N is the number of elements and the data is presented in how many milliseconds it took.

Table of random N

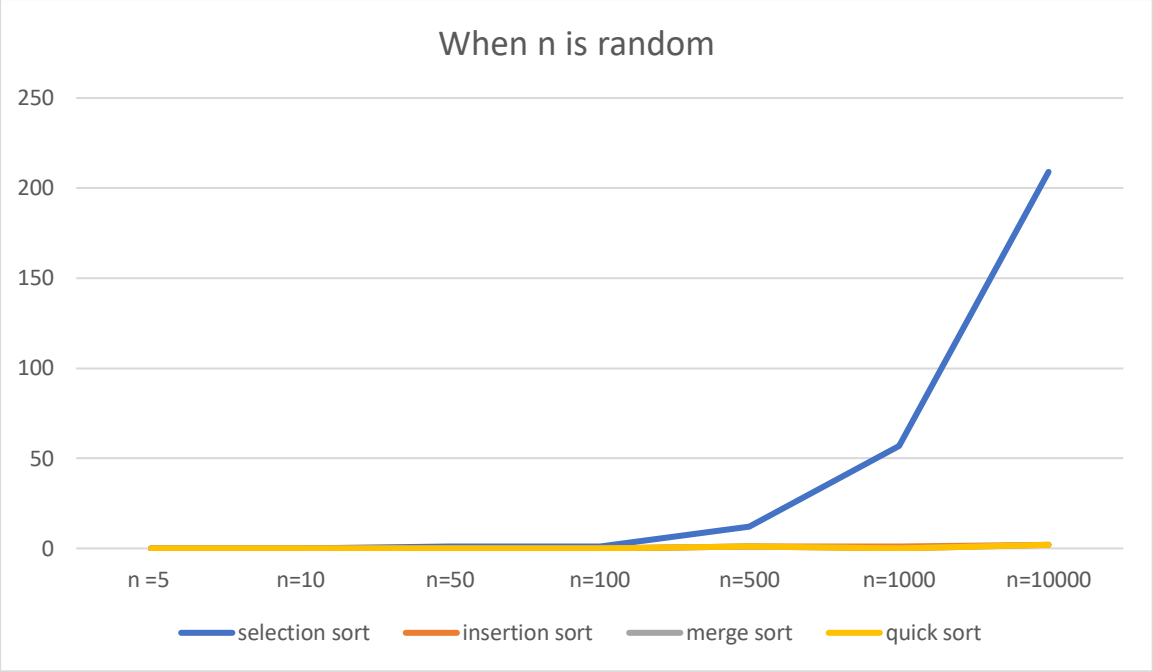
	N=5	N=10	N=50	N=100	N=500	N=1000	N=10000
Selection sort	0	0	1	1	12	57	209
Insertion sort	0	0	0	0	1	1	2
Merge sort	0	0	0	0	1	0	2
Quick sort	0	0	0	0	1	0	2

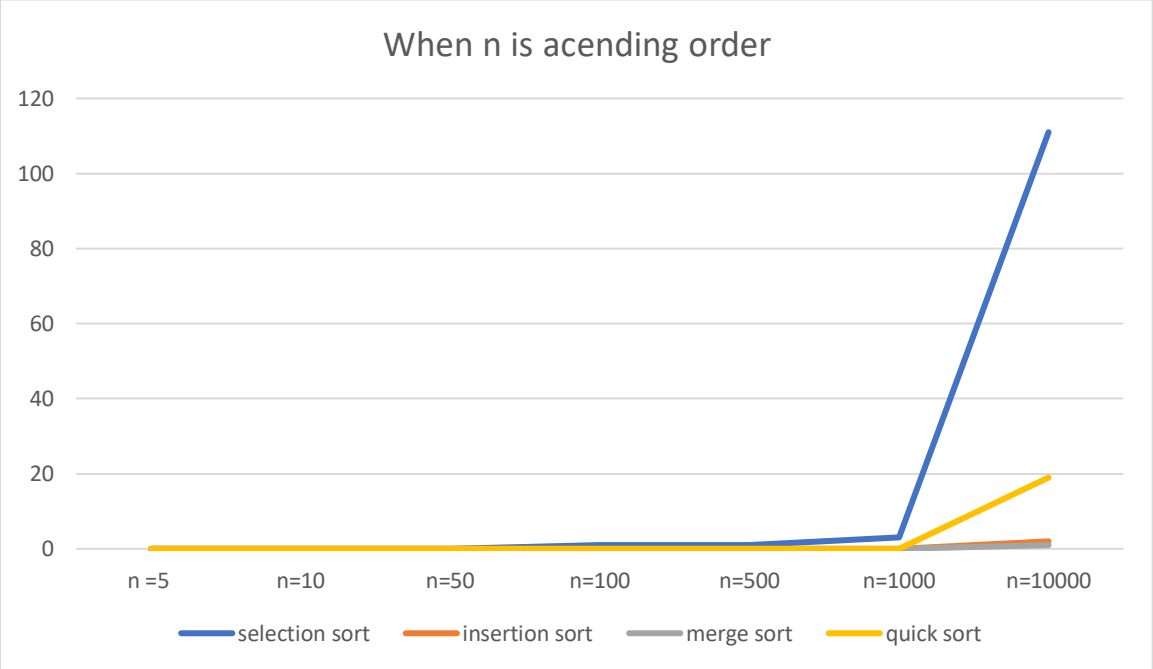
Table of ascending order of N

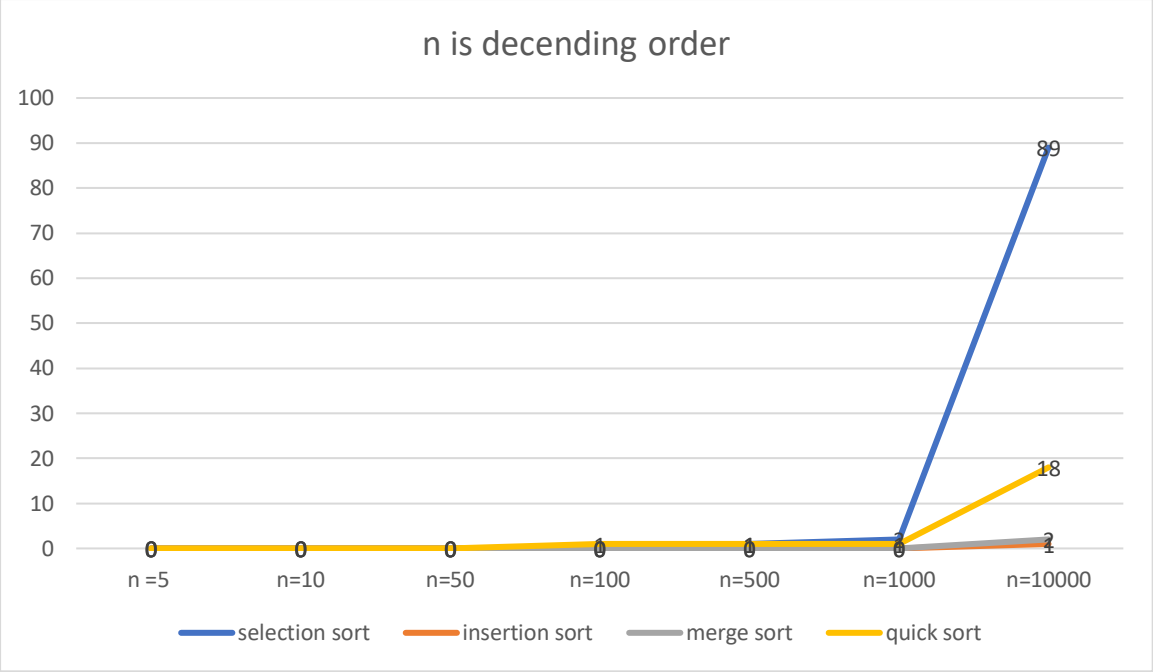
	N=5	N=10	N=50	N=100	N=500	N=1000	N=10000
Selection sort	0	0	0	1	1	3	111
Insertion sort	0	0	0	0	0	0	2
Merge sort	0	0	0	0	0	0	1
Quick sort	0	0	0	1	0	0	10

descending order of n

	N=5	N=10	N=50	N=100	N=500	N=1000	N=10000
Selection sort	0	0	0	0	1	2	89
Insertion sort	0	0	0	0	0	0	1
Merge sort	0	0	0	0	0	1	2
Quick sort	0	0	0	1	0	1	15


When n is random
250
200
150
100
50
0
n =5
n=10
n=50
n=100
n=500
n=1000
n=10000
selection sort
insertion sort
merge sort
quick sort


When n is acending order
120
100
80
60
40
20
0
n =5
n=10
n=50
n=100
n=500
n=1000
n=10000
selection sort
insertion sort
merge sort
quick sort



Just like expected according to the asymptomatic notations, selection sort has the bad big O. That is big O(n^2). That worst case is clear in our results. However, surprising insertion sort is much better than the selection even though asymptomatically both has the same worst case. The execution cost of selection increase in a way that is much bigger. Thus, one can easily conclude selection is bad choice if n is more than 50 and worst choice if n is more than 500 elements. However, it is doing better than quick sort and on par with other Sorting methods when N less than equal to 10. Merge sort has the standard big O(nlog(n)) irrespective to the sorted or random array. This point is quite clear in out results. No matter how the N number of elements, whether sorted in any order or random, merge sort made almost same time. Surprising in quick sort, has taking little longer time if N is too small but result was random. Sometimes its notable and sometimes not and performance is varying. Unlike, merge sort, providing similar result, Quick sort noticeable doing great in when the array is random than sorted. Also, quick sort seems to take longer time either because of stack method implementation of java or may be the N should be even larger to get quick sort more efficient.

```
import java.util.ArrayList;

import java.util.Collections;


public class SortingAlgoTimimgAnalisys {
```

```
public static void main(String[] args) {
        // TODO Auto-generated method stub
        boolean reversed = false;
        SortingAlgorithms sa = new SortingAlgorithms();
        Integer five[] = new Integer[5];
        Integer ten[] = new Integer[10];
        Integer fifty[] = new Integer[50];
        Integer hundred[] = new Integer[100];
        Integer fiveHundred[] = new Integer[500];
        Integer thausand[] = new Integer[1000];
        Integer tenThausand[] = new Integer[10000];
        long x = 0;
        long y = 0;
        ;
        //
        // for n unsorted random nubers
        SortingAlgoTimimgAnalisys ta = new SortingAlgoTimimgAnalisys();
        int flag = 1; // for case one
        int lengthofList;
        lengthofList = 5;
        five = ta.populateArray(five, lengthofList, flag);
        ta.checkFive(x, y, five, sa, reversed);

        lengthofList = 10;
        ten = ta.populateArray(ten, lengthofList, flag);
        ta.checkTen(x, y, ten, sa, reversed);

        lengthofList = 50;
        fifty = ta.populateArray(fifty, lengthofList, flag);
        ta.checkFifty(x, y, fifty, sa, reversed);
```

```
lengthofList = 100;
hundred = ta.populateArray(hundred, lengthofList, flag);
ta.checkHundred(x, y, hundred, sa, reversed);

lengthofList = 500;
fiveHundred = ta.populateArray(fiveHundred, lengthofList, flag);
ta.checkFiveHundred(x, y, fiveHundred, sa, reversed);

lengthofList = 1000;
thausand = ta.populateArray(thausand, lengthofList, flag);
ta.checkThausand(x, y, thausand, sa, reversed);

lengthofList = 10000;
tenThausand = ta.populateArray(tenThausand, lengthofList, flag);
ta.checkTenThausand(x, y, tenThausand, sa, reversed);

// n sorted incrementing order
flag = 2;
lengthofList = 5;
five = ta.populateArray(five, lengthofList, flag);
ta.checkFive(x, y, five, sa, reversed);

lengthofList = 10;
ten = ta.populateArray(ten, lengthofList, flag);
ta.checkTen(x, y, ten, sa, reversed);

lengthofList = 50;
fifty = ta.populateArray(fifty, lengthofList, flag);
ta.checkFifty(x, y, fifty, sa, reversed);
```

```
lengthofList = 100;
hundred = ta.populateArray(hundred, lengthofList, flag);
ta.checkHundred(x, y, hundred, sa, reversed);

lengthofList = 500;
fiveHundred = ta.populateArray(fiveHundred, lengthofList, flag);
ta.checkFiveHundred(x, y, fiveHundred, sa, reversed);

lengthofList = 1000;
thausand = ta.populateArray(thausand, lengthofList, flag);
ta.checkThausand(x, y, thausand, sa, reversed);

lengthofList = 10000;
tenThausand = ta.populateArray(tenThausand, lengthofList, flag);
ta.checkTenThausand(x, y, tenThausand, sa, reversed);

// n sorted decreasing order
                flag = 3;
                lengthofList = 5;
                five = ta.populateArray(five, lengthofList, flag);
                ta.checkFive(x, y, five, sa, reversed);

                lengthofList = 10;
                ten = ta.populateArray(ten, lengthofList, flag);
                ta.checkTen(x, y, ten, sa, reversed);

                lengthofList = 50;
                fifty = ta.populateArray(fifty, lengthofList, flag);
                ta.checkFifty(x, y, fifty, sa, reversed);

                lengthofList = 100;
```

```
                    hundred = ta.populateArray(hundred, lengthofList, flag);
                    ta.checkHundred(x, y, hundred, sa, reversed);

                    lengthofList = 500;
                    fiveHundred = ta.populateArray(fiveHundred, lengthofList, flag);
                    ta.checkFiveHundred(x, y, fiveHundred, sa, reversed);

                    lengthofList = 1000;
                    thausand = ta.populateArray(thausand, lengthofList, flag);
                    ta.checkThausand(x, y, thausand, sa, reversed);

                    lengthofList = 10000;
                    tenThausand = ta.populateArray(tenThausand, lengthofList, flag);
                    ta.checkTenThausand(x, y, tenThausand, sa, reversed);
}

Integer[] populateArray(Integer array[], int lengthofList, int flag) {
        ArrayList<Integer> list = new ArrayList<Integer>();
        for (int i = 0; i < lengthofList; i++) {
                list.add(new Integer(i + i));
        }
        if (flag == 1) { // for random thing in case 1
                Collections.shuffle(list);
        }
        if (flag == 1 || flag == 2) {
                for (int i = 0; i < lengthofList; i++) {
                        array[i] = list.get(i);
                }
        }
        if (flag == 3) {// for decresing order
                int j=0;
```

```
                for (int i = lengthofList - 1; i >= 0; i--) {
                        array[j] = list.get(i);
                        j++;
                }
        }
        list.clear();
        return array;

}

void checkFive(long x, long y, Integer five[], SortingAlgorithms sa, Boolean reversed) {
        x = System.currentTimeMillis();
        sa.selectionSort(five, reversed);
        y = System.currentTimeMillis();
        System.out.println("selection sort of 5 :  " + (y - x));
        x = System.currentTimeMillis();
        sa.insertionSort(five, reversed);
        y = System.currentTimeMillis();
        System.out.println("insertion sort 5 :  " + (y - x));
        x = System.currentTimeMillis();
        sa.mergeSort(five, reversed);
        y = System.currentTimeMillis();
        System.out.println("mergeSort 5 :  " + (y - x));
        x = System.currentTimeMillis();
        sa.quickSort(five, reversed);
        y = System.currentTimeMillis();
        System.out.println("quickSort 5 :  " + (y - x));

}

void checkTen(long x, long y, Integer ten[], SortingAlgorithms sa, Boolean reversed) {
```

```
        x = System.currentTimeMillis();
        sa.selectionSort(ten, reversed);
        y = System.currentTimeMillis();
        System.out.println("selection sort of 10 :  " + (y - x));
        x = System.currentTimeMillis();
        sa.insertionSort(ten, reversed);
        y = System.currentTimeMillis();
        System.out.println("insertion sort 10 :  " + (y - x));
        x = System.currentTimeMillis();
        sa.mergeSort(ten, reversed);
        y = System.currentTimeMillis();
        System.out.println("mergeSort 10 :  " + (y - x));
        x = System.currentTimeMillis();
        sa.quickSort(ten, reversed);
        y = System.currentTimeMillis();
        System.out.println("quickSort 10 :  " + (y - x));
}

void checkFifty(long x, long y, Integer fifty[], SortingAlgorithms sa, Boolean reversed) {
        x = System.currentTimeMillis();
        sa.selectionSort(fifty, reversed);
        y = System.currentTimeMillis();
        System.out.println("selection sort of 50 :  " + (y - x));
        x = System.currentTimeMillis();
        sa.insertionSort(fifty, reversed);
        y = System.currentTimeMillis();
        System.out.println("insertion sort 50 :  " + (y - x));
        x = System.currentTimeMillis();
        sa.mergeSort(fifty, reversed);
        y = System.currentTimeMillis();
        System.out.println("mergeSort 50 :  " + (y - x));
```

```
            x = System.currentTimeMillis();
            sa.quickSort(fifty, reversed);
            y = System.currentTimeMillis();
            System.out.println("quickSort 50 :  " + (y - x));
        }

        void checkHundred(long x, long y, Integer hundred[], SortingAlgorithms sa, Boolean
reversed) {
            x = System.currentTimeMillis();
            sa.selectionSort(hundred, reversed);
            y = System.currentTimeMillis();
            System.out.println("selection sort of 100 :  " + (y - x));
            x = System.currentTimeMillis();
            sa.insertionSort(hundred, reversed);
            y = System.currentTimeMillis();
            System.out.println("insertion sort 100 :  " + (y - x));
            x = System.currentTimeMillis();
            sa.mergeSort(hundred, reversed);
            y = System.currentTimeMillis();
            System.out.println("mergeSort 100 :  " + (y - x));
            x = System.currentTimeMillis();
            sa.quickSort(hundred, reversed);
            y = System.currentTimeMillis();
            System.out.println("quickSort 100 :  " + (y - x));
        }

        void checkFiveHundred(long x, long y, Integer fiveHundred[], SortingAlgorithms sa, Boolean
reversed) {
            x = System.currentTimeMillis();
            sa.selectionSort(fiveHundred, reversed);
            y = System.currentTimeMillis();
            System.out.println("selection sort of 500 :  " + (y - x));
```

```
                x = System.currentTimeMillis();
                sa.insertionSort(fiveHundred, reversed);
                y = System.currentTimeMillis();
                System.out.println("insertion sort 500 :  " + (y - x));
                x = System.currentTimeMillis();
                sa.mergeSort(fiveHundred, reversed);
                y = System.currentTimeMillis();
                System.out.println("mergeSort 500 :  " + (y - x));
                x = System.currentTimeMillis();
                sa.quickSort(fiveHundred, reversed);
                y = System.currentTimeMillis();
                System.out.println("quickSort 500 :  " + (y - x));

        }

        void checkThausand(long x, long y, Integer thausand[], SortingAlgorithms sa, Boolean
reversed) {
                x = System.currentTimeMillis();
                sa.selectionSort(thausand, reversed);
                y = System.currentTimeMillis();
                System.out.println("selection sort of 1000 :  " + (y - x));
                x = System.currentTimeMillis();
                sa.insertionSort(thausand, reversed);
                y = System.currentTimeMillis();
                System.out.println("insertion sort 1000 :  " + (y - x));
                x = System.currentTimeMillis();
                sa.mergeSort(thausand, reversed);
                y = System.currentTimeMillis();
                System.out.println("mergeSort 1000 :  " + (y - x));
                x = System.currentTimeMillis();
                sa.quickSort(thausand, reversed);
```

```
            y = System.currentTimeMillis();
            System.out.println("quickSort 1000 :  " + (y - x));
        }

        void checkTenThausand(long x, long y, Integer tenThausand[], SortingAlgorithms sa, Boolean reversed) {
            x = System.currentTimeMillis();
            sa.selectionSort(tenThausand, reversed);
            y = System.currentTimeMillis();
            System.out.println("selection sort of 10000 :  " + (y - x));
            x = System.currentTimeMillis();
            sa.insertionSort(tenThausand, reversed);
            y = System.currentTimeMillis();
            System.out.println("insertion sort 10000 :  " + (y - x));
            x = System.currentTimeMillis();
            sa.mergeSort(tenThausand, reversed);
            y = System.currentTimeMillis();
            System.out.println("mergeSort 10000 :  " + (y - x));
            x = System.currentTimeMillis();
            sa.quickSort(tenThausand, reversed);
            y = System.currentTimeMillis();
            System.out.println("quickSort 10000 :  " + (y - x));

        }

}
```