**Sorting Algorithms using JAVA**

**CSCI 334**

**Research Paper**

**Aashish Gopali**

**Abstract:**

Sorting algorithm has attracted a lot of computer scientist as it has been the first step to a lot of complex algorithms. In addition, this has numerous applications on mathematics, computer science and many other related fields. In first i first define sorting and then i provide criteria to compare sorting algorithms and then i compare 5 sorting algorithms using those criteria. While comparing i also provide a complete JAVA code for the sorting algorithm and provide complete demonstration. And then i conclude which sorting algorithm will provide the best result. For doing this i create a random number of integers in java, and i will have different size of array and those random values will be then kept in a file and then copied to an array. Then that array will be sorted with all 5 sorting algorithms.

After sorting all five-sorting algorithm i will give the conclusion. The work is dedicated to finding the time complexity, data usage and stability of the algorithm. With the time taken i will create a graph and show how the time complexity works when the data keeps increasing.

**Introduction:**

Sorting is one of the most important things that i use in our daily life. Sorting helps us to keep the things together and in a managed order. Can you think of a dictionary where the words are not sorted? How much time will it take to find word which you want to search in that dictionary? If i calculate and if the word is at the end of dictionary, then it will take the time to go through every word. That sounds complicated right. Guess the similar thing on phonebook records and many more, if i did not had sorting then it would have been tough in our life. Similar thing goes with the sorting in computer programs. I need sorting methods for searching, removing a data and a lot. Today i will see how the sorting algorithms works and i compare 5 different sorting algorithms on their stability, time complexity and memory usage.

Before directly hopping into the algorithms, i should first work on the comparing criteria. The first and one of the vital criteria is the time complexity. Time complexity is the computational complexity(Lavore). In this, the time taken for a certain process to completed is calculated. Even though the computer system is fast on small scale. Hoiver, when i sort a data of million size the time complexity matters a lot. For example, which searching a data in google it takes a fraction of seconds and provides us all the things it finds on the ib. What if it takes 1 hour to find a single data, how would the life be? This example shows how important is the time complexity. Next, i will learn about stability of a sorting algorithm.

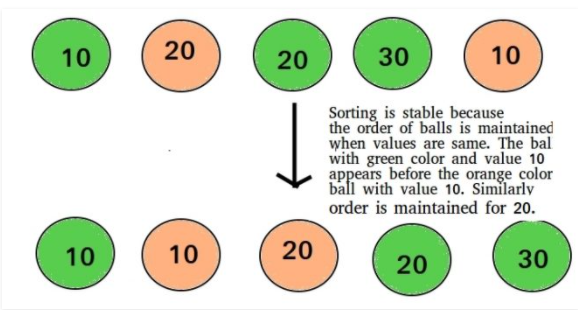
Stability word itself defines itself. It is how stable is the sorting algorithm. This is important when i are sorting the arrays which have some similar data. On this case, the sorting algorithm should sort as how the data is in present in the input(Sorting Algorithms).. In the figure 1, i can see how the two balls with same value are sorted in the way they appear in data. The green 10 comes earlier than the orange 10. That is why it is sorted and placed before orange 10. The sorting algorithms should be stable whether there is similar value or not. Now, let’s see what if the sorting algorithm is not stable. In figure 2, there are cards which are sorted and there is difference when a sorting algorithm is stable and is unstable. Next, i will have a look at another important criteria which is the memory usage.

Figure 1: Stability(Geeks)

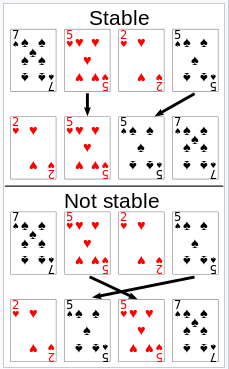


Figure 2 1Stable vs Unstable

Memory usage is one of another important that is not just important in sorting algorithms but in every programs. Memory is where i keep the data that is needed to run the program. While sorting i need the array or the data of sorting. For some sorting algorithms i need to copy the array before sorting and for some i do not so, the memory usage differs betien different sorting algorithms. Nowadays, memory usage has not been the big issue as i have abundant data. But before, i used to have a various problem as i use to have very small memory. And algorithm which had less memory usage ire thought to be better than the others. Now, it’s not just the memory usage which sorting algorithm is better. So, i compare in 3 important criteria. On this paper, i will work on 5 different sorting algorithms which are Quicksort, bubble sort, heapsort, merge sort and radix sort(Sorting Algorithms).

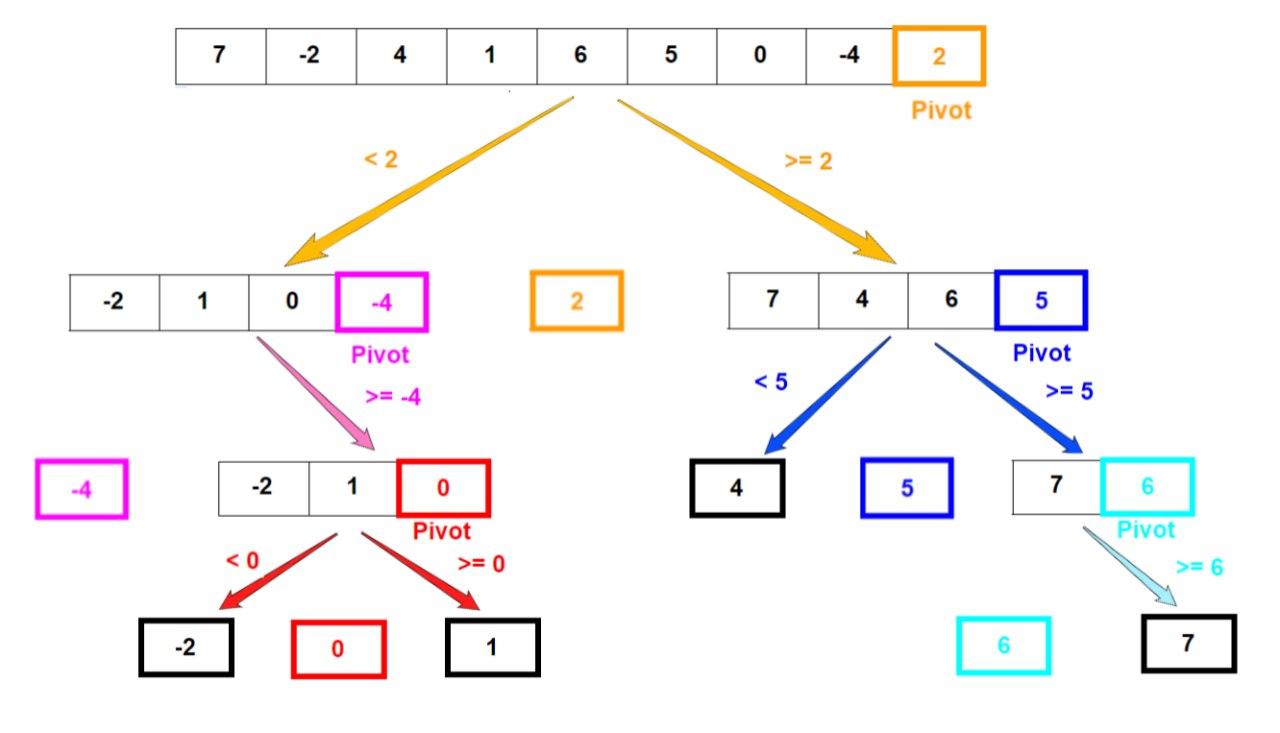
**Datatype to be used:**

The datatype i will be using in this research is integer. It is easier to see how the sorting works on integer as i can see all three criteria in the shorter period. In addition, i can create random integers easily with the Random function in java. Since there was a lot of benefit using the integer, Integer datatype was used to compare different sorting methods.

**Quicksort:**

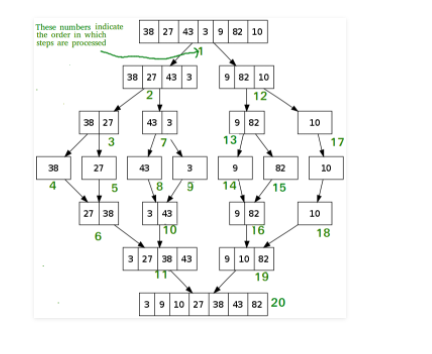
**Introduction:**

Quick sort uses a divide and conquer approach. This algorithm was first developed by Tony Hoare in 1959 and was first published in 1961. Using this algorithm, i create a pivot element in an array and then partition it with the other elements in the two sub arrays. These arrays are created by less than or greater than. Due to this reason quicksort is also mentioned as partition- exchange sort where i change the partition and exchange as per the size. Then recursively the sub arrays are sorted and then the big array is sorted. Here, the big array is divided into several smaller parts to get the sorted array at the end(Sedgwick).



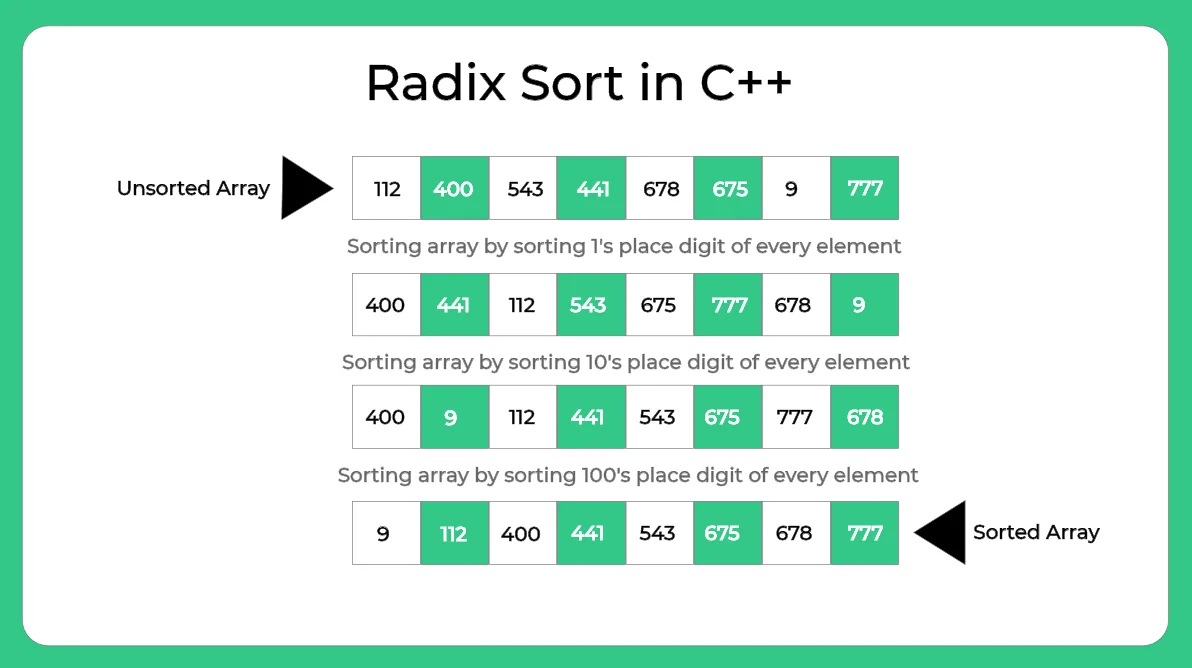
**Merge Sort:**

This sorting is another divide and conquer sorting algorithm. This was invented by John Van Neumann in 1945(Knuth). Hoiver, the report for this came in 1948 which was given by Goldstine and von Neumann. In this i divide the input array into two halves. Which again divides into two halves and then merges the two sorted haves. Then i merge all those divided arrays. That is why this sorting is called a merge sort. The picture below can help you understand the merge sort(Sedgwick).



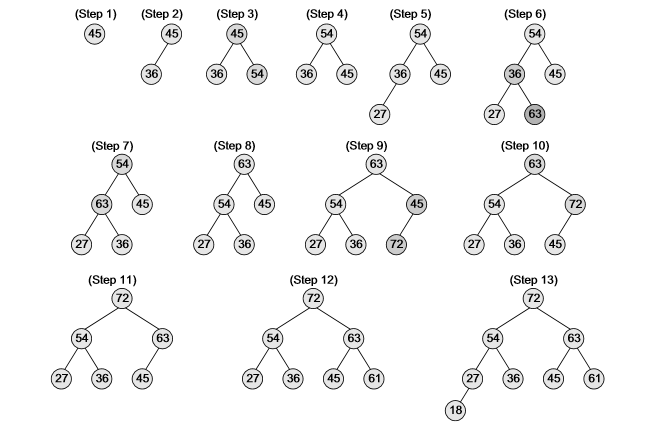
**Radix Sort:**

Radix sort is a non-comparative sorting algorithm. This avoids comparison by creating and distributing elements into buckets according to their Radix. Radix sort dates back as far as 1887. This was the work of Herman Hollerith. Now the radix sort is used in binary strings and integers. This was first used to linear scan to determine the required bucket sizes and offset beforehand. For numerous data, the bucketing process is repeated for each digit, while preserving the ordering of the prior step(Sedgwick).



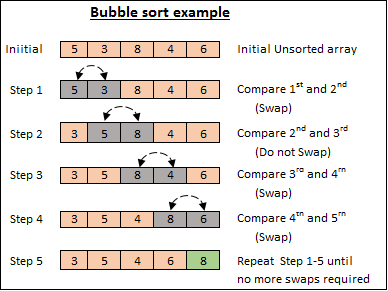
**Heap Sort:**

Heap sort is a comparison-based sorting algorithm. This is based on binary heap data structure. In this i first find the minimum element and place that element on the beginning. This process repeats for all the elements. Going to the history of heap sort, heapsort was invented by W.J. Williams in 1964. This also generated the birth of the heap and was treated as one of the famous data structures. In the same year R W. Floyd published an improved version which could sort an array(Fundamentals).



**Bubble Sort:**

Bubble sort, also known as a sinking sort. In this sort they repeatedly step through the list and swaps them if they are in wrong order. This keeps on continuing till the list is completely sorted. This was referred as sorting by exchange and then it was referred as Exchange Sorting and in 1962 Iverson named it the Bubble sort term. Hoiver, i do not have the clear history how this started but became one of the famous sorts on its time. The figure below has explained how the bubble sort works.



**Research with the code:**

**Comparison Table received from Research**

|  |  |  |  |
| --- | --- | --- | --- |
| Sorting Algorithm | Time complexity | Stability | Memory Usage |
| Quick Sort | nlogn | Not Stable | Logn |
| Merge sort | nlogn | Stable | n |
| Radix Sort | N\*k/d | Stable | N+r |
| Heap Sort | nlogn | Not Stable | 1 |
| Bubble Sort | n | Stable | 1 |

Time Complexity test:

While i ire calculating the time taken by all those five-sorting algorithms, i found that 3 of them which are heap sort, merge sort and quick sort ire nearly similar. The bubble sort took the longest of all. The time complexity of any sorting algorithms is base on the cases. Now from the book introduction to algorithms by Thomas H Cormen, Charles Lieserson, Ronald L Rivest, Clifford Stein i can get the following table for quicksort, merge sort, radix sort heapsort and bubble sort.

|  |  |  |  |
| --- | --- | --- | --- |
| Sorting Algorithm | Best case | Average case | Worst case |
| Quicksort | nlogn | nlogn | n^2 |
| Merge Sort | nlogn | nlogn | nlogn |
| Heap Sort | nlogn | nlogn | nlogn |
| Radix sort | D\*(n+b) | D\*(n+b) | D\*(n+b) |
| Bubble sort | n | n^2 | n^2 |

So before explaining the table lets know what best case, worst case and the average case is.

Best case: Best case is when the elements are already arranged in the order. For example, if the array is ordered already and is already sorted then then the sorting algorithm does not work a lot when ordering the algorithm. This is the fastest case for that algorithm. Instead of sorting if i go on searching an element the best case is when the element is on the first of the list.

Worst case: Worst case is exactly opposite of the best case. When the sorting algorithm must do a lot of work as the smaller elements are at the end of the list. This is when the algorithm will take a longer time. Instead of sorting if i go on searching an element the worst case is when the element is in last of the list.

Average case: Average case is exactly in betien two cases above where some parts are sorted, and some are not. Example, instead of sorting if i go on searching an element then average case is when the element is in the middle of list.

Now i look at the algorithms that i have used. Let us look at worst case for all five algorithms. Quicksort depends on the pivot for sorting. If the pivot is the largest or smallest element in the list, or when all elements are equal. When this happens then in every partition, then each recursive call process a lost of size one less than the previous list. This means that i are making *n-1* This means there is tree is a linear chain of n-1 nested calls. The ith call does O(n-i) for partitioning and when i sum it up, i get O(n^2). That is the reason quicksort takes O(n^2) time [Faron Moller]. In merge sort, it uses approximately 39 % feir comparisons than quicksort does in its average case, this makes merge sort worst case complexity same as that of quicksort’s best case [ Donald]. Heap sort on the other hand is non recursive code. As a result, it does not suffer to O(n^2) complexity as the quicksort. Due to this the time required for the heapsort remains constant for the best and worst case. On the other hand, the radix sort is a non-comparative algorithm so therefore has linear time complexity for all value. As a result, for large and large data this becomes faster than that of other sorting algorithms. Hoiver, the bubble sort takes the longest on the worst case which is in exponential. So, in looking at all algorithms and their time complexity. It looks like the merge sort and heapsort is fast even on the worst case. Hoiver, radix sort has the linear time complexity. So, i might consider these three algorithms for fast time complexity.

Stability: Quicksort is not stable as the partition or pivot element just separates as per the pivot as a result the quicksort becomes unstable. Hoiver, in merge sort the list is completely divided and sorted as a result there is complete stability as the first elements equal comes first in sorted list too. On the other side heapsort does not contain stability as it just inserts value on the heap. On bubble sort and radix sort is stable (Sahni).

Memory Usage: On memory usage, quicksort uses constant additional space before making additional recursive call. This needs to store constant amount of information for each nested recursive call. In best case the quicksort calls for O(logn) recursive call hoiver on the worst case it requires O(n) recursive calls requiring O(n) recursive space (Sedgewick). In merge sort the space complexity becomes same as a that of worst case of quicksort, as i must divide till the smallest size which is till O(n). In heap sort i do not have recursive method so, there is constant memory usage. Similar goes with the radix sort and bubble sort (Donald).

**Code used:**

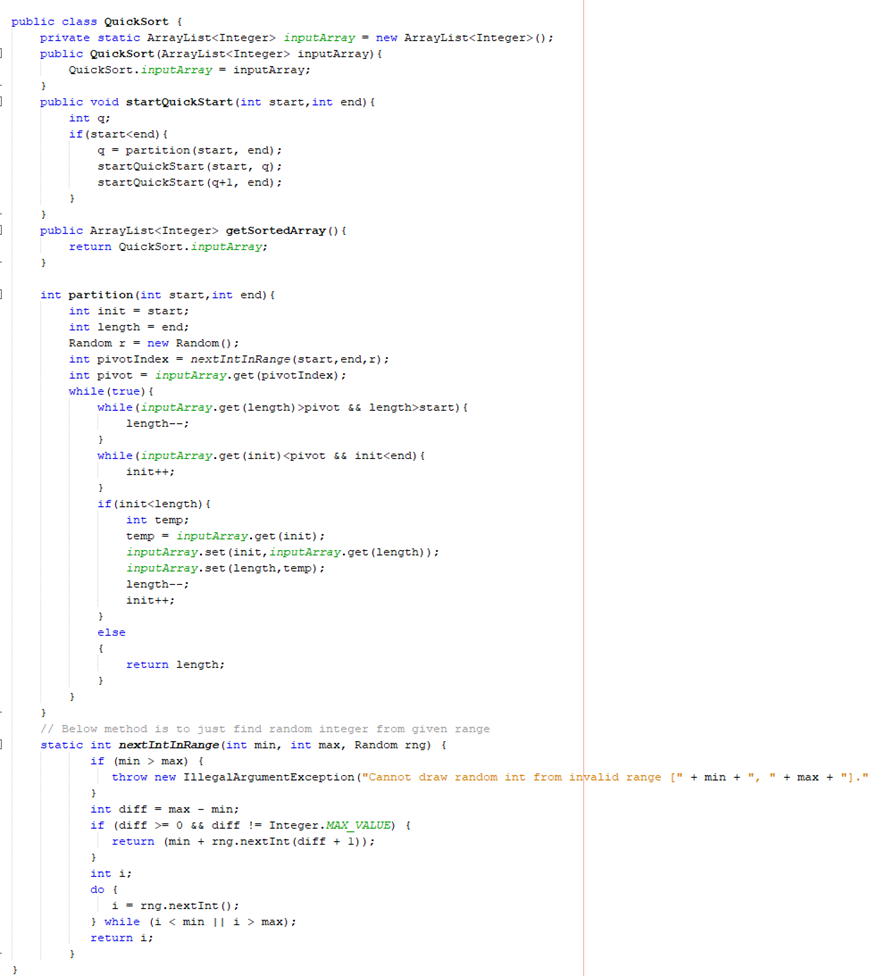


Figure 2QuickSort Algorithm

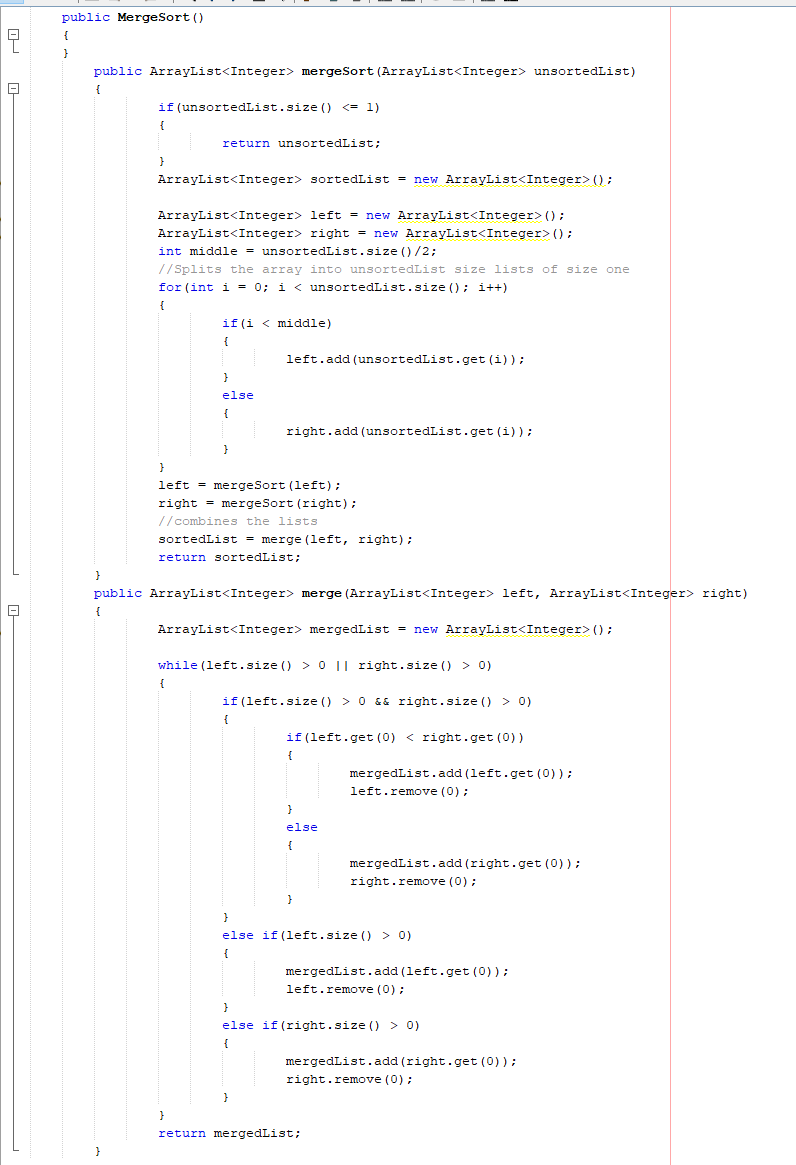


Figure 2 3Merge Sort code

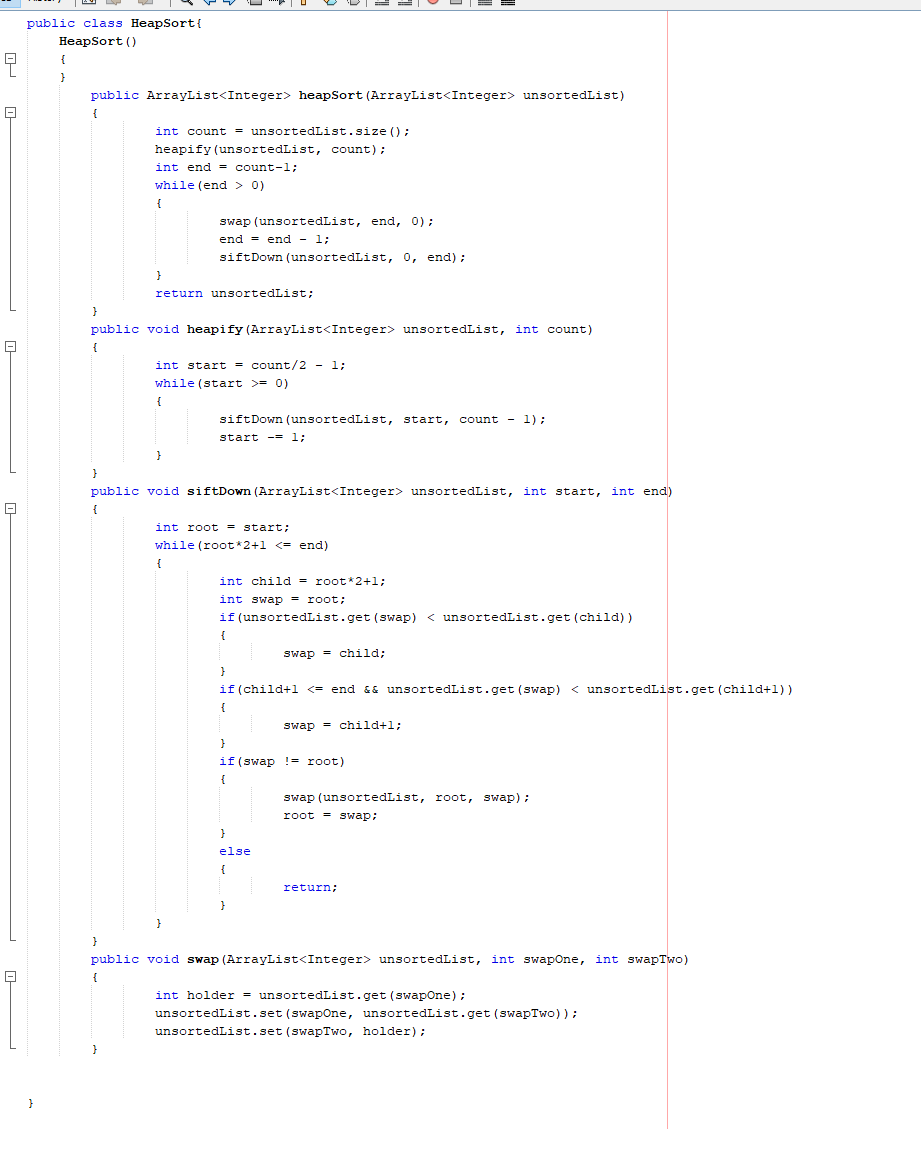


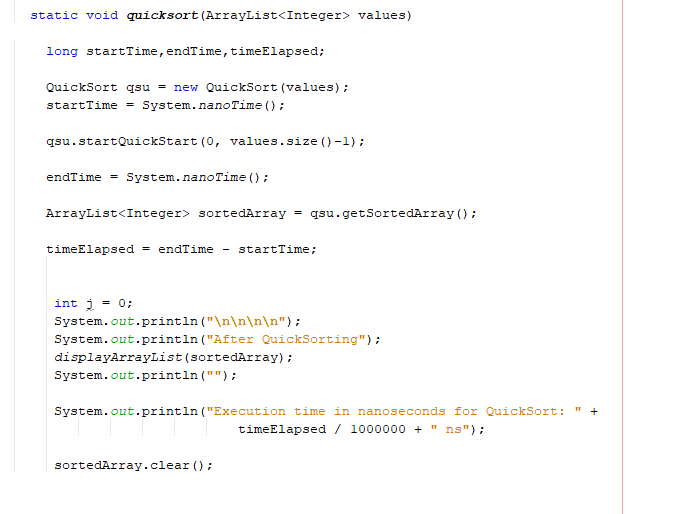
Figure 3 Merge Sort

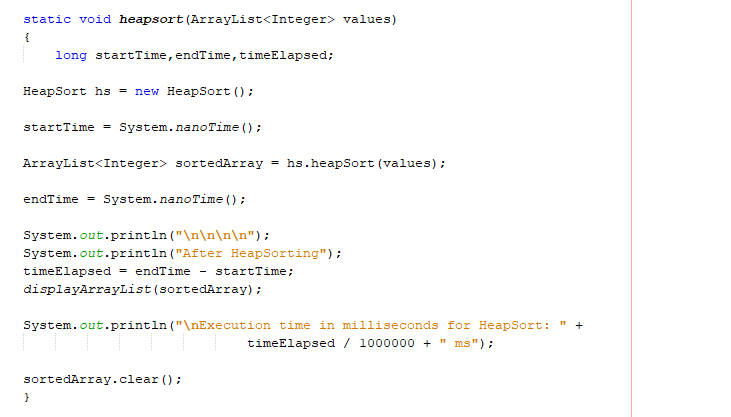


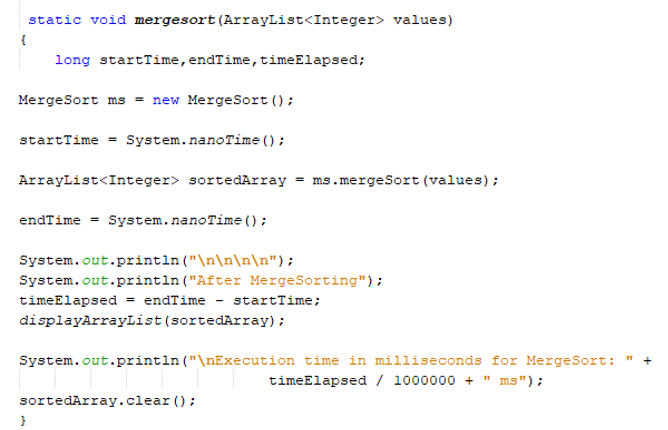
Figure 4 Radix Sort

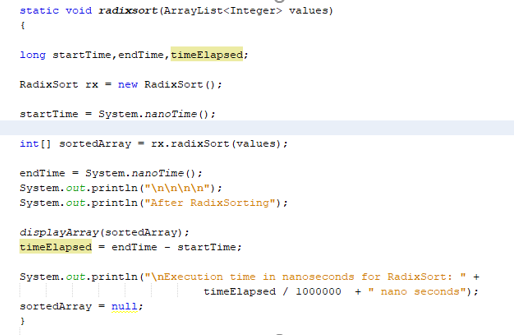


Figure 5 Bubble Sort

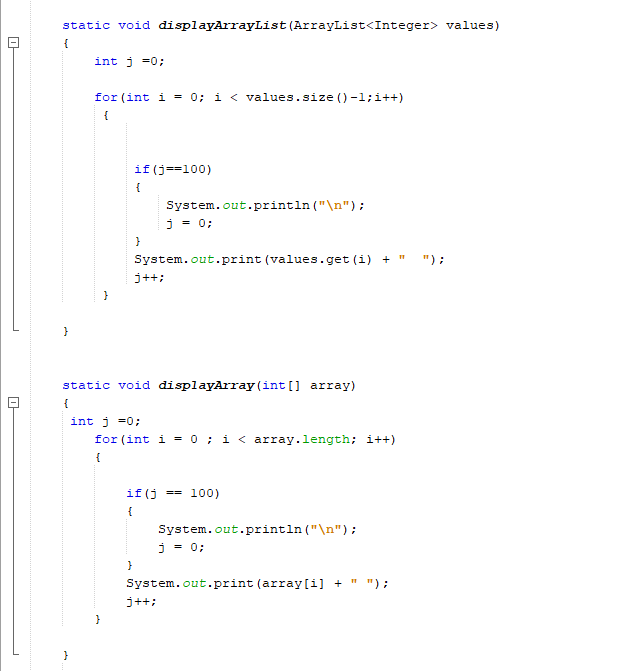
**Code of functions used**

****







****

**Code of Main:**

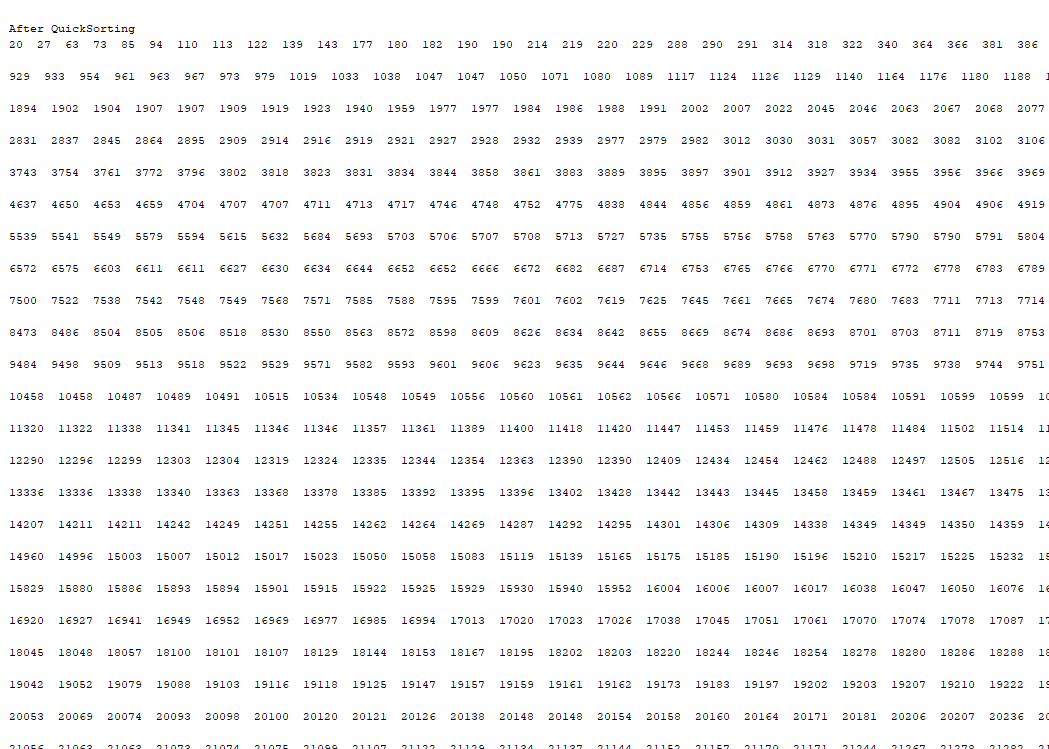
****

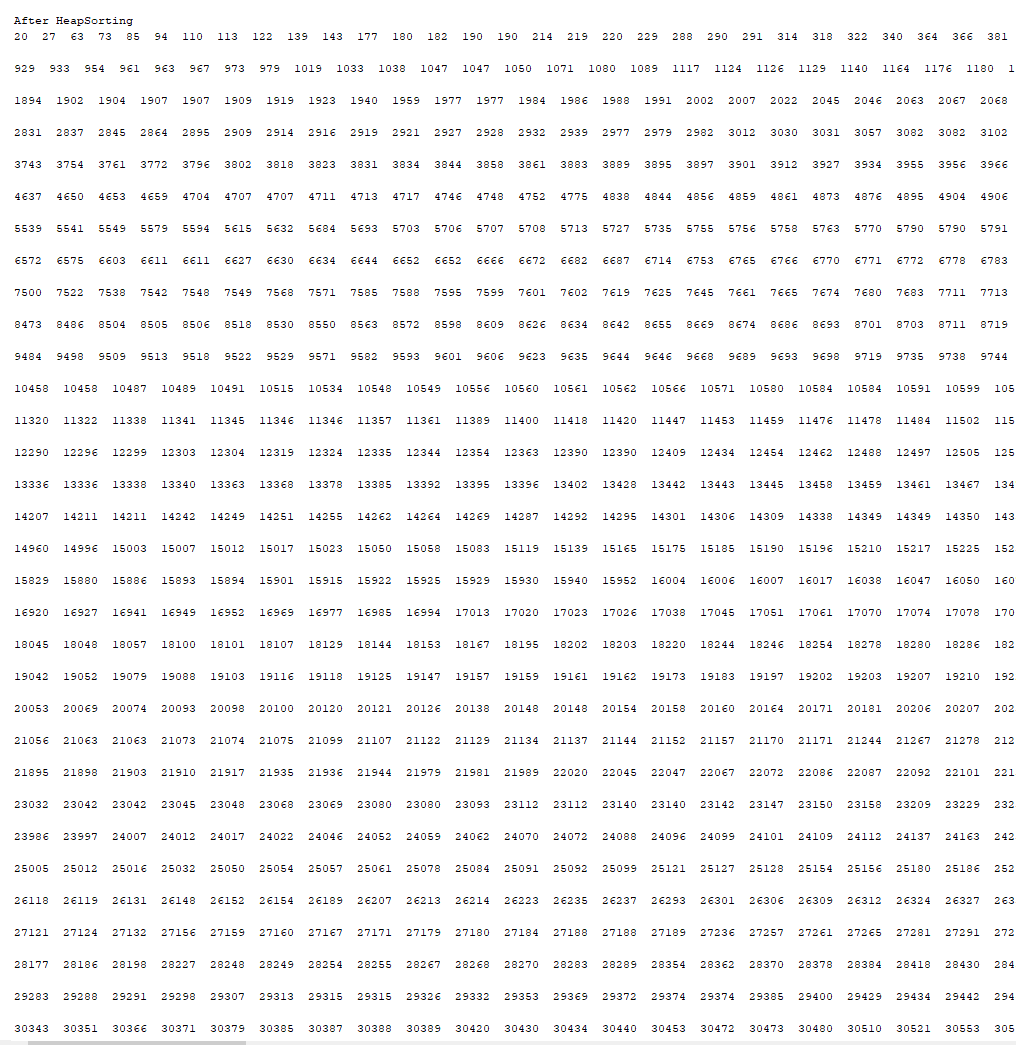
****

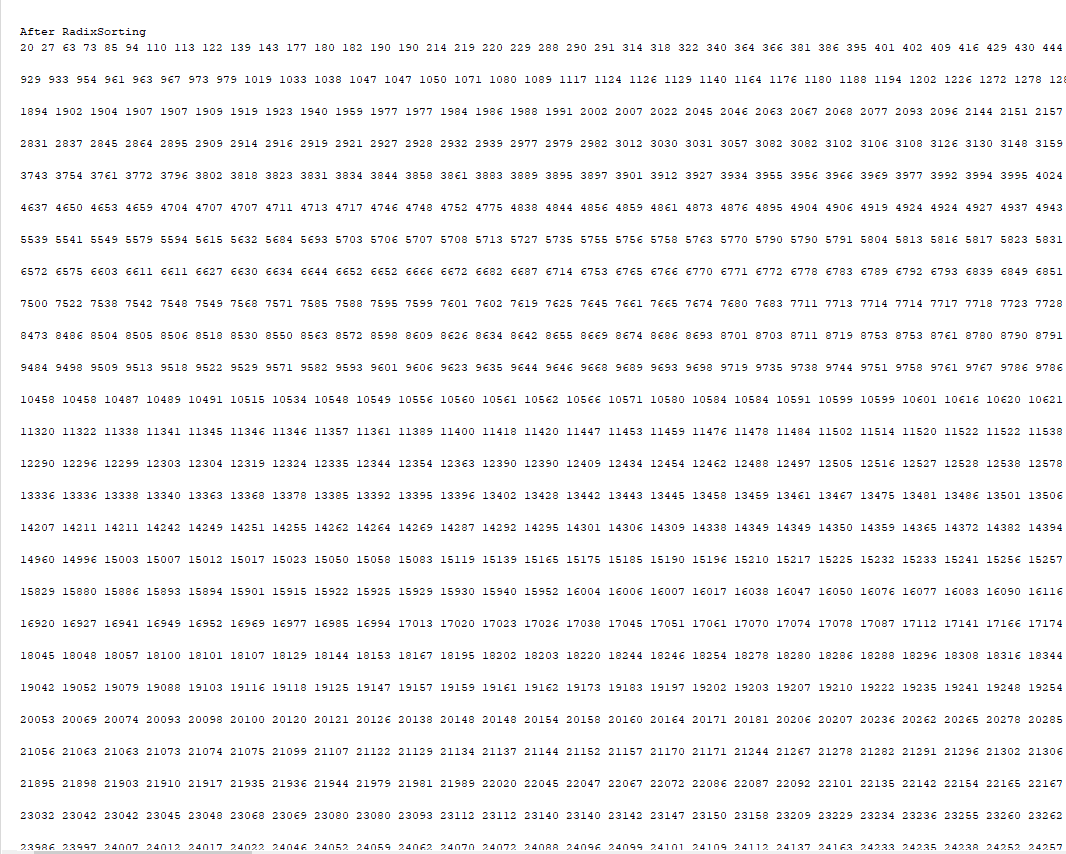
**Output:**

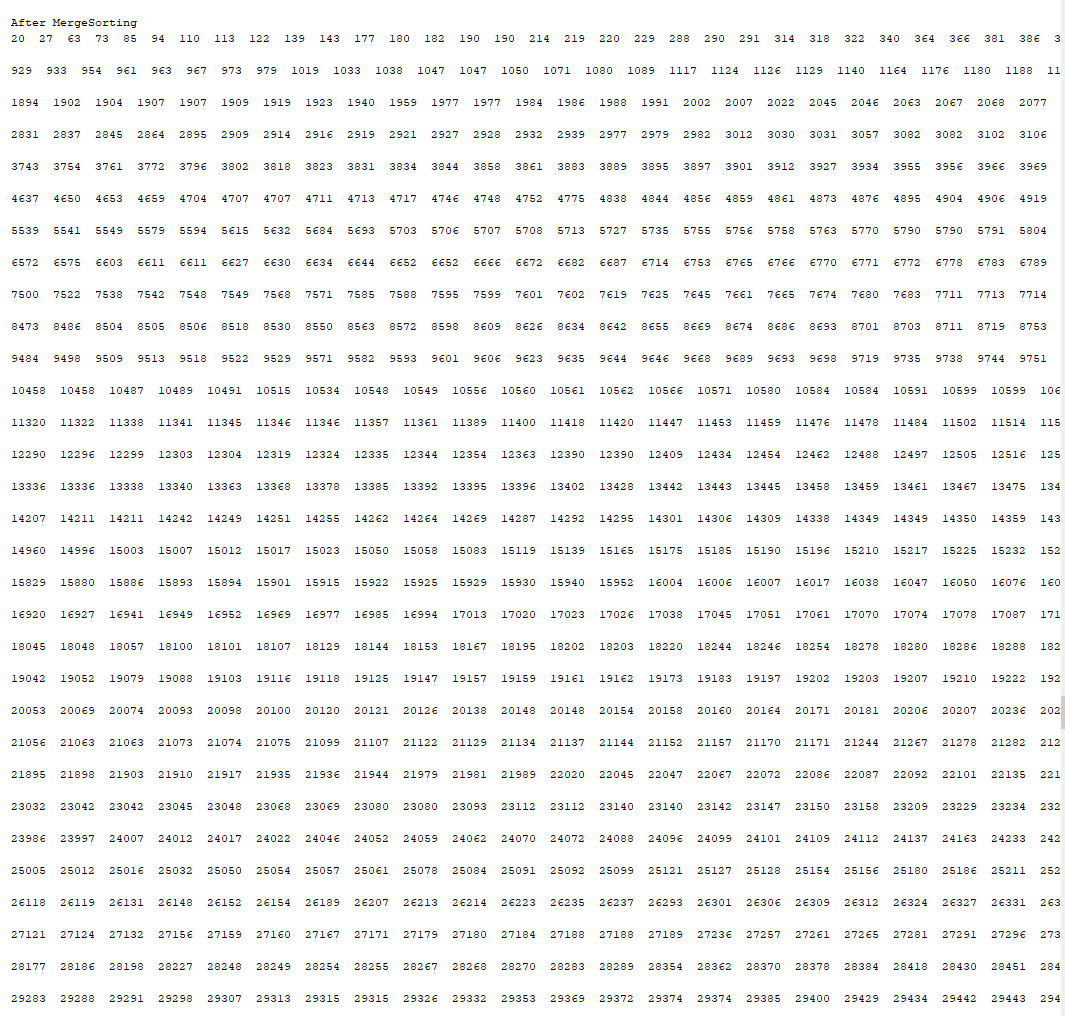
****

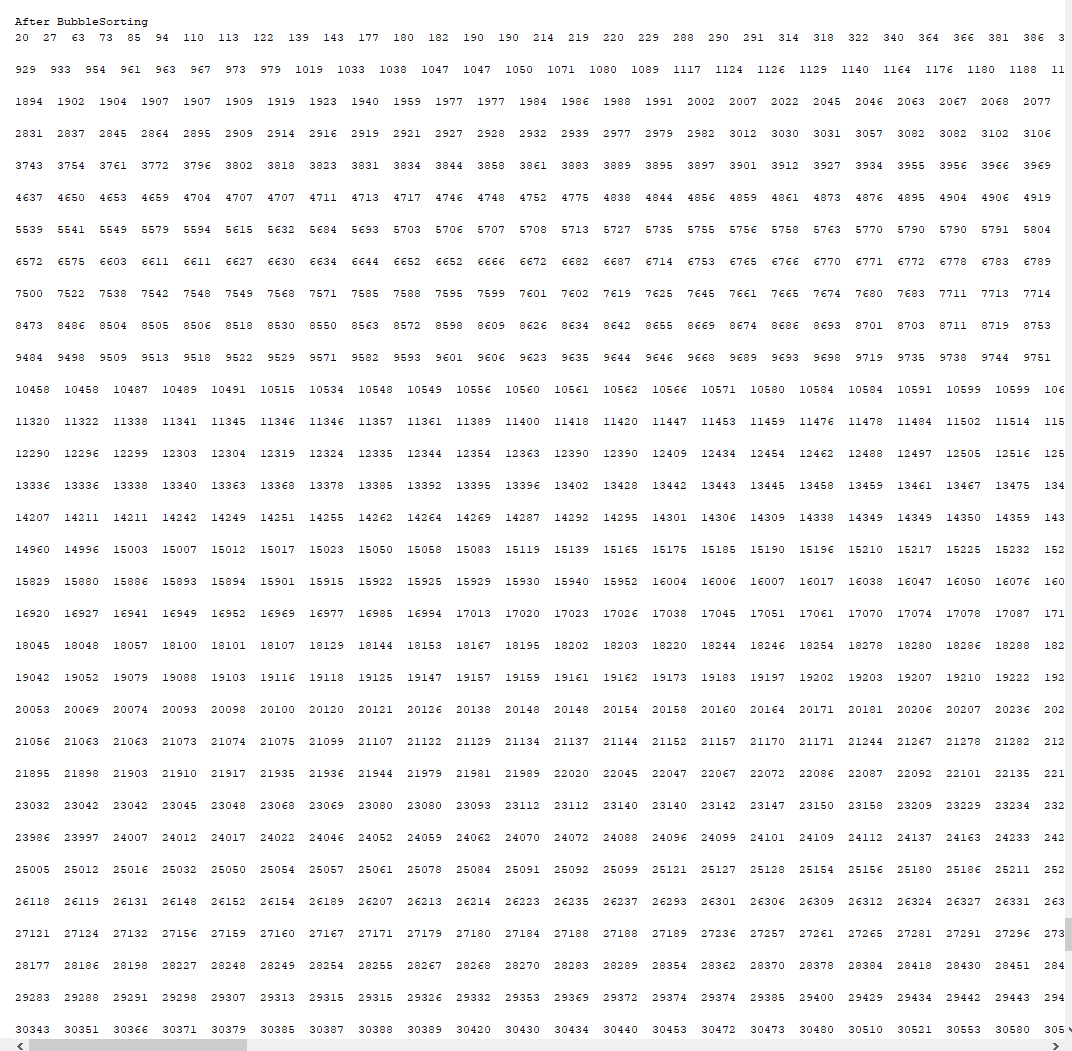
Figure 6 Before Sorting

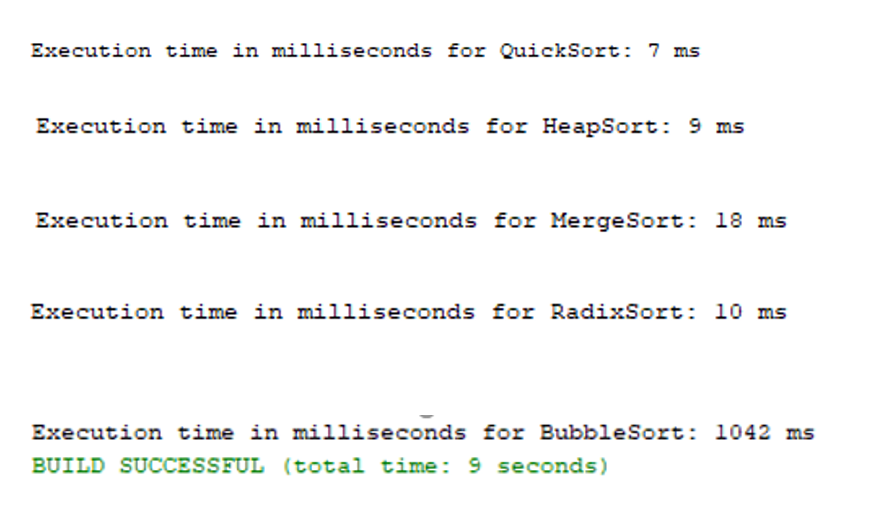












**Discussion:**

So, after i looked at the algorithms and after i tested it, every algorithm had it own good and its bad. Quicksort was fast hoiver not stable and was slow on the worst-case scenario. On the other hand, merge sort was fast, but the memory usage was higher than other sorting algorithms. Heapsort lacked the stability even being fast and efficient. Bubble sort on the other hand, even having stability and constant memory, it is a very slow process and takes a longer time. Radix sort is stable, is faster (not as quicksort) and is stable. Hoiver, radix sort depended on letters or digits as a result was less flexible than the other sorting algorithms.

**Conclusion:**

This paper discusses five sorting algorithms and how they work. Every sorting algorithm had their advantages and disadvantages. Some sort ire faster but lacked stability. Some ire stable but not fast enough. Some ire both fast and stable but did not contain the good memory usage. And which had all was not flexible.

**Work Cited from**

* Hsieh, Paul (2004). ["Sorting revisited"](http://www.azillionmonkeys.com/qed/sort.html). azillionmonkeys.com.
* [Faron Moller](https://en.wikipedia.org/wiki/Faron_Moller). [Analysis of Quicksort](http://www.cs.swan.ac.uk/~csfm/Courses/CS_332/quicksort.pdf). CS 332: Designing Algorithms. Department of Computer Science, [Swansea University](https://en.wikipedia.org/wiki/Swansea_University).
* [*Knuth, Donald*](https://en.wikipedia.org/wiki/Donald_Knuth) (1998). "Section 5.2.4: Sorting by Merging". Sorting and Searching. [*The Art of Computer Programming*](https://en.wikipedia.org/wiki/The_Art_of_Computer_Programming). **3** (2nd ed.). Addison-Isley. pp. 158–168. [*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier)) [*0-201-89685-0*](https://en.wikipedia.org/wiki/Special:BookSources/0-201-89685-0).
* [*Sedgewick, R.*](https://en.wikipedia.org/wiki/Robert_Sedgewick_(computer_scientist)) (1978). "Implementing Quicksort programs". [*Comm. ACM*](https://en.wikipedia.org/wiki/Communications_of_the_ACM). **21** (10): 847–857. [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier)):[*10.1145/359619.359631*](https://doi.org/10.1145%2F359619.359631). [*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier)) [*10020756*](https://api.semanticscholar.org/CorpusID:10020756).
* [Ellis Horowitz](https://en.wikipedia.org/wiki/Ellis_Horowitz) and [Sartaj Sahni](https://en.wikipedia.org/wiki/Sartaj_Sahni), “*Fundamentals of Data Structures”*, H. Freeman & Co., [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [0-7167-8042-9](https://en.wikipedia.org/wiki/Special:BookSources/0-7167-8042-9).
* Lavore, Robert. Arrays, “Big O Notation and Simple Sorting. Data Structures and Algorithms in Java (2nd Edition)” . Sams, 978-0-672-32453-6
* Savitch, Walter and Carrano, Frank. “Arrays. Java: Introduction to Problem Solving and Programming (5th Edition)”. Prentice Hall, 9780136072256. Sedgewick, Robert.
* Fundamentals, Data Structures, Sorting, Searching, and Graph Algorithms. Bundle of Algorithms in Java, Third Edition. Addison-Isley
* Sorting Algorithms. Online. http://people.cs.ubc.ca/ harrison/Java/sorting-demo.html
* Wikipedia, the free encyclopedia. Online. Sorting Algorithms.