CS 2001 W10 Practical
180000560
20th November 2019
Tutor: Tom Kelsey

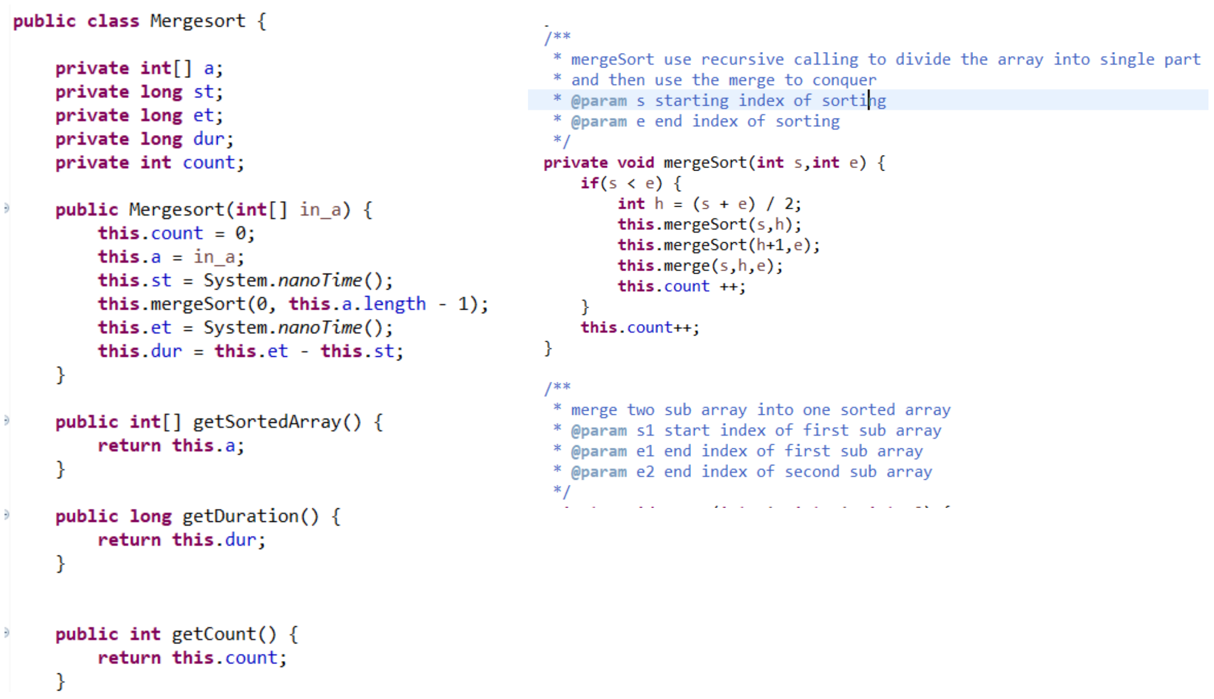
Overview

Implementing the code of merge sort and selection sort respectively, then measuring the amount of time for sorting array with different size by two methods. Finally, analysing how the time complexities of two algorithms varies in random sequences with different sizes.

Design and Implementation

Merge sort and Selection sort is implemented for testing purpose.

Merge sort

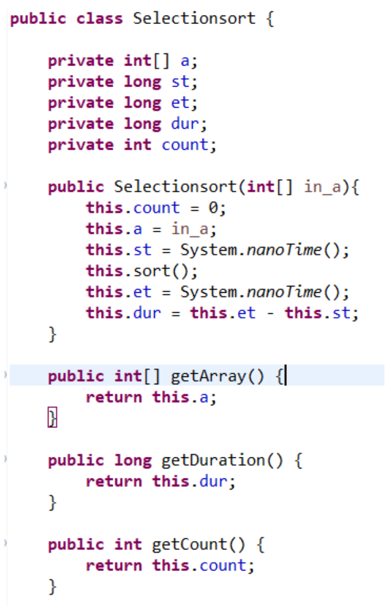


```java
public class Mergesort {

    private int[] a;
    private long st;
    private long et;
    private long dur;
    private int count;

    public Mergesort(int[] in_a) {
        this.count = 0;
        this.a = in_a;
        this.st = System.nanoTime();
        this.mergeSort(0, this.a.length - 1);
        this.et = System.nanoTime();
        this.dur = this.et - this.st;
    }

    public int[] getSortedArray() {
        return this.a;
    }

    public long getDuration() {
        return this.dur;
    }


    public int getCount() {
        return this.count;
    }
```

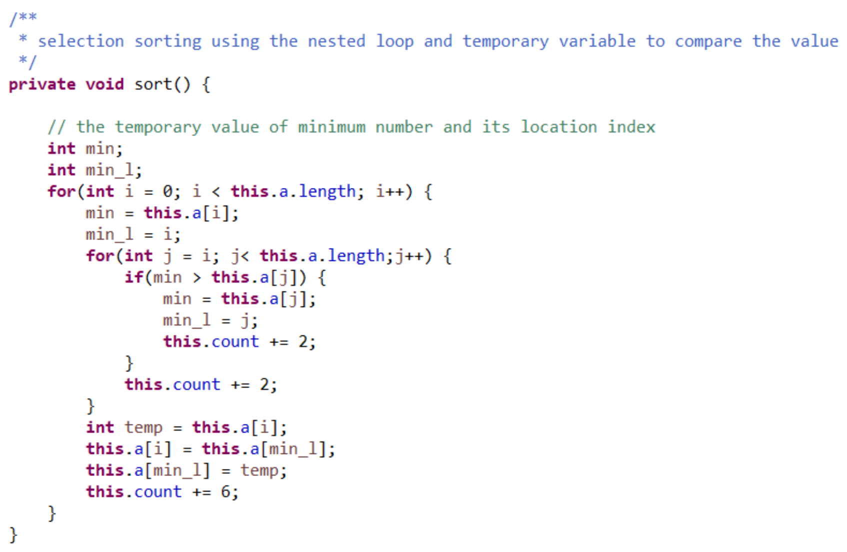
```java
/**
 * mergeSort use recursive calling to divide the array into single part
 * and then use the merge to conquer
 * @param s starting index of sorting
 * @param e end index of sorting
 */
private void mergeSort(int s,int e) {
    if(s < e) {
        int h = (s + e) / 2;
        this.mergeSort(s,h);
        this.mergeSort(h+1,e);
        this.merge(s,h,e);
        this.count ++;
    }
    this.count++;
}

/**
 * merge two sub array into one sorted array
 * @param s1 start index of first sub array
 * @param e1 end index of first sub array
 * @param e2 end index of second sub array
 */
```

```java
    private void merge(int s1, int e1, int e2) {
        int a1_len = e1 - s1 + 1, a2_len = e2 - e1;

        // declare sentinel value for possible problems due to different size of array
        int sen = 1;
        if(this.a[e1] > this.a[e2]) {
            sen += this.a[e1];
        }else {
            sen += this.a[e2];
        }
        int[] a1 = new int[a1_len + 1];
        int[] a2 = new int[a2_len + 1];
        count += 7;
        // create temporary array for sorting and then merge them together
        for(int i = 0; i < a1.length - 1; i ++) {
            a1[i] = a[s1 + i];
            count += 3;
        }
        for(int i = 0; i < a2.length - 1; i ++) {
            a2[i] = a[e1 + 1 + i];
            count += 3;
        }
        // last element of array is sentinel value which is larger than any other values in array
        // sentinel value will not enter the array due to the loop time is actually the sum of two array
        a1[a1.length - 1] = sen;
        a2[a2.length - 1] = sen;
        int t1 = 0, t2 = 0;
        count += 4;
        for(int i = s1; i <= e2 ; i++) {
            if(a1[t1] < a2[t2]) {
                this.a[i] = a1[t1];
                t1++;
            }else {
                this.a[i] = a2[t2];
                t2++;
            }
            count += 5;
        }
```

Selection Sort



```java
public class Selectionsort {

    private int[] a;
    private long st;
    private long et;
    private long dur;
    private int count;

    public Selectionsort(int[] in_a){
        this.count = 0;
        this.a = in_a;
        this.st = System.nanoTime();
        this.sort();
        this.et = System.nanoTime();
        this.dur = this.et - this.st;
    }

    public int[] getArray() {
        return this.a;
    }

    public long getDuration() {
        return this.dur;
    }

    public int getCount() {
        return this.count;
    }
```



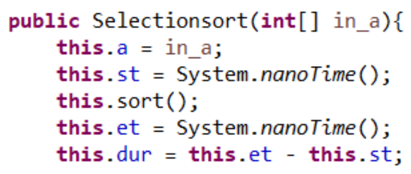
```java
    /**
     * selection sorting using the nested loop and temporary variable to compare the value
     */
    private void sort() {

        // the temporary value of minimum number and its location index
        int min;
        int min_l;
        for(int i = 0; i < this.a.length; i++) {
            min = this.a[i];
            min_l = i;
            for(int j = i; j< this.a.length;j++) {
                if(min > this.a[j]) {
                    min = this.a[j];
                    min_l = j;
                    this.count += 2;
                }
                this.count += 2;
            }
            int temp = this.a[i];
            this.a[i] = this.a[min_l];
            this.a[min_l] = temp;
            this.count += 6;
        }
    }
```

To measure the time complexity of algorithms, two ways is considered simultaneously:

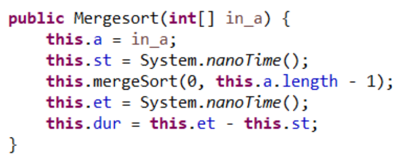
- Measuring running time of algorithms
- Number of executions take places during sorting

Nano second timer is considered for testing the amount of time.

Variables is declared to count the number of executions over whole sorting processes.

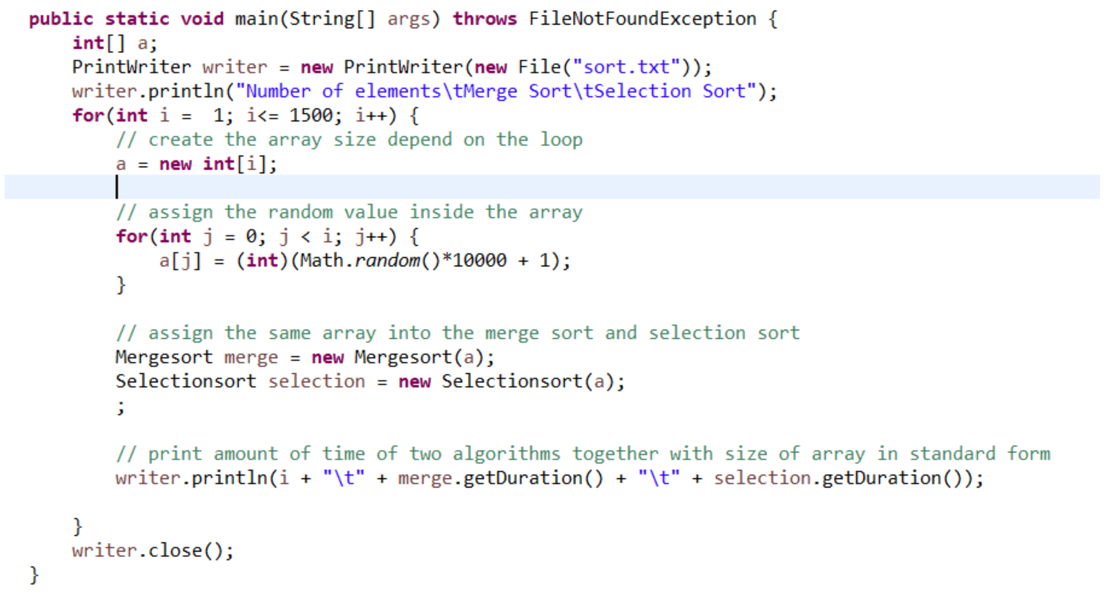


```
public Selectionsort(int[] in_a){
    this.a = in_a;
    this.st = System.nanoTime();
    this.sort();
    this.et = System.nanoTime();
    this.dur = this.et - this.st;
```



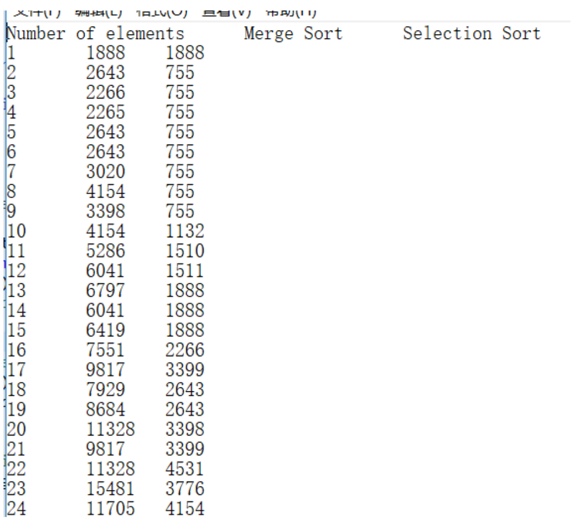
```
public Mergesort(int[] in_a) {
    this.a = in_a;
    this.st = System.nanoTime();
    this.mergeSort(0, this.a.length - 1);
    this.et = System.nanoTime();
    this.dur = this.et - this.st;
}
```

The function to test the two algorithms on random sequences with various sizes is automatic.



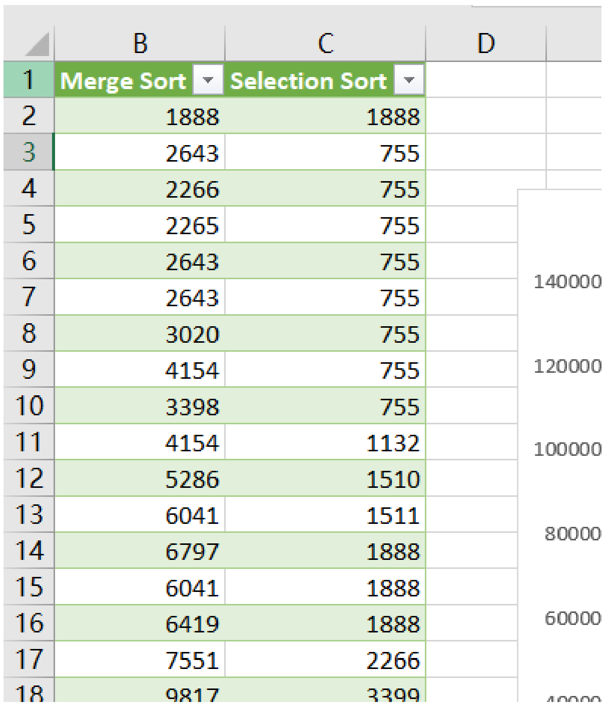
```
public static void main(String[] args) throws FileNotFoundException {
    int[] a;
    PrintWriter writer = new PrintWriter(new File("sort.txt"));
    writer.println("Number of elements\tMerge Sort\tSelection Sort");
    for(int i =  1; i<= 1500; i++) {
        // create the array size depend on the loop
        a = new int[i];

        // assign the random value inside the array
        for(int j = 0; j < i; j++) {
            a[j] = (int)(Math.random()*10000 + 1);
        }

        // assign the same array into the merge sort and selection sort
        Mergesort merge = new Mergesort(a);
        Selectionsort selection = new Selectionsort(a);
        ;

        // print amount of time of two algorithms together with size of array in standard form
        writer.println(i + "\t" + merge.getDuration() + "\t" + selection.getDuration());

    }
    writer.close();
}
```

Print writer is used to write time period into text file with formatted output. In the for loop, array's size will be set up depend on increasing looping time. Then random number will be assigned into the array. After generating a random array, both two sorting algorithms will sort the same array and their amount of time in sorting will be written into a file together with the size of the array they sort. Then next loop will begin with another random sequences with larger size.



```
Number of elements       Merge Sort       Selection Sort
1         1888      1888
2         2643      755
3         2266      755
4         2265      755
5         2643      755
6         2643      755
7         3020      755
8         4154      755
9         3398      755
10        4154      1132
11        5286      1510
12        6041      1511
13        6797      1888
14        6041      1888
15        6419      1888
16        7551      2266
17        9817      3399
18        7929      2643
19        8684      2643
20        11328     3398
21        9817      3399
22        11328     4531
23        15481     3776
24        11705     4154
```

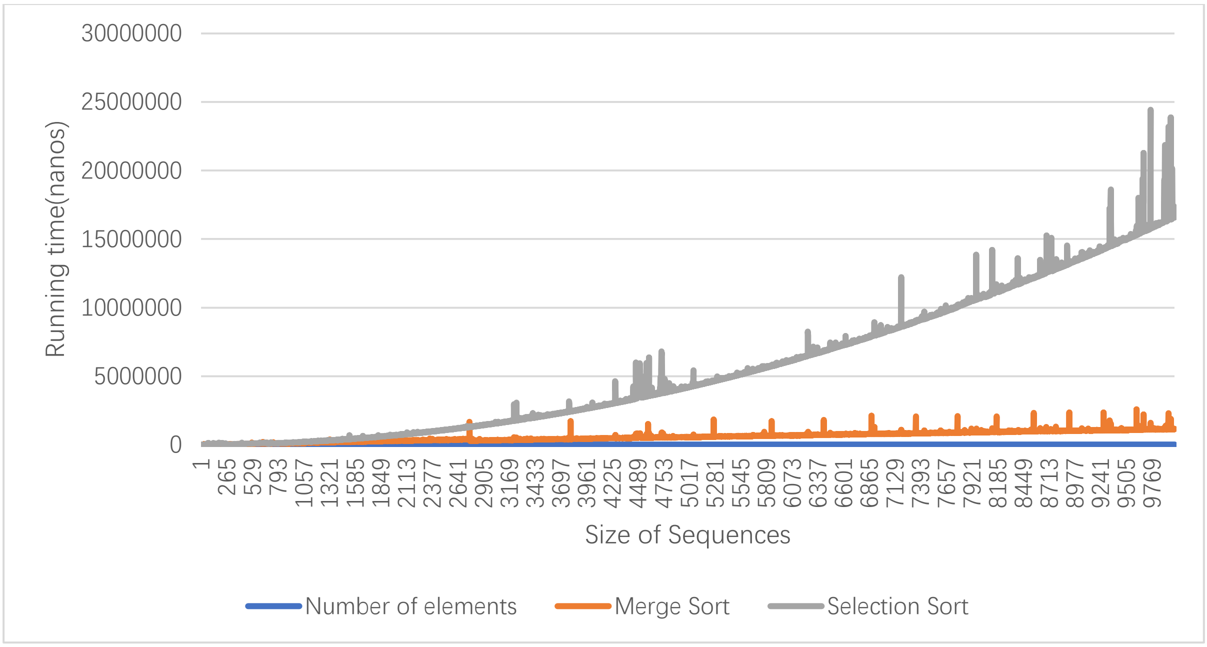
Excel application is used to collect the data and plot them into visualised graph for analysing.



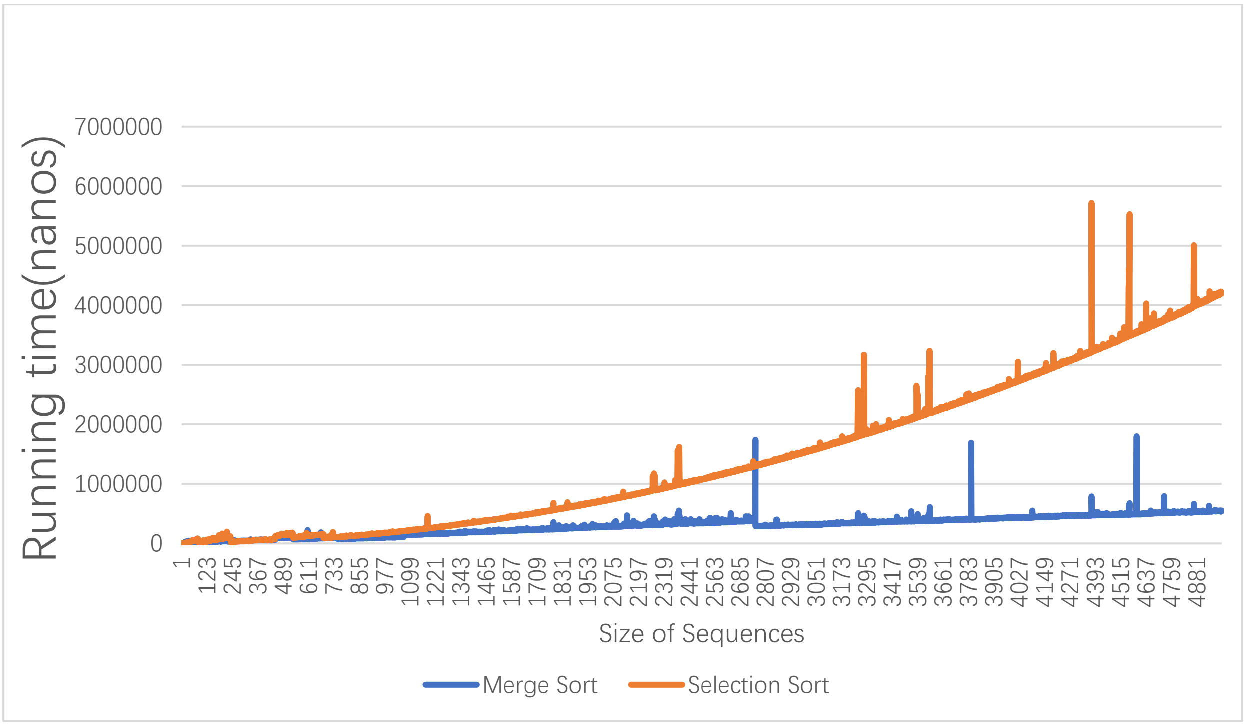
	B	C	D
1	Merge Sort	Selection Sort	
2	1888	1888	
3	2643	755	
4	2266	755	
5	2265	755	
6	2643	755	
7	2643	755	
8	3020	755	
9	4154	755	
10	3398	755	
11	4154	1132	
12	5286	1510	
13	6041	1511	
14	6797	1888	
15	6041	1888	
16	6419	1888	
17	7551	2266	
18	9817	3399	

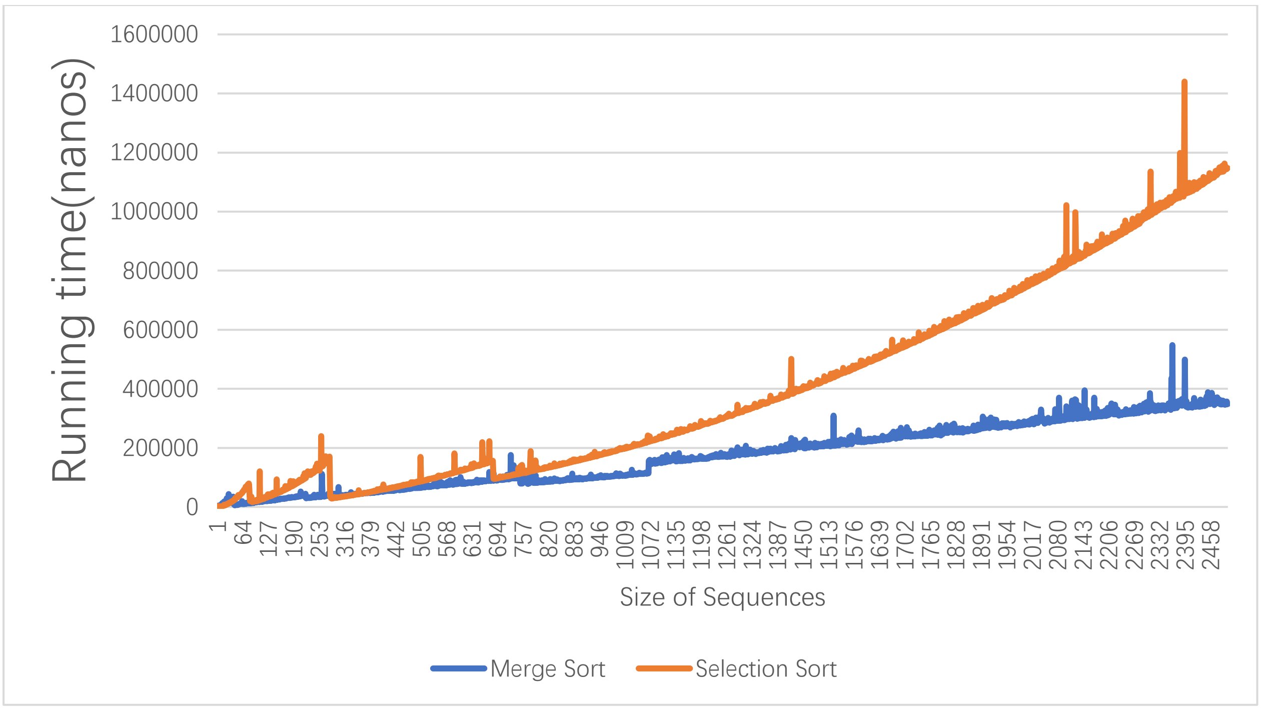
Test and Analyse

Time complexity is analysed by two ways. First way is to check the running time of two algorithms

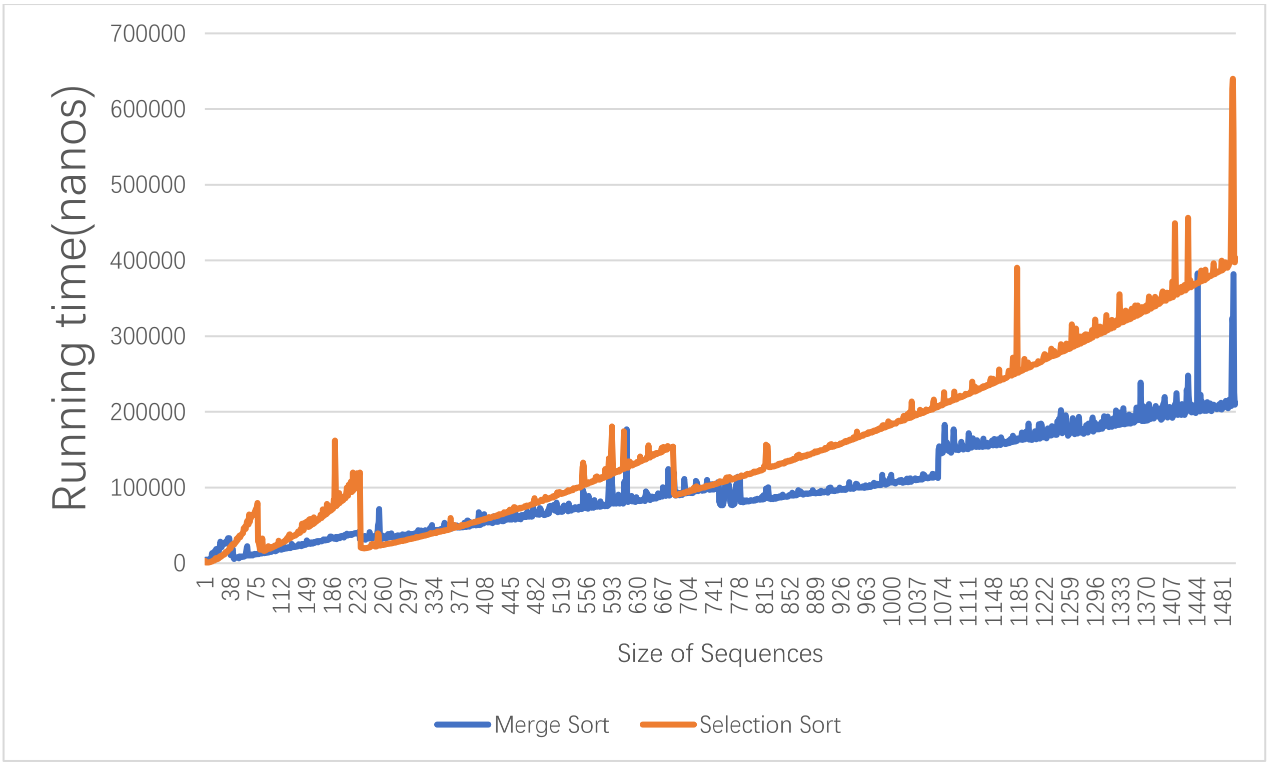


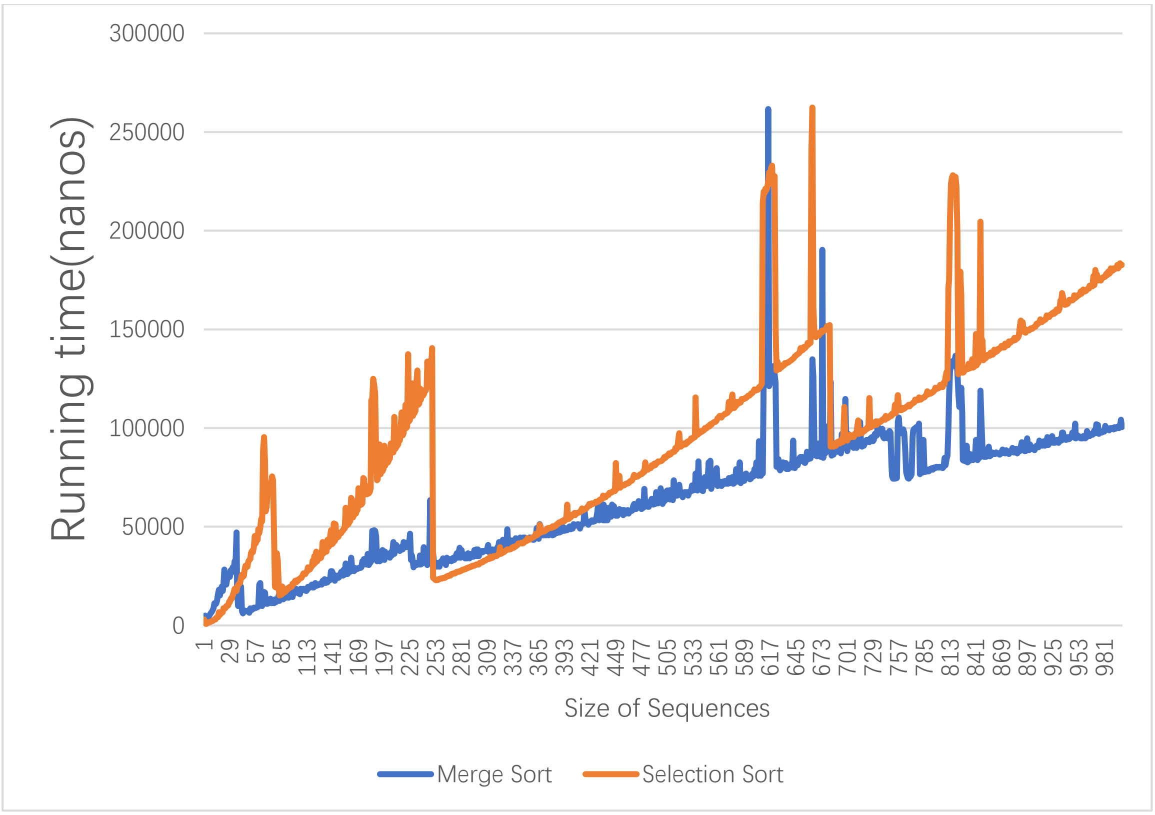
To look for an asymptotic time complexity, size of the problem needs to get significantly large. As graph above indicates, the merge sort is faster algorithm than selection sort when the array size is quite large ---------- when the size of sequence is increased to exactly 10,000.

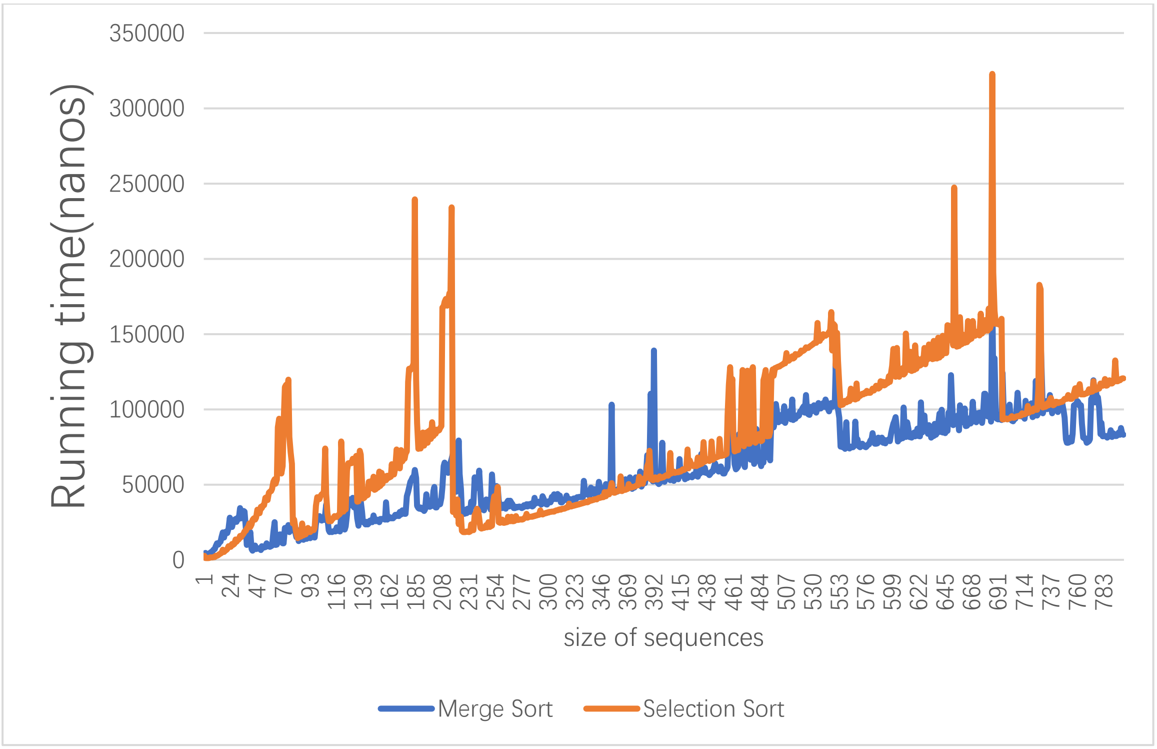




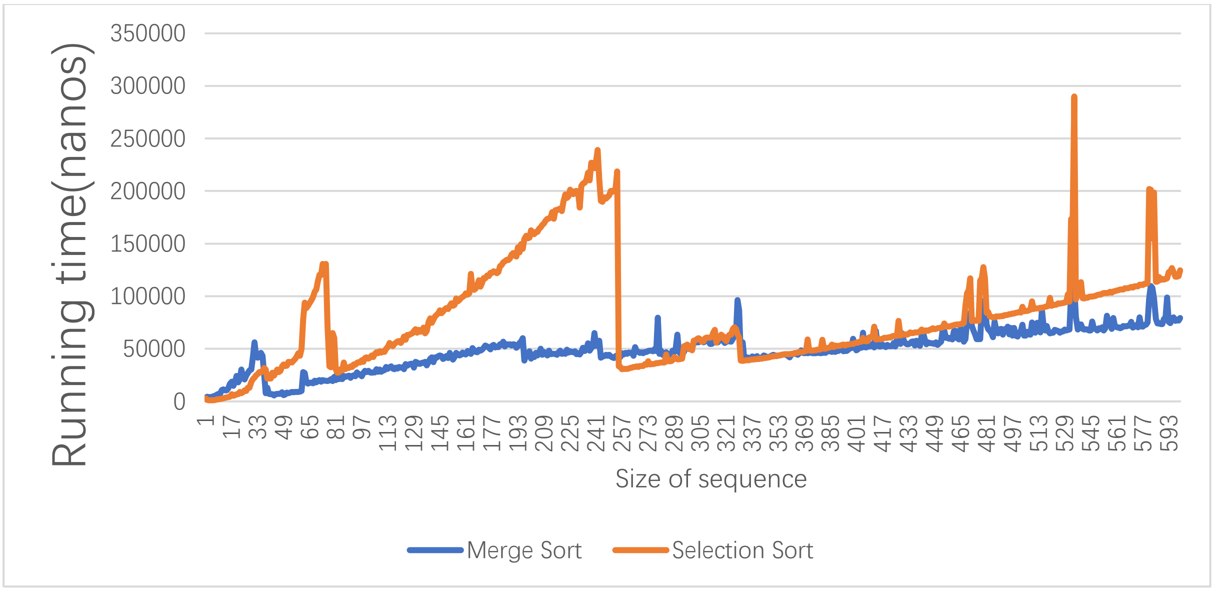
Reducing number of arrays generated by programs by adjusting the times of looping to get a higher resolution of graph. From the two graphs above, merge sort is not always faster than selection sort when the array size is smaller than around 350 elements. Therefore, the array size needs to be reduced to get higher resolution for further analysing.

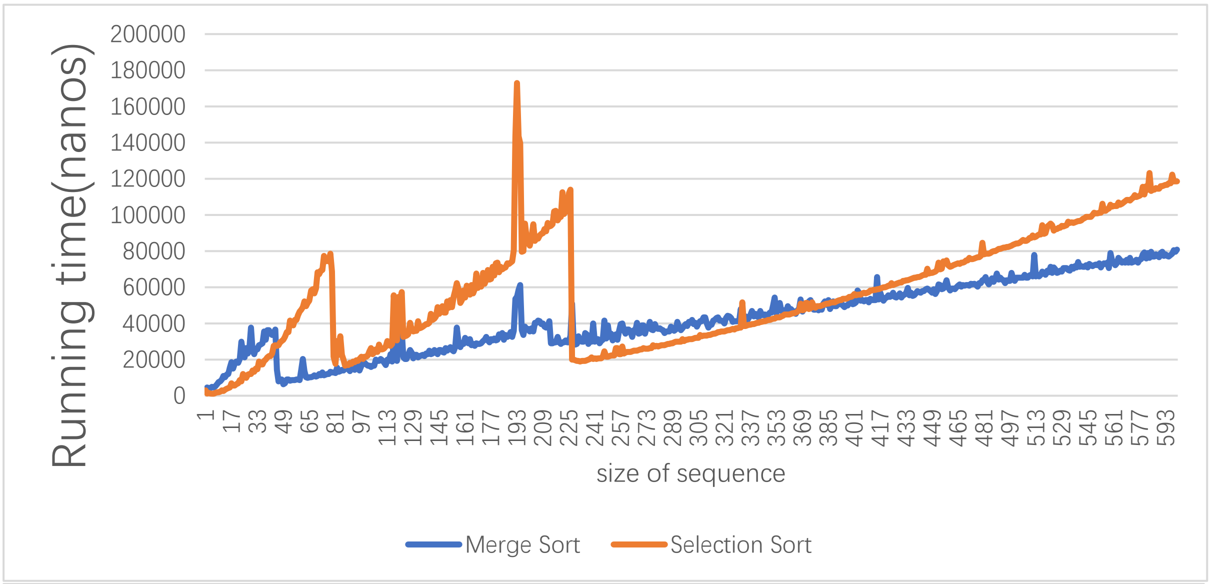


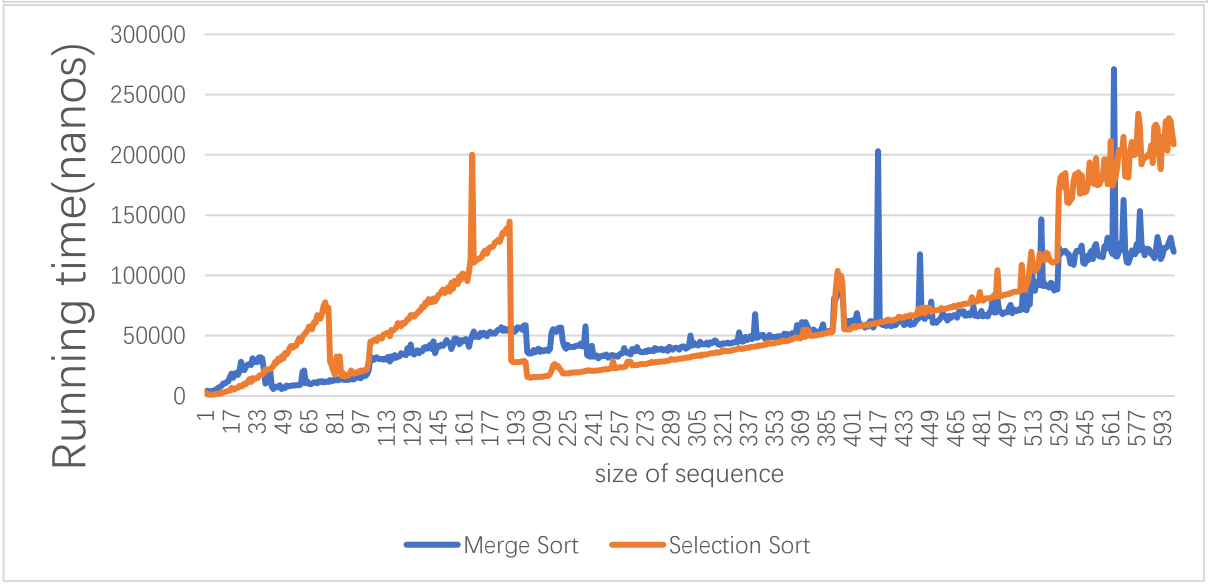




With higher resolution, the time complexity of two algorithms over the sequences with relatively small sizes is shown by the diagram above. The merge sort does sort slower than selection sort when sequence has size smaller than around 380 despite there is large vibration of running time especially the one with selection sort.


Running time(nanos)
350000
300000
250000
200000
150000
100000
50000
0
1 17 33 49 65 81 97 113 129 145 161 177 193 209 225 241 257 273 289 305 321 337 353 369 385 401 417 433 449 465 481 497 513 529 545 561 577 593
Size of sequence
Merge Sort
Selection Sort


Running time(nanos)
200000
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
1 17 33 49 65 81 97 113 129 145 161 177 193 209 225 241 257 273 289 305 321 337 353 369 385 401 417 433 449 465 481 497 513 529 545 561 577 593
size of sequence
Merge Sort
Selection Sort


Running time(nanos)
300000
250000
200000
150000
100000
50000
0
1 17 33 49 65 81 97 113 129 145 161 177 193 209 225 241 257 273 289 305 321 337 353 369 385 401 417 433 449 465 481 497 513 529 545 561 577 593
size of sequence
Merge Sort
Selection Sort

To analyse the trend of running time of two algorithms, resolution is increased furthermore with three trials. Some conclusion could be driven to by general features of three graphs above.

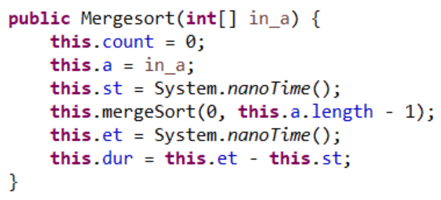
Selection sort has faster running time than merge sort in smaller size of sequences. At least between the size of 0 and 31. Hence running time of selection sort increase rapidly until the size of arrays increased to around 200. Selection sort then become faster than merge sort again linearly. When the array size increased over around 395, the merge sort finally becomes faster algorithms permanently (As we can observe from the graph with more sequences with different size). Therefore, 390 is the crossover point.

Considering the time complexity of merge sort and selection sort, merge sort has complexity of n log(n) in best, average and worst case, for selection sort it is n^2. Theoretically, even though they share the general features from the graph plotted from data collected above, there are still some differences. Selection sort is faster algorithm than merge sort when the size of sequence is small generally hence there is a crossover and then merge sort become faster than sequence sort. However, in the graph, the trend of running time of selection sort is quite different from expected trending.

Some external factor might affect the running time. Therefore, execution time is considered to analyse the time complexity of two algorithms.

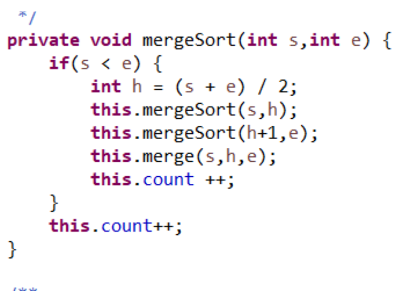
Design and Implementation (Execution time counting)

Again, some basic idea of implementation will be introduced.



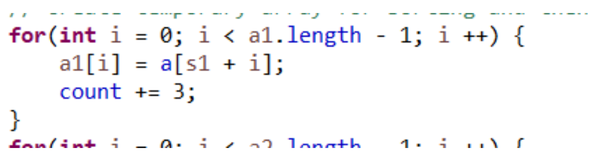
```
public Mergesort(int[] in_a) {
    this.count = 0;
    this.a = in_a;
    this.st = System.nanoTime();
    this.mergeSort(0, this.a.length - 1);
    this.et = System.nanoTime();
    this.dur = this.et - this.st;
}
```

Before sorting, count of times of execution is initialised. It is same in selection sot.



```
 */
private void mergeSort(int s,int e) {
    if(s < e) {
        int h = (s + e) / 2;
        this.mergeSort(s,h);
        this.mergeSort(h+1,e);
        this.merge(s,h,e);
        this.count ++;
    }
    this.count++;
}
 /**
```

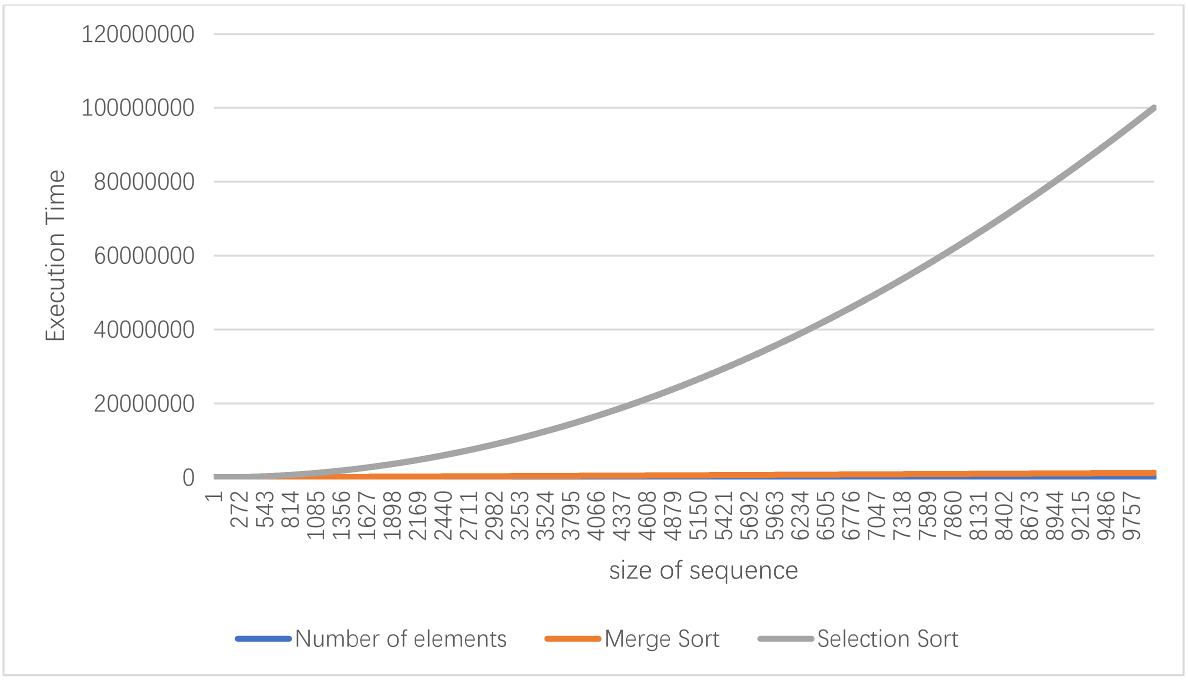
Each execution such as conditional comparison and calculation will be consider as one execution. In merge function, variable will count every execution that will takes place inside the function. This counting is same in selection sort.



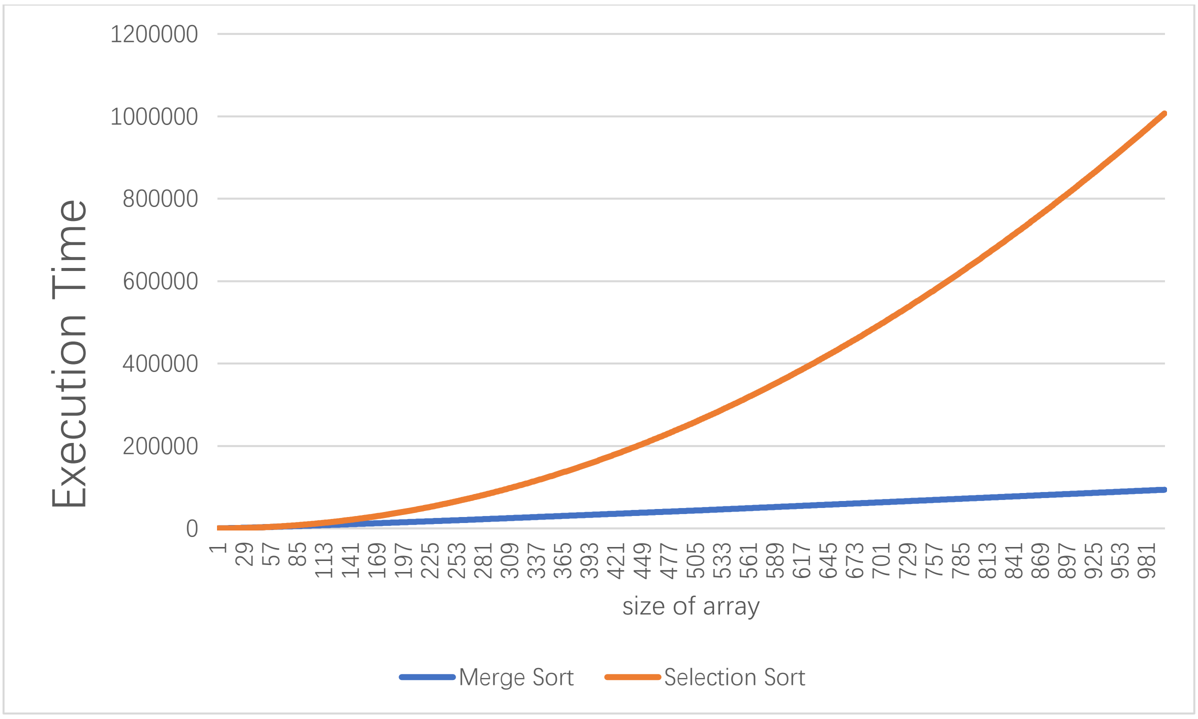
```
for(int i = 0; i < a1.length - 1; i ++) {
    a1[i] = a[s1 + i];
    count += 3;
}
```

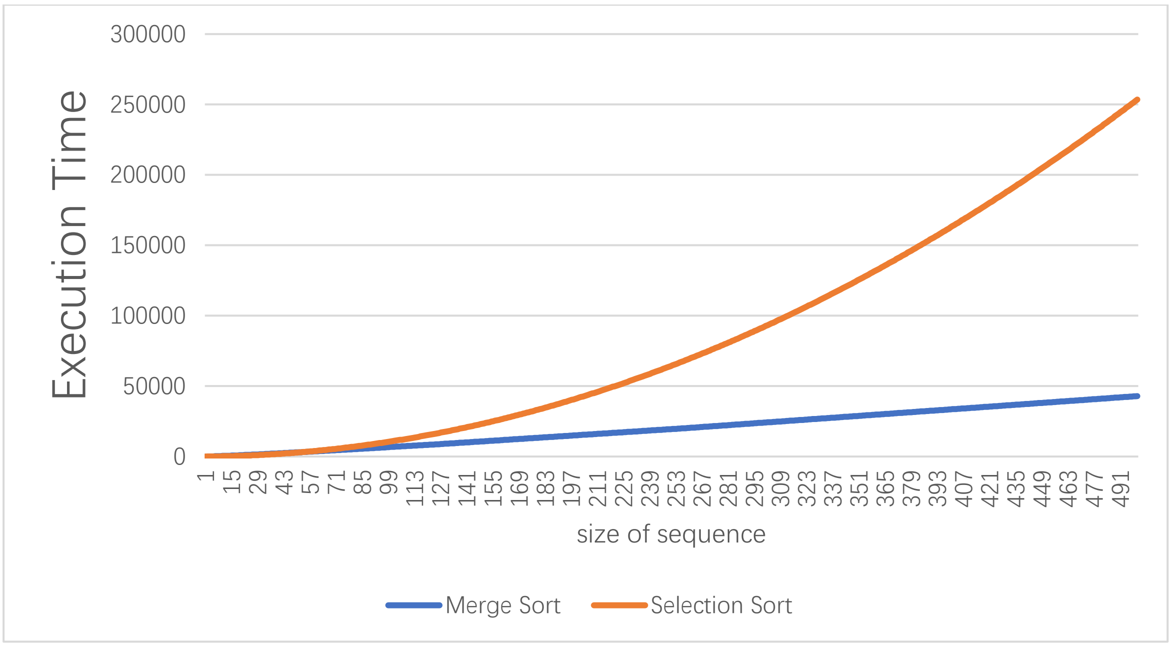
Use for loop as an example, for loop has conditional comparison and increment then has following execution in each turn, therefor in each round, three times is counted into execution times.

Testing and Analyse

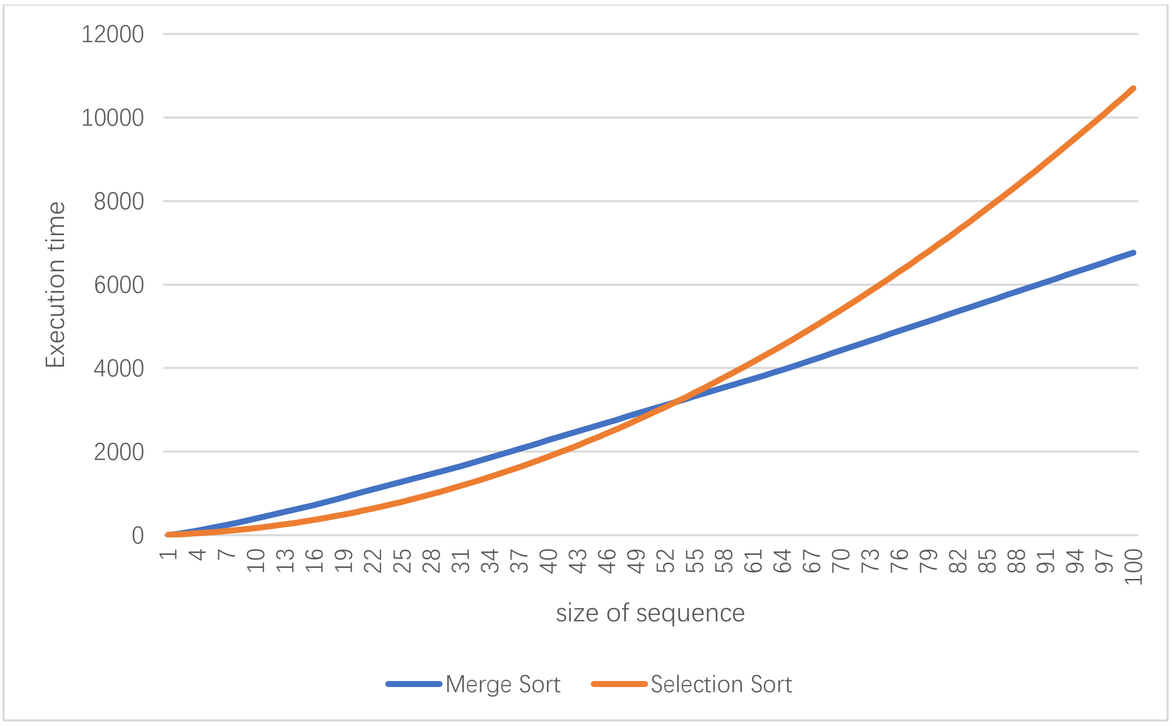


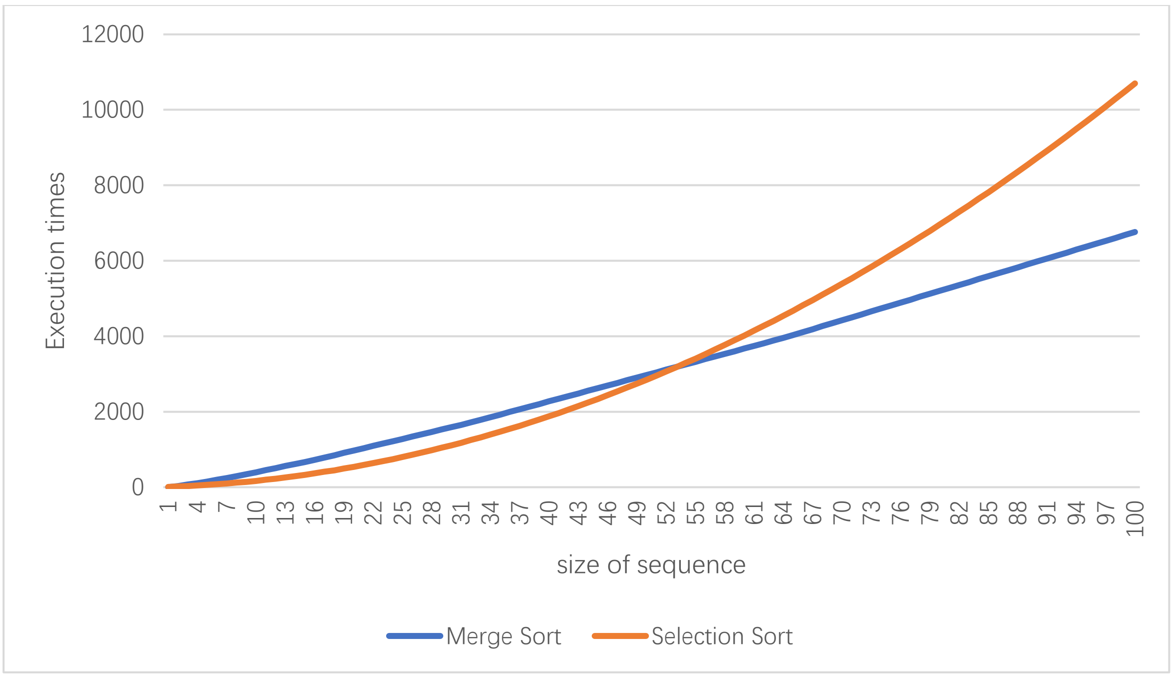
Starting from 10,000 sequences with increasing sizes, the curve become smoother than running time graph. It clearly shows merge sort is faster algorithms than selection sort in trends to infinity.

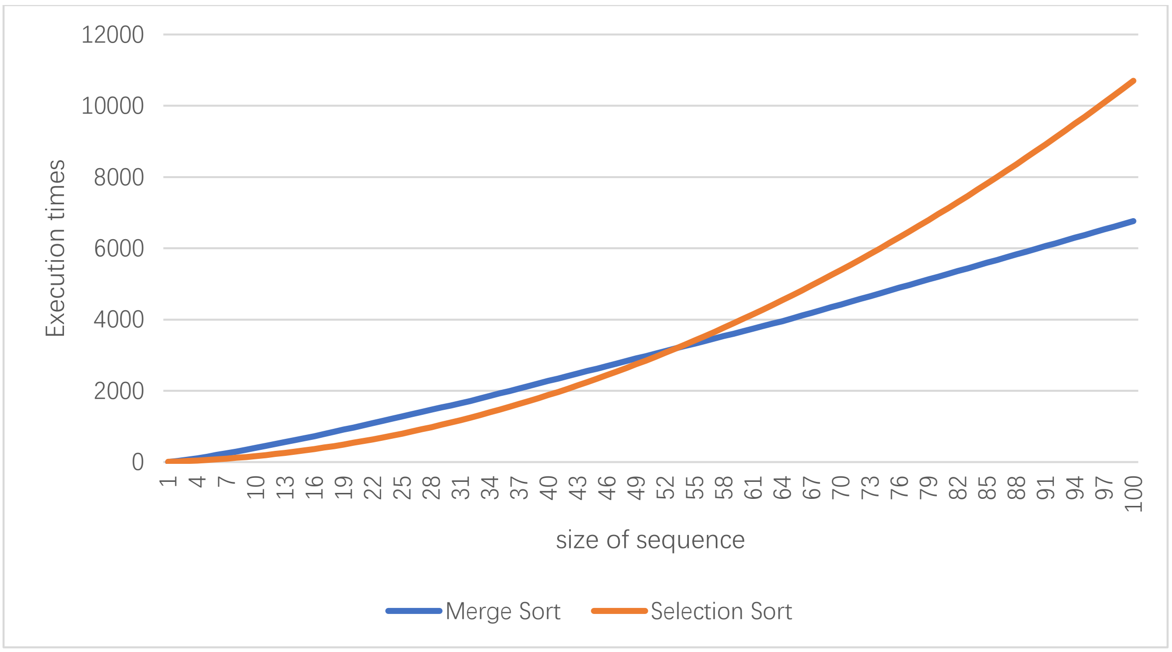




With increasing resolution, selection sort is slower than merge sort when the size of sequence is relatively large stably. There is no exceptional vibration on the curve. The selection sort has more times of executions than merge sort when the size of sequence increased.





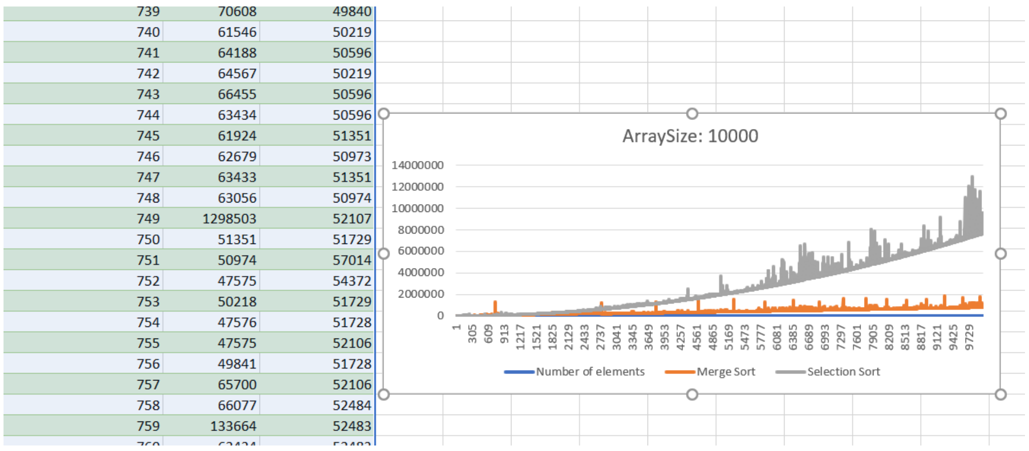


Two algorithms are tested with three trials in random sequences with same size of 100. In fact, their execution times are the same respectively. The time complexity of two algorithms could be described by the graph by the features on the graph. Selection sort is faster than merge sort in sorting random sequences with smaller size. The crossover point is 52, any array with bigger sizes would be sorted by merge sorts.

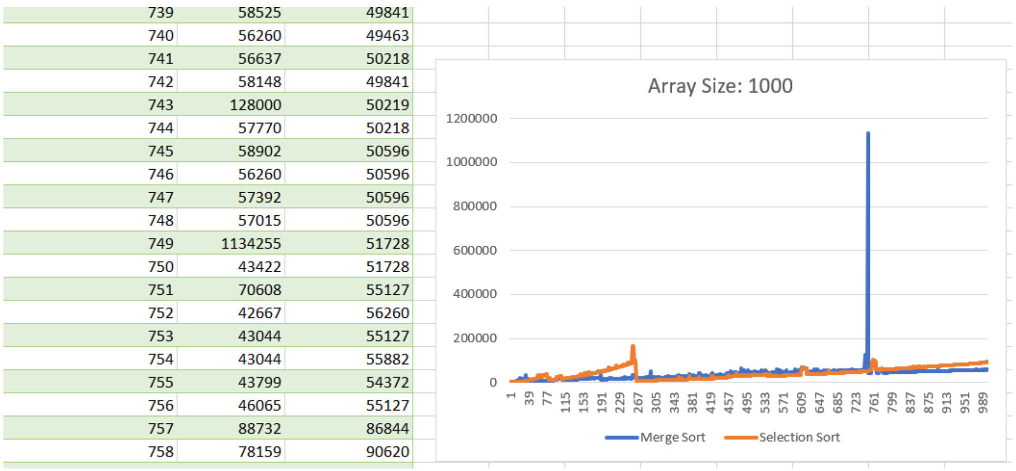
Evaluation

Some problems are raised from the testing results. Firstly, the crossover point of running time graph and execution graph is quite different in terms of their size of sequences. Although graph of running time obtains a crossover point in 39 which is nearly to the crossover point 53 from execution time graph. However, the running time of selection sort drop down suddenly around 190, then rise gently over the running time of merge sort around the crossover point around 395.

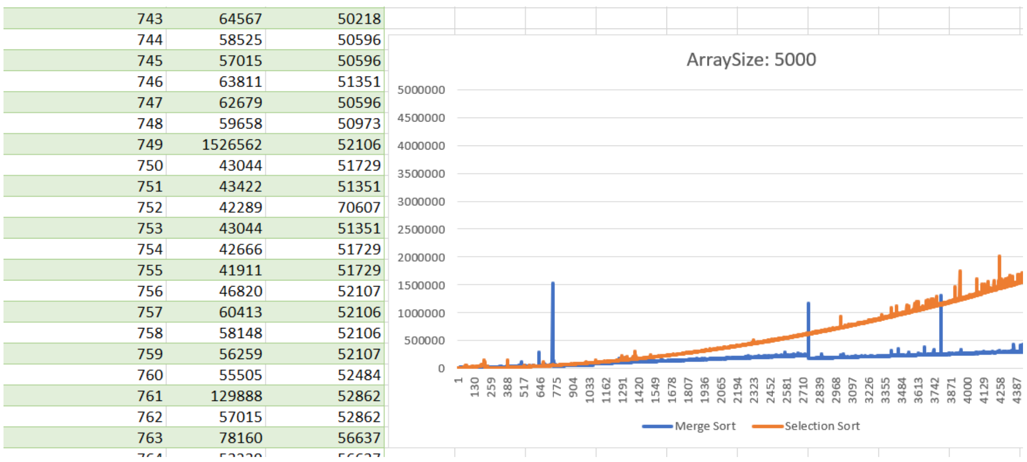
Moreover, there are some interesting pathological case collected in early data collection of running time. For the merge sort, its running time would rise rapidly when the size of sequences is nearly 750 whatever the random sequences is generated.



739	70608	49840
740	61546	50219
741	64188	50596
742	64567	50219
743	66455	50596
744	63434	50596
745	61924	51351
746	62679	50973
747	63433	51351
748	63056	50974
749	1298503	52107
750	51351	51729
751	50974	57014
752	47575	54372
753	50218	51729
754	47576	51728
755	47575	52106
756	49841	51728
757	65700	52106
758	66077	52484
759	133664	52483



739	58525	49841
740	56260	49463
741	56637	50218
742	58148	49841
743	128000	50219
744	57770	50218
745	58902	50596
746	56260	50596
747	57392	50596
748	57015	50596
749	1134255	51728
750	43422	51728
751	70608	55127
752	42667	56260
753	43044	55127
754	43044	55882
755	43799	54372
756	46065	55127
757	88732	86844
758	78159	90620



743	64567	50218
744	58525	50596
745	57015	50596
746	63811	51351
747	62679	50596
748	59658	50973
749	1526562	52106
750	43044	51729
751	43422	51351
752	42289	70607
753	43044	51351
754	42666	51729
755	41911	51729
756	46820	52107
757	60413	52106
758	58148	52106
759	56259	52107
760	55505	52484
761	129888	52862
762	57015	52862
763	78160	56637

There is not any implementation of code afterward, however, this pathological case disappears when the java environment is restarted (restart the computer) which is represented in testing part above. Although those pathological case could be ignored in considering trending of running times as they are extremely rare cases. Some features are discovered for those pathological cases:

- Increase the clock speed of CPU will decrease the running time of those cases, but still higher than other values as ten times more

Some interesting assumption is considered,

- basics of Java Virtual Machine affected the running time
- Space complexity might affect the running time, not only the time complexity in JVM

The space complexity of merge sort is O(n), merge sort is recursive function. It requires more spaces when the problem size increases. The space complexity of selection sort is O (1).

Furthermore, some feature of memory model might be related to the varies in running time. Java variables and objects will be stored into stack and heap respectively. To provide the further memory to execute, unused variables and objects are garbage collected from the memory. It works on iterative method in selection sorting but not for recursive method like merge sorting. There are still lots of occupied memory for divided problems when the function is still in recurse. Space occupying is not a problem for a conceptual computer model, but in practice Java Virtual machine is allocated with limited memory and CPU performance, therefore memory might affect running time during processing especially for the recursive function like merging sort.

Conclusion

In the report, two methods are applied to test the time complexity of merge sort and selection sort. Basically, the trending is found by analysing collection of running time and execution times in sorting random sequences with different sizes. Some assumptions are written inside evaluation part to discover the reason for some pathological and exceptional cases inside running time graph. Some extra reading on JVM and complexity might be needed for further discovery.