language=Java, breaklines=true

UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

University of Witwatersrand
Johannesburg, South Africa

Advanced Analysis of Algorithms

Tamlin Love

1438243

Lab Exam

Semester 2, 2018

1 Question 1

1.1 A

The algorithm presented is a sorting algorithm. It operates by moving through the list, A, from left to right until it encounters a number smaller than that in the "pivot position" (denoted by a, initially 0). If no such number exists then the pivot position shifts and the algorithm resumes.
In essence, it creates a sorted sublist at the beginning of the list and steadily inserts numbers to the right of that sublist into the sublist at the correct position.

1.2 B

For this analysis, we take array operations (e.g. indexing and setting elements of an array) to be our basic operation. Equivalently, we can count the number of times each for-loop and while-loop runs.
We begin by considering the case when the list is already sorted. In this case, the condition $x < A[c]$ is always false as $c < i$. Thus the else condition is always executed. After the final execution of the while-loop, $a = b = i$. Thus the second for-loop only executes once per iteration of the first loop. In this case, the while loop runs approximately $lg(n)$ times per iteration of the outer for-loop. Overall, this case is $\mathcal{O}(nlg(n))$ and is our best case, as it minimises the number of array operations.
Similarly, the worst case occurs when the list is sorted in reverse order. In this case, the condition $x < A[c]$ is always true. Thus the value of a does not change, which forces the second for-loop to run i times. Since the outer loop runs $n - 1$ times and the while loop runs approximately $lg(n)$ times, we have that the complexity of the algorithm in this case is $\mathcal{O}(nlg(n) + n^2)$, which is equivalent to $\mathcal{O}(n^2)$.

1.3 C

Refer to section (2) for the source code used to implement the algorithm and test against inputs.
Three tests were conducted. In all tests, $1000 \leq n \leq 150000$, with n incrementing in steps of 1000. In the first test, the arrays were generated with elements in ascending order, simulating the best case. In the second test, the elements were in descending order, simulating the worst case. Finally, arrays with elements in random order were used, simulating the average case.

1.4 D

For all three tests, the time taken to execute the algorithm was recorded. Plotting the time (in miliseconds) against n yields the following graph.

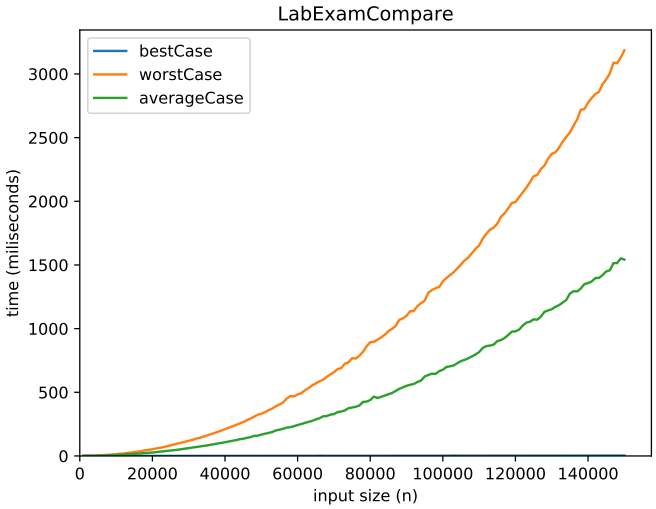

Figure 1: Time (miliseconds) against n

Clearly, the above graph supports the analysis in (B), as the worst case and best case graphs take the forms we would expect from the theoretical analysis. As the best case grows far slower than the other cases, it is difficult to represent alongside them. Thus figure (2) is included for visual clarity.
Examining the graph of the average case, we note a parabolic shape. Thus the graph implies that the average case complexity of the algorithm is $\mathcal{O}(n^2)$.

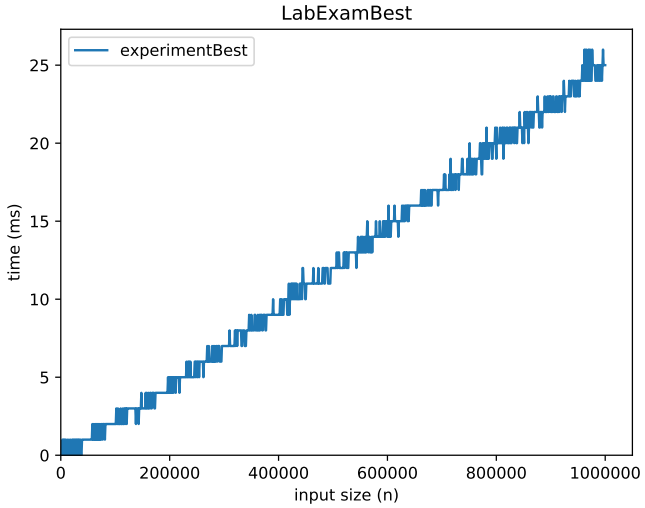

Figure 2: Time (miliseconds) against n for the best case, included for clarity. The fluctuations along the linear line are due to the error inherent to such small time scales

2 Code

```
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;

public class Program {

        static int start = 1000;
        static int inc = 1000;
        static int end = 150000;

        public static void main(String args[]){
                ArrayList<Long> dursB = new ArrayList<>();
                ArrayList<Long> dursW = new ArrayList<>();
                ArrayList<Long> dursA = new ArrayList<>();
                for(int n=start;n<=end;n=n+inc){
                        //Best
                        int[] arr = ascendingList(n);
                        long startTime = System.currentTimeMillis();
                        int[] sol = algorithm(arr,n);
                        long elapsedTime = (new Date()).getTime() - startTime;
                        dursB.add(elapsedTime);
                        //Worst
                        int[] arr2 = descendingList(n);
                        long startTime2 = System.currentTimeMillis();
                        int[] sol2 = algorithm(arr2,n);
                        long elapsedTime2 = (new Date()).getTime() - startTime2
                        dursW.add(elapsedTime2);
                        //Average
                        int[] arr3 = randomArray(n);
                        long startTime3 = System.currentTimeMillis();
                        int[] sol3 = algorithm(arr3,n);
                        long elapsedTime3 = (new Date()).getTime() - startTime3
                        dursA.add(elapsedTime3);
                        print("n="+n);
                }
                writeToCSV(dursB,"bestCase.csv");
                writeToCSV(dursW,"worstCase.csv");
                writeToCSV(dursA,"averageCase.csv");

        }

        public static int[] algorithm(int[] A,int n){
                for(int i=1;i<n;i++){
                        int x = A[i];
                        int a = 0;
                        int b = i;
```

```
                while(a<b){
                        int c = (a+b)/2;
                        if(x<A[c]){
                                b = c;
                        }
                        else{
                                a = c+1;
                        }
                }
                for(int j=i;j>a;j--){
                        A[j] = A[j-1];
                }
                A[a] = x;
                //print(A);
        }
        return A;
}

public static void print(int[] arr){
        String x = "";
        for(int i=0;i<arr.length;i++){
                x+=arr[i]+",";
        }
        System.out.println(x);
}

public static void print(int x){
        System.out.println(x);
}

public static void print(String x){
        System.out.println(x);
}

public static int[] randomArray(int n){
        int[] arr = ascendingList(n);
        return shuffleArray(arr);
}

public static int[] shuffleArray(int[] arr){
        ArrayList<Integer> alist = new ArrayList<>();
        for(int i=0;i<arr.length;i++){
                alist.add(arr[i]);
        }
        Collections.shuffle(alist);
        for(int i=0;i<arr.length;i++){
                arr[i] = alist.get(i);
        }
        return arr;
}
```

```
        public static int[] ascendingList(int n){
                int[] arr = new int[n];
            for(int i=0;i<n;i++){
              arr[i] = i;
            }
            return arr;
        }

        public static int[] descendingList(int n){
                int[] arr = new int[n];
            for(int i=0;i<n;i++){
              arr[i] = n-1-i;
            }
            return arr;
        }

        public static void writeToCSV(ArrayList<Long> durs, String name){
                try{
                  PrintWriter writer = new PrintWriter(name, "UTF-8");
                  writer.println("input,time");
                  for (int i=0;i<durs.size();i++){
                    writer.println(start + inc*i+","+durs.get(i));
                  }
                  writer.close();
                }
                catch(Exception e){
                        print("Woops");
                }
        }
}
```