort, Insertion sort, Selection sort, Double selection sort, Quick sort, Merge sort, Counting sort, Gravity sort, Bitonic sort, Tree sort, Intro sort, Heap sort, Radix sort, Shell sort, Comb sort, Quick-Merge sort, Wave sort, Gnome sort, Cocktail-shaker sort, Cocktail-Merge sort, Odd-even sort, (Batcher)Odd-even sort, slow sort, stooge sort, Bogo sort, cube sort etc, and u can chose any one of them and use it to understand how a particular algorithm works.

Not only this you can find different kinds of buttons like “Amount”, ”Pause[ms]”, ”Lower boundary”, ”Upper boundary”, ”Theme” through you can change the size, amount of pillars, lower boundary and upper boundary size , theme and accept from this another option is there named “Creation option” that can help u to chose whether your pillars, dots ,etc .



Algorithm: BubbleSort
Amount: e.g.: 1000
Pause [ms]: e.g.: 100
Lower boundry: e.g.: 10
Upper boundry: e.g.: 10000
Display option: As colored pillars
Creation options: Completely random

Figure 4: Completely random(it will chose your values randomly for that outcome will be not in a perfect shape)



Algorithm: BubbleSort
Amount: e.g.: 1000
Pause [ms]: e.g.: 100
Lower boundry: e.g.: 10
Upper boundry: e.g.: 10000
Display option: As colored pillars
Creation options: "Semi" random

Figure 5: “semi” random(it will help you to chose your values randomly but in such way for that your outcome is perfect)

I have tried to keep the interface as simple as possible and grouped related buttons together to combat confusion in usability. On right side we can see three things “Comparisions”, ”Movements” and ”Runtime[s]” that will show you everything according to your amount, speed and the algorithms that you will chose. After selecting algorithm and display option and set the amount of pillars, dots ,etc when you click on the “start” button the visualizer will start working and you can easily understand how this visualizer work.

In this project user can find lots of visualizing effect like:

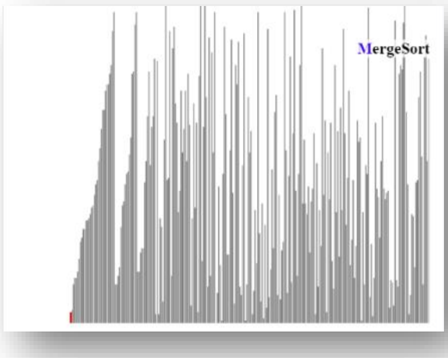


Figure 6: pillar

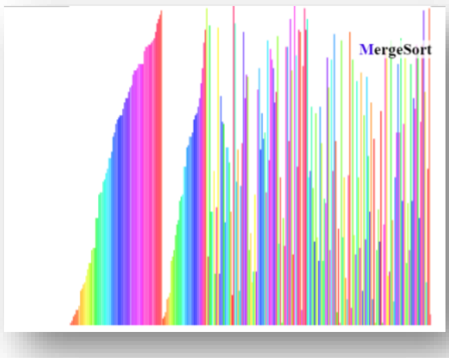


Figure 7: coloured pillar

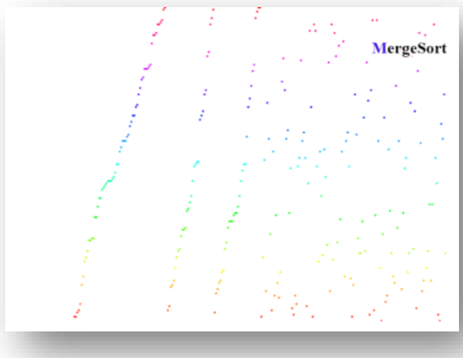


Figure 8: colour dot

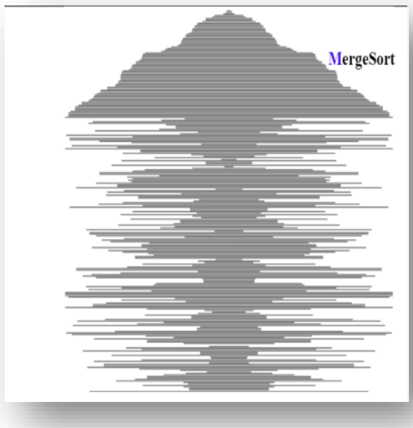


Figure 9: Pyramid

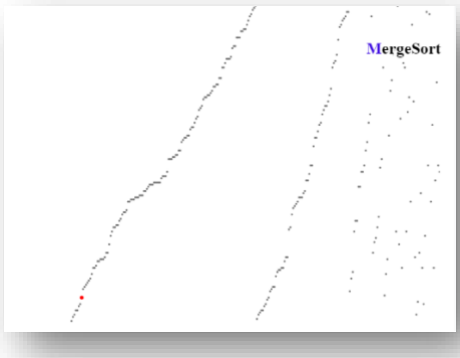


Figure 10: Dot

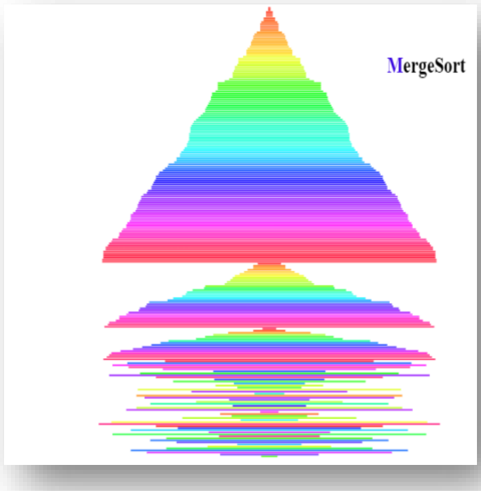


Figure 11: Coloured Pyramid

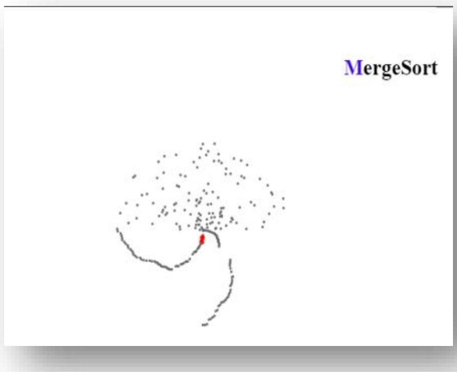


Figure 12: Spiral Dots.

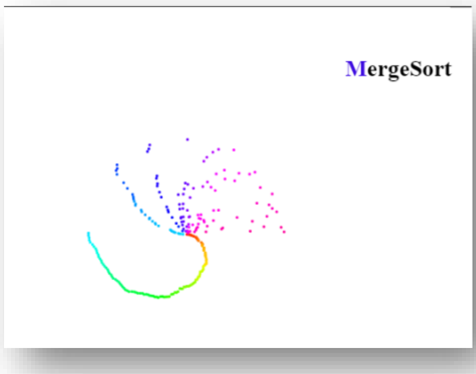


Figure 13: Coloured Spiral Dots.

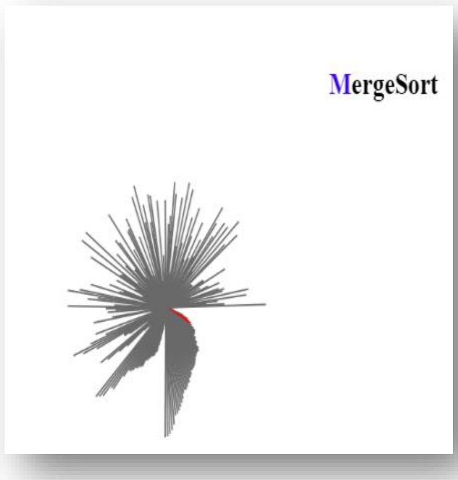


Figure 14: Spiral Pillars

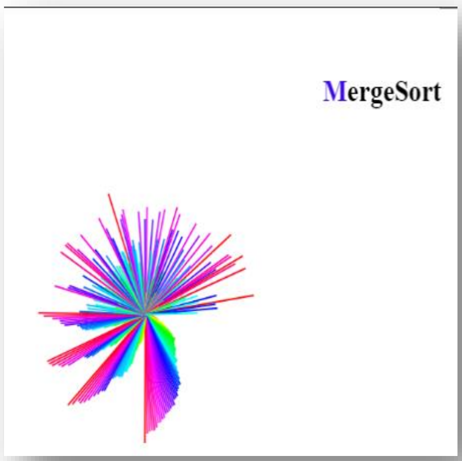


Figure 15: Colored Spiral Pillars

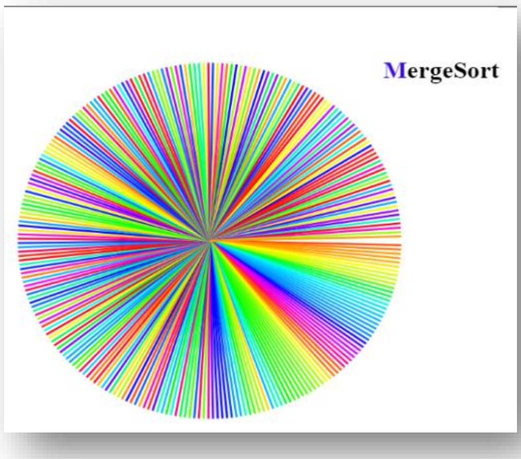


Figure 16: Coloured Circle

This all effect will help user to understand this visualizer perfectly and help user to know more about this.

Sorting Algorithm's that we used:

BUBBLE SORT: The algorithm, which is a comparison sort, is named for the way smaller or larger elements "bubble" to the top of the list[9]

Time Complexity

i)Best: O(n)
ii)Average: O(n^2)
iii)Worst: O(n^2)
Worst Case Space Complexity-O(1)

INSERTION SORT: A simple sorting algorithm that builds the final sorted array (or list) one item at a time.[9]

Time Complexity
i)Best: O(n)
ii)Average: $O(n^2)$
iii)Worst: $O(n^2)$
Worst Space Complexity-O(1)

SELECTION SORT: It is an in-place comparison sorting algorithm. It has an O(n2) time complexity, which makes it inefficient on large lists, and generally performs worse than the similar insertion sort.[9]

Time Complexity
i)Best: $O(n^2)$
ii)Average: $O(n^2)$
iii)Worst: $O(n^2)$
Worst Space Complexity- O(1)

QUICK SORT: It is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller array.[9]

Time Complexity
i)Best: O(n logn)
ii)Average: O(n log n)
iii)Worst: $O(n^2)$
Worst Space Complexity-O(log n)

DOUBLE SELECTION SORT: An elementary sorting algorithm that is designed to minimize the number of exchanges that are performe[9]

Time Complexity
i)Best: O(n)
ii)Average: $O(n^2)$
iii)Worst: $O(n^2)$
Worst Case Space Complexity-O(1)

MERGE SORT: It is an efficient, general-purpose, and comparison-based sorting algorithm. [9]

Time Complexity
i)Best: O(n log n)
ii)Average: O(n log n)
iii)Worst: O(n log n)
Worst Space Complexity-O(n)

COUNTING SORT: It is a linear time sorting algorithm which works faster by not making a comparison.[9]

Time Complexity
Worst: O(n + k),where k is the range of the non-negative key value
Worst Space Complexity- O(n + k)

GRAVITY SORT/BEAD SORT: Gravity is a sorting algorithm that is based on the phenomenon of gravity .Data is arranged in an abacus form with rows and columns represent the numbers in the unsorted list[9]

Time Complexity
i)Best: O(1)
ii)Average: $O(n^{1/2})$
iii)Worst: O(n)
Worst Space Complexity- $O(n^2)$

BITONIC SORT: Bitonic merge sort is a parallel algorithm for sorting. It is also used as a construction method for building a sorting network.[9]

Time Complexity
i)Best: $O(\log^2(n))$
ii)Average: $O(\log^2(n))$
iii)Worst: $O(\log^2(n))$
Worst Space Complexity- $O(n \log^2(n))$

TREE SORT: It is a sort algorithm that builds a binary search tree from the elements to be sorted, and then traverses the tree (in-order) so that the elements come out in sorted order.[9]

Time Complexity
i)Best: O(n log n)
ii)Average: O(n log n)
iii)Worst: $O(n^2)$
Worst Space Complexity- $\Theta(n)$

INTRO SORT/INTROSPECTIVE SORT: It is a hybrid sorting algorithm that provides both fast average performance and (asymptotically) optimal worst-case performance.[9]

Time Complexity
Worst: O(n log n)
Worst Space Complexity-O(n)

HEAP SORT: It is a comparison-based sorting algorithm. Heap sort can be thought of as an improved selection sort.[9]

Time Complexity
i)Best: O(n log n)
ii)Average: O(n log n)
iii)Worst: O(n log n)
Worst Space Complexity-O(1)

RADIX SORT: Radix sort is one of the sorting algorithms used to sort a list of integer numbers in order. Radix sort dates back as far as 1887 to the work of Hollerith on tabulating machines. Radix sorting algorithms came into common use as a way to sort punched cards as early as 1923[9]

Time Complexity
Worst: $O(w*n)$,where w is the key length
Worst Space Complexity-$O(w + n)$

SHELL SORT: The shell sort improves on the insertion sort by breaking the original list into a number of smaller sublists, each of which is sorted using an insertion sort. The unique way that these sublists are chosen is the key to the shell sort[9]

Time Complexity
i)Best: O(n log n)
ii)Average: Depend On Graph
iii)Worst: $O(n^2)$
Worst Space Complexity-O(1)

COMB SORT: Comb sort is a relatively simple sorting algorithm originally designed by Włodzimierz Dobosiewicz and Artur Borowy in 1980, later rediscovered (and given the name "Comb sort") by Stephen Lacey and Richard Box in 1991. Comb sort improves on bubble sort in the same way that Shell sort improves on insertion sort.[9]

Time Complexity
i)Best: O(n log n)
ii)Average: $O(n^2)$
iii)Worst: $O(n^2/2p)$
Worst Space Complexity-O(1)

WAVE SORT: Sort an array in a Waveform.[9]

Time Complexity
i)Best: O(n)
ii)Average: $O(n^2)$
iii)Worst: $O(n^2)$
Worst Space Complexity-O(1)

GNOME SORT: Put items in order by comparing the current item with the previous item. If they are in order, move to the next item (or stop if the end is reached). If they are out of order, swap them and move to the previous item. If there is no previous item, move to the next item. This sorting algorithm proposed by Iranian computer scientist Hamid Sarbazi-Azad in 2000. [9]

Time Complexity
i)Best: O(n)
ii)Average: $O(n^2)$
iii)Worst: $O(n^2)$
Worst Space Complexity-O(1)

COCKTAIL-SHAKER SORT: Cocktail Sort is a variation of Bubble sort.[9]

Time Complexity
i)Best: O(n)
ii)Average: $O(n^2)$
iii)Worst: $O(n^2)$
Worst Space Complexity-O(1)

ODD-EVEN SORT: Like bubble sort, odd-even sort works by iterating through the list, comparing adjacent elements and swapping them if they're in the wrong order. [9]

Time Complexity
i)Best: O(n)
ii)Average: $O(n^2)$
iii)Worst: $O(n^2)$
Worst Space Complexity-O(1)

(BATCHER)ODD-EVEN MERGE SORT: A related but more efficient sort algorithm is the Batcher odd–even merge sort, using compare–exchange operations and perfect-shuffle operation.[9]

Time Complexity
i)Best: O(n log n)
ii)Average: $O(n \log^2 n)$
iii)Worst: $O(n \log^2 n)$
Worst Space Complexity-$O(n \log^2 n)$

SLOW SORT: Slow sort is a sorting algorithm that is designed to be deterministically sub-optimal. The algorithm was published in 1986 by Andrei Broder and Jorge Stolfi in their paper Pessimal Algorithms and Simplexity Analysis where they expressed a bunch of very in-efficient algorithms.[9]

Time Complexity
i)Best: O(n log n)
ii)Average: $O(n^2)$
iii)Worst: $O(n^2)$
Worst Space Complexity-O(1)

CUBE SORT: It is a comparison-based sorting algorithm. Heap sort can be thought of as an improved selection sort.[9]

Time Complexity
Worst: O(n log n)
Worst Space Complexity-Θ(1)

These are the all Sorting algorithm that I found and implement in this project and this project takes us two months to Complete and manage

those things in a particular order.

Conclusions and Future Work:

Through much time and effort, We have successfully created a working web-based animation tool for visualizing the following sorting algorithms: Bogo Sort, Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Insertion Sort, etc. There remains, however, a strong mind-set to research and create animations like these to improve learning in the education, which We agree with completely. Learning how to code a web platform was challenging, and We thank the tutorials on W3Schools.com,Our Mentor and other online platforms for getting us there. We had a previous internship where We updated the JavaScript on a webpage, but it was much more concise and did not involve objects and HTML5 for visualizations. The good news is that JavaScript is still one of the most popular web languages, so, We are not too worried about another big refactor soon for a language update. Next would be to add more algorithm like path finding algorithm (ex-Dijkstais Algorithm, A* search Algorithm, etc) and make this web page more interesting for student for that they will use this. Finally, We would make the web tool public on our "GITHUB" for that any student can download and can update it with more things and fix the bugs if any. In Future we will try to improve the web and visualizer with sound track and make it more interesting to user for that user will love to use it.

Special Thanks:

Special thanks to our respected mentor and teacher, **Dipta Mukherjee**, for helping us kick start this project by creating a completion plan and for helping me organize this paper. Furthermore, I thank to my project partners Pallab and Badal for there feedback and recommendation for the code refactor when I was experiencing timer integration issues with my previous version of the animation tool. Also, I thank to our Seniors for there forward-thinking ideas on how to improve the animation tool.

Reference:

1. Sedgewick, Robert , (2001) Algorithms in C++, Third Ediition, Massachusetss: Addison-Wesley.
2. Hearn, Donald, and Pauline Baker, (1996) Computer Graphics C Version, 2nd edition. Upper Saddle River, NJ: Prentice Hall International, Inc.
3. A. Kerren and J. T. Stasko. (2002) Chapter 1 Algorithm Animation. In: Diehl S.(eds) Software Visualization. Lecture Notes in Computer Science, vol 2269. Springer, Berlin, Heidelberg. < https://link.springer.com/chapter/10.1007/3-540-45875-1_1 >
4. BLASA was created by Marc Brown
5. VISUALGO by Dr Steven Halim <https://visualgo.net/en/sorting>
6. "Sorting Visualizer" by Clement Mihilescu <https://clementmihailescu.github.io/Sorting-Visualizer>
7. "sort-visualizer" by Ramiz Rahman.< https://sort-visualizer.ramizrahman.com/>
8. "Sound-of-Sorting" by Timo Bingmann.< https://pathema.net/2013/sound-of-sorting>
9. Shorting algorithm used <https://en.wikipedia.org/wiki/Sorting>
10. J. Stasko. Using Student-built Algorithm Animations As Learning Aids. Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education. SIGCSE '97 (San Jose, California), pages 25-29. < https://dl.acm.org/doi/10.1145/268085.268091 >
11. T. Bingmann. "The Sound of Sorting - 'Audibilization' and Visualization of Sorting Algorithms." Panthemanet Weblog. Impressum, 22 May 2013. Web. 29 Mar. 2017. < https://github.com/bingmann/sound-of-sorting >
12. "SELECTION SORTING ALGORITHM VISUALIZATION USING FLASH" by Hadi Sutopo. <https://airccse.org/journal/jma/3111ijma03.pdf>
13. Ypenburg, Derrick, (2009) ActionScript 3.0, Berkeley, CA: Peachpit Press. < https://www.bookdepository.com/ActionScript-3-0-Derrick-Ypenburg/9780321564252 >
14. Franklin, Derek & Jobe Makar, (2002) Macromedia Flash MX ActionScripting Advanced Training from the Source, Berkeley, CA: Macromedia Press. <https://www.researchgate.net/publication/234790666_Macromedia_Flash_MX_2004_ActionScript_Training_from_the_Source >