Exercise 1. Answer Sheet

Student's Name: Yuta Nemoto Student's ID: s1240234

Problem 1. *(30 points)* For each function ***f(n)*** and time ***T*** in the following table, determine the largest size *n* of a problem that can be solved in time ***T***, assuming that the algorithm to solve the problem takes ***f(n)*** milliseconds.

f(n)	***T*** = 1 second	***T*** = 1 minute	***T*** = 1 hour	***T*** = 1 day	***T*** = 1 month (30 days)
$\sqrt{n}$	10^6	$3.6 * 10^9$	$1.296 * 10^{13}$	$7.46496 * 10^{15}$	$6.718464 * 10^{18}$
n	10^3	$6.0 * 10^4$	$3.6 * 10^6$	$8.64 * 10^7$	$2.592 * 10^9$
n^2	31	$2.4494897 * 10^2$	1897	$9.2951600 * 10^3$	$5.091168 * 10^4$
n^3	10	3.9148676 * 10	$1.5326188 * 10^2$	$4.4208377 * 10^2$	$1.373657 * 10^3$
2^n	9	15	21	26	31

Millisecond is 10^{-3} seconds, so
1 second = 10^3 milliseconds
1 minute = $60 * 10^3 = 6.0 * 10^4$ milliseconds
1 hour = $60 * 6.0 * 10^4 = 3.6 * 10^6$ milliseconds
1 day = $24 * 3.6 * 10^6 = 8.64 * 10^7$ milliseconds
1 month = $30 * 8.64 * 10^7 = 2.592 * 10^9$ milliseconds

Case: $f(n) = \sqrt{n}$

1. $f(n) = \sqrt{n}$ [milliseconds] <= 10^3 [milliseconds]
   $n^{1/2} <= 10^3$
   $n <= 10^6$
2. $f(n) = \sqrt{n}$ [milliseconds] <= $6.0 * 10^4$ [milliseconds]
   $n^{1/2} <= 6.0 * 10^4$
   $n <= 3.6 * 10^9$
3. $f(n) = \sqrt{n}$ [milliseconds] <= $3.6 * 10^6$ [milliseconds]
   $n^{1/2} <= 3.6 * 10^6$
   $n <= 1.296 * 10^{13}$
4. $f(n) = \sqrt{n}$ [milliseconds] <= $8.64 * 10^7$ [milliseconds]
   $n^{1/2} <= 8.64 * 10^7$
   $n <= 7.46496 * 10^{15}$
5. $f(n) = \sqrt{n}$ [milliseconds] <= $2.592 * 10^9$ [milliseconds]
   $n^{1/2} <= 2.592 * 10^9$
   $n <= 6.718464 * 10^{18}$

Case: $f(n) = n^2$

1. $f(n) = n^2$ [milliseconds] <= 10^3 [milliseconds]
   $n^2 <= 10^3$
   $n <= 10^{3/2}$
2. $f(n) = n^2$ [milliseconds] <= $6.0 * 10^4$ [milliseconds]
   $n^2 <= 6.0 * 10^4$
   $n <= 2.44948974278 * 10^2$

3. $f(n) = n^2$ [milliseconds] <= $3.6 * 10^6$ [milliseconds]
   $n^2 <= 36 * 10^5$
   $n <= 6.0 * 10^{5/2}$
4. $f(n) = n^2$ [milliseconds] <= $8.64 * 10^7$ [milliseconds]
   $n^2 <= 8.64 * 10^7$
   $n^2 <= 86.4 * 10^6$
   $n <= 9.2951600309 * 10^3$
5. $f(n) = n^2$ [milliseconds] <= $2.592 * 10^9$ [milliseconds]
   $n^2 <= 2.592 * 10^9$
   $n^2 <= 25.92 * 10^8$
   $n <= 5.09116882454 * 10^4$

Case: $f(n) = n^3$

1. $f(n) = n^3$ [milliseconds] <= 10^3 [milliseconds]
   $n^3 <= 10^3$
   $n <= 10$
2. $f(n) = n^3$ [milliseconds] <= $6.0 * 10^4$ [milliseconds]
   $n^3 <= 6.0 * 10^4$
   $n^3 <= 60 * 10^3$
   $n <= 3.914867641168864 * 10$
3. $f(n) = n^3$ [milliseconds] <= $3.6 * 10^6$ [milliseconds]
   $n^3 <= 3.6 * 10^6$
   $n <= 1.532618864787106 * 10^2$
4. $f(n) = n^3$ [milliseconds] <= $8.64 * 10^7$ [milliseconds]
   $n^3 <= 8.64 * 10^7$
   $n^3 <= 86.4 * 10^6$
   $n <= 4.420837798368464 * 10^2$
5. $f(n) = n^3$ [milliseconds] <= $2.592 * 10^9$ [milliseconds]
   $n^3 <= 2.592 * 10^9$
   $n <= 1.373657091063998 * 10^3$

Case: $f(n) = 2^n$

1. $f(n) = 2^n$ [milliseconds] <= 10^3 [milliseconds]
   $2^n <= 10^3$
   $\log_2 2^n <= \log_2 10^3$
   $n <= 3 * \log_2 10$
   $n <= 3 + 3 * \log_2 5$
   $n <= 3 + 6.96578428466$
   $n <= 9.96578428466 \approx 10$
2. $f(n) = 2^n$ [milliseconds] <= $6.0 * 10^4$ [milliseconds]
   $2^n <= 6.0 * 10^4$
   $\log_2 2^n <= \log_2(6.0 * 10^4) = \log_2(2^5 * 3 * 5^4)$
   $n <= 5 + \log_2 3 + 4 * \log_2 5$
   $n <= 15.8726748803 \approx 15.8$
3. $f(n) = 2^n$ [milliseconds] <= $3.6 * 10^6$ [milliseconds]
   $2^n <= 3.6 * 10^6$
   $\log_2 2^n <= \log_2(36 * 10^5) = \log_2(2^2 * 3^2 * 10^5) = \log_2(2^7 * 3^2 * 5^5)$
   $n <= 7 + 2 * \log_2 3 + 5 * \log_2 5$
   $n <= 21.7795654759 \approx 21.7$
4. $f(n) = 2^n$ [milliseconds] <= $8.64 * 10^7$ [milliseconds]
   $2^n <= 8.64 * 10^7$
   $\log_2 2^n <= \log_2(864 * 10^5) = \log_2(2^{10} * 3^3 * 5^5)$
   $n <= 10 + 3 * \log_2 3 + 5 * \log_2 5$
   $n <= 26.3645279766 \approx 26.4$

5. $f(n) = 2^n$ [milliseconds] <= $2.592 * 10^9$ [milliseconds]
   $2^n <= 2.592 * 10^9$
   $\log_2 2^n <= \log_2(2592 * 10^6) = \log_2(2^{11} * 3^4 * 5^6)$
   $n <= 11 + 4 * \log_2 3 + 6 * \log_2 5$
   $n <= 31.2714185722 \approx 31.3$

Problem 2. *(30 points)* Consider sorting ***n*** numbers stored in array ***A*** by first finding the smallest element of ***A*** and exchanging it with the first element of the array, i.e. ***A[1]***. Them find the second smallest element of ***A***, and exchange it with ***A[2]***. Continue in this manner for the first *n-1* elements of *A*.

a) Write a pseudo-code for this algorithm which is known as "**Selection Sort**".

```
for i = 0 to n – 2:
    min = i

    for j = i to n – 1:
        if A[j] is less than A[min]:
                min = j
        end if
    end for

    temp = A[min]
    A[min] = A[i]
    A[i] = A[min]
end for
```

b) What is the time complexity of the Selection Sort algorithm?

Let t(n) be the time complexity of the algorithm.
$t(n) = n – 1 + (n – 1)(n – 2) / 2 + 3(n – 1) = O(n^2)$

Problem 3. *(40 points)* Using the pseudo-code for **Merge Sort** algorithm given at the lecture, write a program implementing the **Merge Sort** algorithm. Use any programming language you know. Upload your source code with instructions how to compile/run it. Give the input data and the program output in the space below.

<Source Code>
MergeSort.java

```
import java.util.ArrayList;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class MergeSort {

        public MergeSort() { }

        /* merge_sort Method */
        public ArrayList<Integer> merge_sort(ArrayList<Integer> A, int arraySize){

                switch(arraySize) {
                case 0:
                case 1:
                        return A;
                default:
```

```
                        int middle = arraySize / 2;
                        // Prepare the temporary list temp
                        ArrayList<Integer> temp = new ArrayList<Integer>();
                        // Prepare the list left and right for merging
                        ArrayList<Integer> left, right;

                        // Add the 1 to middle elements of A to temp list
                        for(int i = 0; i < middle; i++) {
                                temp.add(A.get(i));
                        }
                        // Initialize the list left
                        left = (ArrayList<Integer>) merge_sort(temp, temp.size()).clone();

                        // Initialize the list temp
                        temp.clear();

                        // Add the middle to final elements of A to temp list
                        for(int i = middle; i < arraySize; i++) {
                                temp.add(A.get(i));
                        }
                        // Initialize the list right
                        right = (ArrayList<Integer>) merge_sort(temp, temp.size()).clone();

                        // Return the merged array list
                        return merge(left, right);
                }
        }

        /* merge Method */
        public ArrayList<Integer> merge(ArrayList<Integer> A, ArrayList<Integer> B){
                ArrayList<Integer> result = new ArrayList<Integer>();

                while(A.size() > 0 || B.size() > 0) {
                        if(A.size() > 0 && B.size() > 0) {
                                if(A.get(0) <= B.get(0)) {
                                        result.add(A.get(0));
                                        A.remove(0);
                                }
                                else {
                                        result.add(B.get(0));
                                        B.remove(0);
                                }
                        }
                        else if(A.size() > 0 && B.size() <= 0) {
                                result.add(A.get(0));
                                A.remove(0);
                        }
                        else if(A.size() <= 0 && B.size() > 0) {
                                result.add(B.get(0));
                                B.remove(0);
                        }
                }

                return result;
        }

        /* Require the element of the Array from the standard input and store it in array. */
        public ArrayList<Integer> doInput(ArrayList<Integer> array) {
                // Output the message
                System.out.println("Please enter the elements of the array for merge sort. \n(Elements
must be INTEGER, and must be separated by space. Input any alphabet to finish input operation.):");
                // Prepare Scanner for input
                Scanner scan = new Scanner(System.in);

                // Loop for the input
                while(true) {
                        // Read the inputed elements
                        String input = scan.next();
                        // Set the acceptable pattern by the regular expression. Minus value is also ac-
ceptable.
                        Pattern p = Pattern.compile("^-?[0-9]*$");
                        // Check whether it matches to the pattern p or not.
                        Matcher m = p.matcher(input);
                        // If the input is integer value, add it to the array. Else, end the loop.
                        if(m.find()) array.add(Integer.valueOf(input));
```

```
                        else break;
                }

                System.out.println("<Loaded elements in the Array>");
                this.showContent(array);

                return array;
        }

        /* Output the element of the Array */
        public void doOutput(ArrayList<Integer> array) {
                System.out.println("<Result of Merge Sort>");
                this.showContent(array);
        }

        /* Output the content of the Array */
        public void showContent(ArrayList<Integer> array) {
                //System.err.println("showContent is called here ");
                for(int i = 0; i < array.size(); i++) System.out.print(array.get(i) + " ");
                System.out.println();
        }

        /* Main Method */
        public static void main(String[] args) {
                ArrayList<Integer> array = new ArrayList<Integer>();
                MergeSort mergeSort = new MergeSort();
                array = mergeSort.doInput(array);
                array = mergeSort.merge_sort(array, array.size());
                mergeSort.doOutput(array);
        }
}
```

<How to compile/run>

Command: **javac MergeSort.java**

java MergeSort

Input: **1 2 3 … (The elements for sorting) … 4 5 end**

When you input the data of the array for sorting, please enter the numbers separated by space. And after you finished this operation, please input any alphabet character(In the example above, it's the string "end") to end the input operation.

<Output result>

```
Case 1 Input: 8 7 6 3 2 1 7 6 5 4 1 2 3
       Output: 1 1 2 2 3 3 4 5 6 6 7 7 8
Case 2 Input: 12 3 5 8 20 6 1 5 3 3 9
       Output: 1 3 3 3 5 5 6 8 9 12 20
```

<Actual Interface>

```
std6dc26{s1240234}75: javac MergeSort.java
Note: MergeSort.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
std6dc26{s1240234}76: java MergeSort
Please enter the elements of the array for merge sort.
(Elements must be INTEGER, and must be separated by space. Input any alphabet to finish input operation.):
8 7 6 3 2 1 7 6 5 4 1 2 3 end
<Loaded elements in the Array>
8 7 6 3 2 1 7 6 5 4 1 2 3
<Result of Merge Sort>
1 1 2 2 3 3 4 5 6 6 7 7 8
std6dc26{s1240234}77: java MergeSort
Please enter the elements of the array for merge sort.
(Elements must be INTEGER, and must be separated by space. Input any alphabet to finish input operation.):
12 3 5 8 20 6 1 5 3 3 9 end
<Loaded elements in the Array>
12 3 5 8 20 6 1 5 3 3 9
<Result of Merge Sort>
1 3 3 3 5 5 6 8 9 12 20
```