Exercise 1. Answer Sheet

Student's Name: Kodai Takihara Student's ID: s1240232

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)
$\sqrt{n}$	10^12	36*10^14	1296*10^16	746496*10^18	6718464*10^18
n	10^6	6*10^7	36*10^8	864*10^8	2592*10^9
n^2	1000	7745	60000	293938	1609968
n^3	100	391	1532	4420	13736
2^n	19	25	31	36	41

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**".

```
SELECTION-SORT(A):
  for i=1 to A.length-1
    min=i
    for j=i+1 to A.length
      if A[j] < A[min]
        min = j
      temp = A[i]
      A[i] = A[min]
      A[min] = temp
```

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

c1(n) + c2(n-1) + c3(n(n+1)/2) + c4(n-1) + c5(n-1) + c6(n-1)
=c3/2 * n^2 + (c1 + c2 + c3/2 + c4 + c5 + c6)*n + (c2 - c4 - c6)
=a*n^2 + b*n + c

Hence, Answer is 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.

```
public class MergeSort {

  public static void mergeSort(int[] list) {
    if (list.length > 1) {
      int[] firstHalf = new int[list.length / 2];
      System.arraycopy(list, 0, firstHalf, 0, list.length / 2);
      mergeSort(firstHalf);

      int secondHalfLength = list.length - list.length / 2;
      int[] secondHalf = new int[secondHalfLength];
      System.arraycopy(list, list.length / 2,
        secondHalf, 0, secondHalfLength);
        mergeSort(secondHalf);

      int[] temp = merge(firstHalf, secondHalf);
      System.arraycopy(temp, 0, list, 0, temp.length);
    }
  }

  private static int[] merge(int[] list1, int[] list2) {
    int[] temp = new int[list1.length + list2.length];

    int current1 = 0;
    int current2 = 0;
    int current3 = 0;

    while (current1 < list1.length && current2 < list2.length) {

      if (list1[current1] < list2[current2])
        temp[current3++] = list1[current1++];
      else
        temp[current3++] = list2[current2++];
    }
    while (current1 < list1.length)
      temp[current3++] = list1[current1++];
    while (current2 < list2.length)
      temp[current3++] = list2[current2++];

    return temp;
  }

  public static void main(String[] args) {
    int[] list = {5,4,7,28,3,27,327,54,1000};
    mergeSort(list);
    for (int i = 0; i < list.length; i++)
      System.out.print(list[i] + " ");
  }
}
```