CS209A – LAB2

Key Content

- Sorting
- Recursion

In this tutorial, we provide practical work related to this week's theory lecture on Sorting and Recursion. At the same time, program style and design are emphasized: see for example https://github.com/twitter-archive/commons/blob/master/src/java/com/twitter/common/styleguide.md and the discussion in the lectures about the book Code Complete by Steve McConnel for further information. Code conventions and best practices are something worth knowing about as you further develop your skills in programming.

1. Question 1 Sorting

This question requires to implement a variety of sorting algorithms and evaluate the computation complexity in terms of running time.

A helper class is provided to enable you to survey the time complexity of different algorithms on inputs of increasing size. Please see the attached file **SortRunningTimeSurvey.java**.

There some tasks and methods registered in a variable called taskList:

```
//                                  Task Name           Function Name      run times upper
static String[][] taskList = { { "InsertionSortTest", "insertionSortTime", "100000" },
                               { "BubbleSortTest",    "bubbleSortTime",    "100000" },
                               { "SelectionSortTest", "selectionSortTime", "100000" },
                               { "QuickSortTest",     "quickSortTime",     "100000" },
                               { "MergeSortTest",     "mergeSortTime",     "100000" },
                               { "HeapSortTest",      "heapSortTime",      "100000" }};
```

In order to use the helper class, it is necessary to follow the specifications for method naming in the task list array. I.e. each method has some name "xxxxx", which should be the same as the name registered in task list. The input parameter should be int, the return type should be long, the return value is the running time of the method.

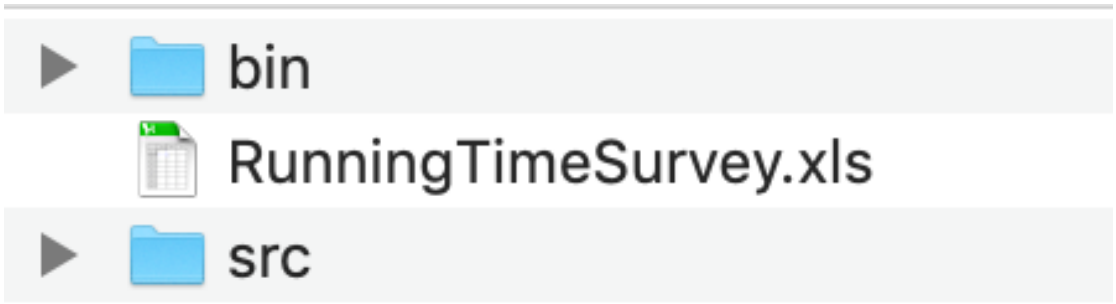
BubbleSortTest is completed as an example, the function name must be bubbleSortTime:

```
public static long bubbleSortTime(int n) {
    int[] list = data;
    long timeStart = System.currentTimeMillis();
    bubbleSort(n, list);
    long timeEnd = System.currentTimeMillis();
    long timeCost = timeEnd - timeStart;
```

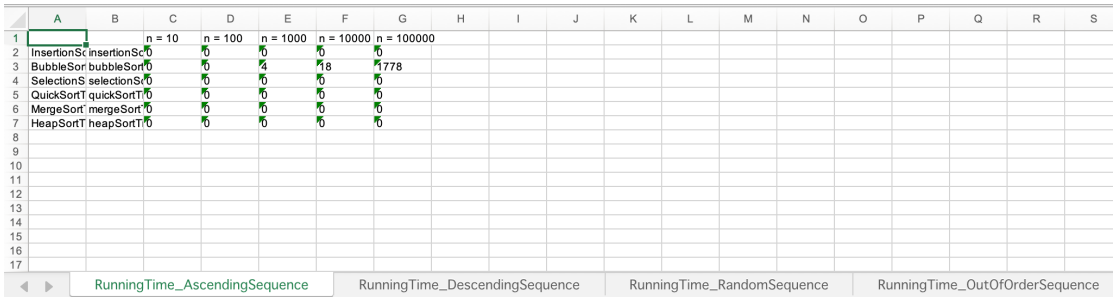
```
        return timeCost;
    }
```

```
    public static void bubbleSort(int n, int[] list) {
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (list[j] > list[j + 1]) {
                    // swap
                    int tmp = list[j + 1];
                    list[j + 1] = list[j];
                    list[j] = tmp;
                }
            }
        }
    }
```

When we run the sample, it generates a excel file named **RunningTimeSurvey.xls** in your project root directory:



Open the file, there are four sheets:



The RunningTime_AscendingSequence uses Ascending Sequence to test the algorithm, while RunningTime_DescendingSequence use Descending Sequence and so on.

For the example of Bubble Sort, we can see it takes least time when the data is in an ascending sequence as was mentioned in the lecture.

You should fill all other sort tests and provide a brief analysis of the result to identify the cases where each algorithm performs well and the ones in which it does not. You may refer to examples in the lectures as well as code online but you should ensure you understand your implementation and not blindly copy.

```
public static void insertionSort(int n, int[] list) {
    // TODO :add your code here
    // reference: http://math.hws.edu/eck/cs124/javanotes8/c7/s4.html 7.4.3

}

public static void selectionSort(int n, int[] list) {
    // TODO :add your code here
    // reference: http://math.hws.edu/eck/cs124/javanotes8/c7/s4.html 7.4.4

}

public static void quickSort(int n, int[] list) {
    // TODO :add your code here
    // Adam's ppt

}

public static void mergeSort(int n, int[] list) {
    // TODO :add your code here
    // https://introcs.cs.princeton.edu/java/42sort/ Mergesort

}

public static void heapSort(int n, int[] list) {
    // TODO :add your code here
    // optional

}
```

We will check the following items:

- Correctness of the program logic.
- Running time looks reasonable.
- The analysis makes sense.

2. Recursion

This section provides several examples of recursion for you to study in addition to the lectures. The first, factorial, requires a single parameter and has a single recursive call, the others demonstrate cases with multiple parameters for the recursive function and more than one recursive call in the methods.

2.1 Factorial function

Although we noted in the lecture that it is not necessarily ideal to use recursion when a loop could suffice, we include here several simple examples to introduce the idea of recursion. The "Hello, World" for recursion is the *factorial* function, which is defined for positive integers *n* by the equation:

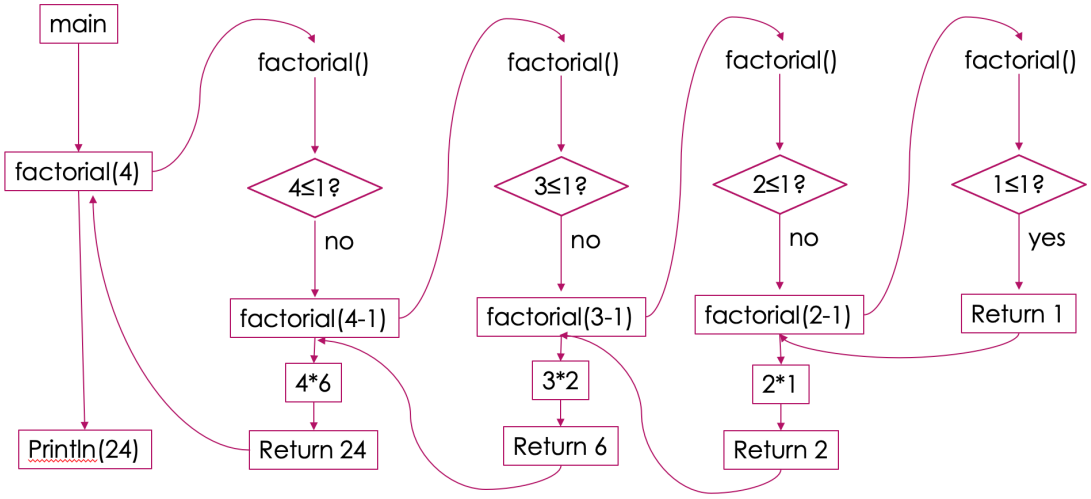
$n!=n\times(n-1)\times(n-2)\times\ldots\times2\times1$

```
public class Factorial {

   // return n!
   // precondition: n >= 0 and n <= 20
   public static long factorial(long n) {
      if     (n <  0) throw new RuntimeException("Underflow error in factorial");
      else if (n > 20) throw new RuntimeException("Overflow error in factorial");
      else if (n <= 1) return 1;
      else           return n * factorial(n-1);
   }

   public static void main(String[] args) {
      long n = Long.parseLong(args[0]);
      System.out.println(factorial(n));
   }
}
```

Trace this computation:



2.2 Towers of Hanoi puzzle

Recursion solution: First we move the top n−1 discs to an empty pole, then we move the largest disc to the other empty pole, then complete the job by moving the n−1 discs onto the largest disc. TowersOfHanoi.java is a direct implementation of this strategy.

```
public class TowerOfHanoi {
   static void towerOfHanoi(int n, char from_rod,
                            char to_rod, char aux_rod) {
      if (n == 1) {
         System.out.println("Move disk 1 from rod " + from_rod + " to rod " + to_rod);
         return;
      }
      towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
      System.out.println("Move disk " + n + " from rod " + from_rod + " to rod " + to_rod);
      towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
   }
```

```
    // Driver code
    public static void main(String args[]) {
        int n = 4; // Number of disks
        towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
    }
}
// This code is contributed by jyoti369
```

Input: 4

Output:

```
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
```

2.3 Euclid's algorithm

The greatest common divisor (gcd) of two positive integers is the largest integer that divides evenly into both of them. For example, the gcd(111, 37) = 37.

We can efficiently compute the gcd using the following property, which holds for positive integers p and q:

If p > q, the gcd of p and q is the same as the gcd of q and p % q.

The static method gcd() in Euclid.java is a compact recursive function whose reduction step is based on this property.

```
public class Euclid {

   // recursive implementation
   public static int gcd(int p, int q) {
      if (q == 0) return p;
      else return gcd(q, p % q);
   }
   // non-recursive implementation
   public static int gcd2(int p, int q) {
      while (q != 0) {
         int temp = q;
         q = p % q;
         p = temp;
      }
      return p;
   }

   public static void main(String[] args) {
      int p = Integer.parseInt(args[0]);
      int q = Integer.parseInt(args[1]);
```

```
        int d  = gcd(p, q);
//      int d2 = gcd2(p, q);
        System.out.println("gcd(" + p + ", " + q + ") = " + d);
//        System.out.println("gcd(" + p + ", " + q + ") = " + d2);
    }
}
```

Input: 111 37

Output: 37

Reference

http://math.hws.edu/eck/cs124/javanotes8/c7/s4.html
https://introcs.cs.princeton.edu/java/42sort/
https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/
https://www.youtube.com/watch?v=YstLjLCGmgg