Selection Sort

```java
import java.util.Random;
public class main {
    public static void main(String[] args) {
        int n = 1000;
        int[] randomArray = new int[n];
        Random rand = new Random();
        for (int i = 0; i < n; i++) {
            randomArray[i] = rand.nextInt(1000);
        }
        long start = System.currentTimeMillis();
        sort(randomArray);
        long finish = System.currentTimeMillis();
        long timeElapsed = finish-start;
        System.out.println(timeElapsed);
    }
    static void sort(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int min_idx=i;
            for(int j = i+1;j<n;j++)

                if(arr[j]<arr[min_idx])
                    min_idx=j;

            int temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i]=temp;
        }
    }
}
```

Used Java's Random Library in order to fill our array.

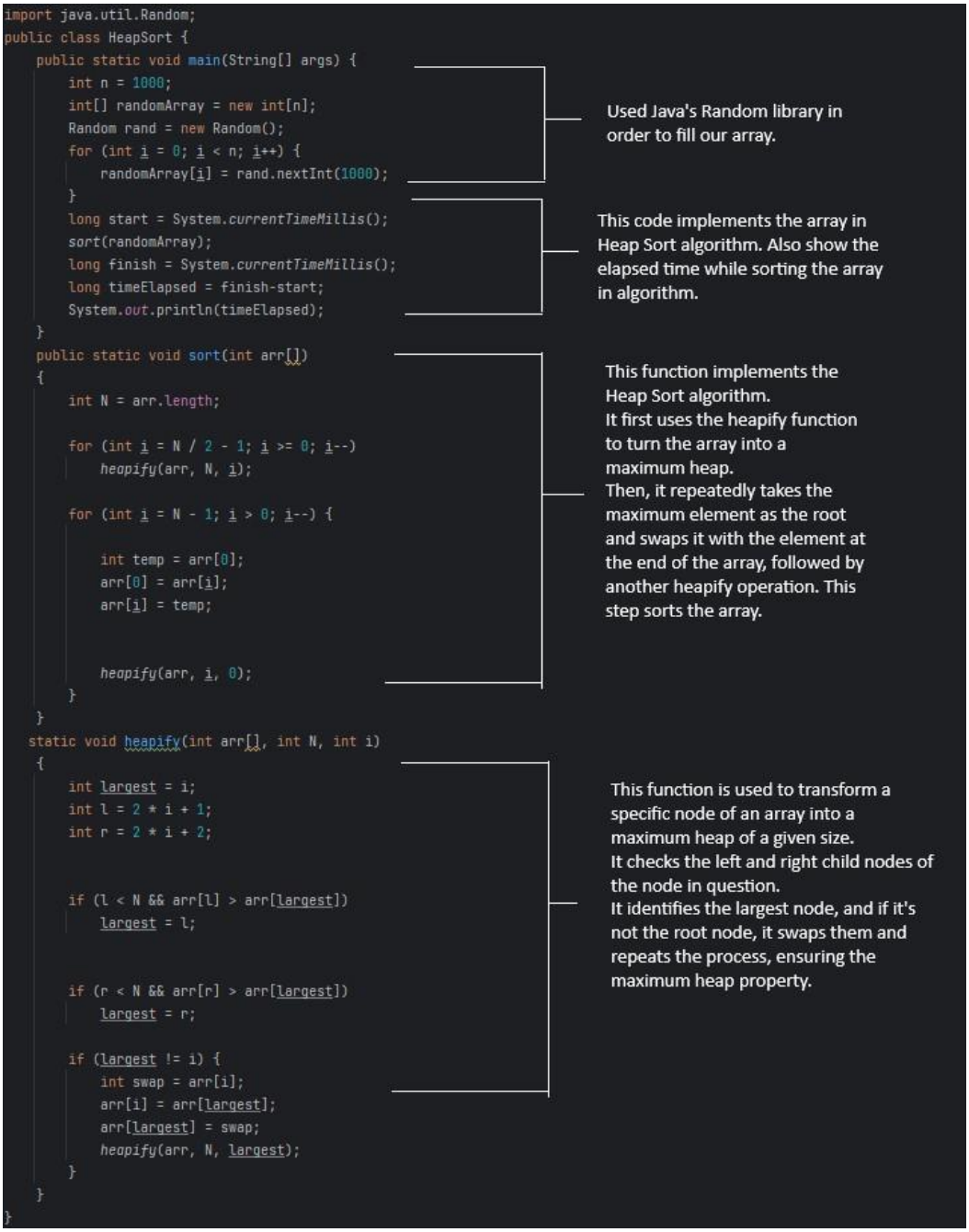
This code implements the array in Selection Sort algorithm. Also shows the elapsed time while sorting the array in algorithm.

Selection Sort method

Table of showing the array sizes and the sorting time in the Selection Sort algorithm:

Array Size (integers)❼	1000	10000	100000	1000000	3133456
Selection Sort Elapsed Time (millisecond)❼	3	47	4471	336587	723675

Heap Sort




```java
import java.util.Random;
public class HeapSort {
    public static void main(String[] args) {
        int n = 1000;
        int[] randomArray = new int[n];
        Random rand = new Random();
        for (int i = 0; i < n; i++) {
            randomArray[i] = rand.nextInt(1000);
        }
        long start = System.currentTimeMillis();
        sort(randomArray);
        long finish = System.currentTimeMillis();
        long timeElapsed = finish-start;
        System.out.println(timeElapsed);
    }
    public static void sort(int arr[])
    {
        int N = arr.length;

        for (int i = N / 2 - 1; i >= 0; i--)
            heapify(arr, N, i);

        for (int i = N - 1; i > 0; i--) {

            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;


            heapify(arr, i, 0);
        }
    }
   static void heapify(int arr[], int N, int i)
    {
        int largest = i;
        int l = 2 * i + 1;
        int r = 2 * i + 2;


        if (l < N && arr[l] > arr[largest])
            largest = l;


        if (r < N && arr[r] > arr[largest])
            largest = r;

        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
            heapify(arr, N, largest);
        }
    }
}
```

Table of showing the array sizes and the sorting time in the Heap Sort algorithm:

Array Size (integers)❼	1000	10000	100000	1000000	3133456
Heap Sort Elapsed Time (millisecond)❼	1	2	15	180	632

Merge Sort

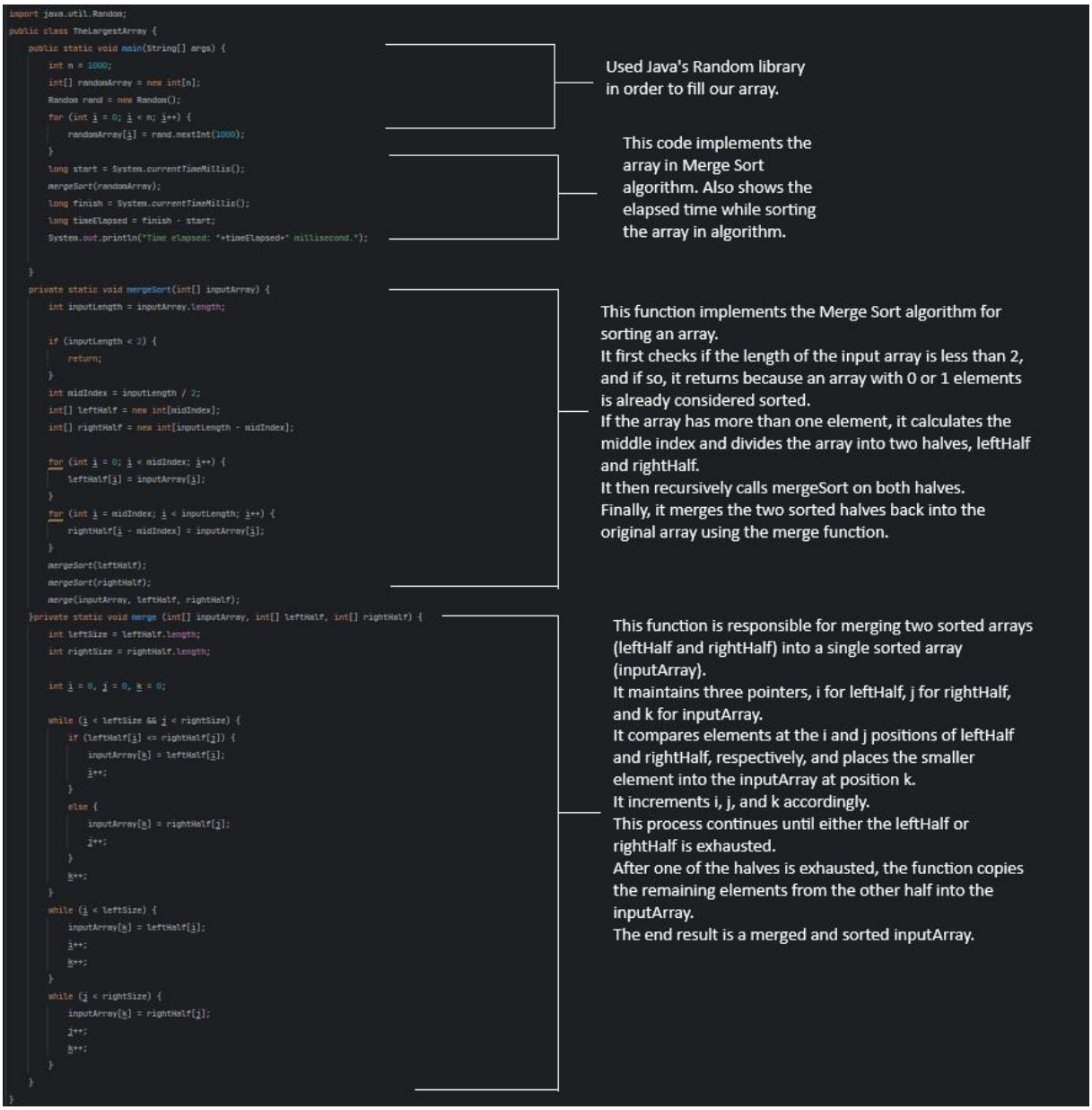


Table of showing the array sizes and the sorting time in the Merge Sort algorithm:

Array Size(integers) -->	**1000**	**10000**	**100000**	**1000000**	**3133456**
Merge Sort Elapsed Time (millisecond)❼	1	2	45	181	520

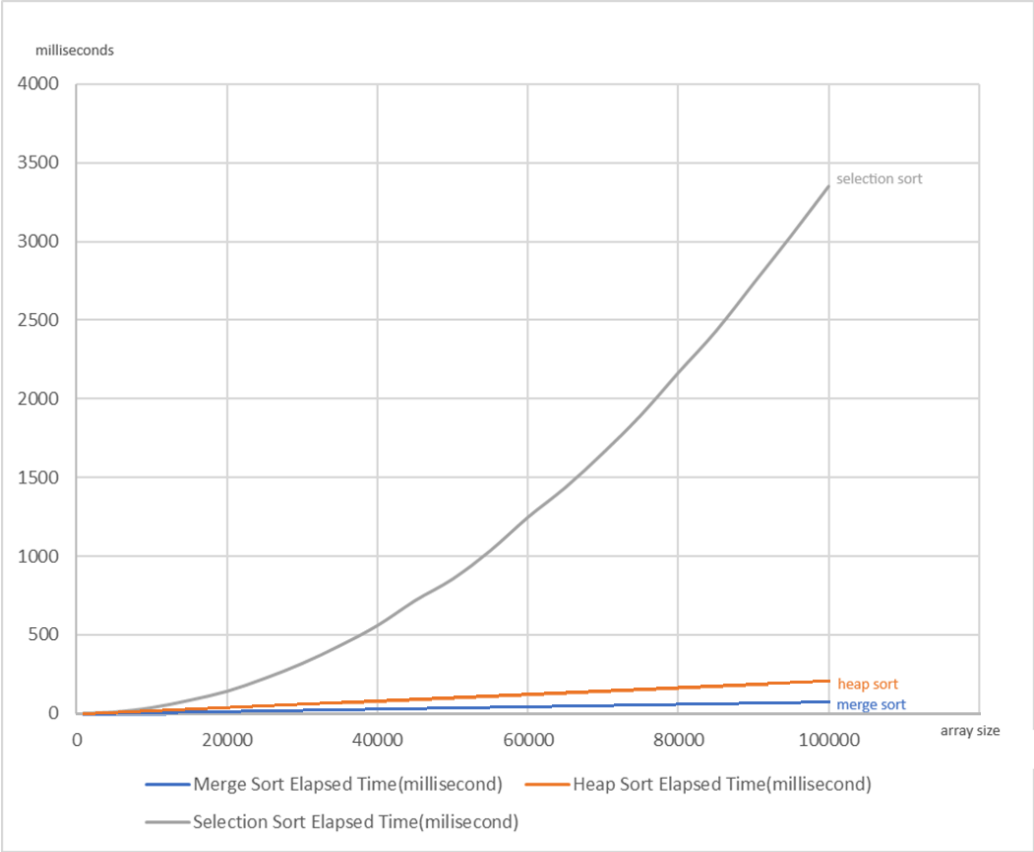
Comparison of the three sorting algorithms

Array size (integers)	1000	10000	100000	1000000	3133456
Selection Sort Elapsed Time (millisecond)	3	47	4471	336587	723675
Heap Sort(millisecond)	1	2	15	180	632
Merge Sort (millisecond)	1	2	45	181	520

In order to make a comprehensive comparison and make the graph it is possible to narrow the array sizes and make a better inference.

Array Size (integers)	1000	5000	10000	15000	20000	25000	30000	35000	40000	45000	50000	55000	60000	65000	70000	75000	80000	85000	90000	95000	100000
Merge Sort Elapsed Time(millisecond)	1	1	2	3	3	4	4	5	6	6	6	7	8	8	9	9	10	10	11	11	11
Heap Sort Elapsed Time(millisecond)	1	1	2	3	5	5	6	9	9	9	9	9	10	12	13	13	14	16	17	17	18
Selection Sort Elapsed Time(milisecond)	2	10	38	85	142	224	319	432	560	718	855	1035	1247	1437	1656	1894	2165	2429	2732	3035	3349

Graph of the table is given below:



Comparison and Conclusion:

As it easily can be seen on the graph, Selection Sort takes too much time compared with Merge and Heap Sort algorithms while sorting large sized arrays. Also selection sort is inferior in terms of performance compared to the other two algorithms. On the other hand, Heap and Merge Sort algorithms has a small diffrences between eachother. Both algorithms have O(n log n) complexities, but the difference occurs since Merge Sort uses extra memory because it creates new subarrays and merges them during the sorting process. However, Heap sort would be more efficient in terms of extra memory usage, because it only converts the data into heap structure.

Best Performing ❼Merge Sort

Worst Performing ❼ Selection Sort

References:

GeeksforGeeks. (n.d). Selection Sort. https://www.geeksforgeeks.org/selection-sort/

GeeksforGeeks. (n.d). Heap Sort. https://www.geeksforgeeks.org/heap-sort/

GeeksforGeeks. (n.d). Merge Sort. https://www.geeksforgeeks.org/merge-sort/

Ted University CMPE223 Sorting Algorithms Lecture Notes (n.d)