Coursework 1

1. Submission for this coursework will be individual; detailed submission instructions will be given on Canvas.
2. There is a total of 40 marks to be gained by answering the questions.
3. Please do this seriously and write things clearly – this is meant to be an opportunity to get ready for the exam and get some feedback on your individual written work before then. The other coursework will be purely coding.

# Question 1: warm-up

For each of the following java function, give the asymptotic complexity in function of the input. Justify your answer. **[8 marks]**

1. static int[] fun1(int n)

{

if(n == 0)

return new int[2]; int[] bla = fun1(n-1); bla[0] += 1 + bla[1]; bla[1] \*= 3;

return bla;

}

1. static int fun2(int n)

{

if(n <= 15)

return 5;

return fun2(n-1) \* (1 - n/2);

}

1. static void fun3(int[][] arr)

{

final int n = arr.length;

if(n == 0 || arr[0].length != n) return;

for(int k = 0; k < n; ++k)

{

if(k%2 == 0)

for(int j = n; j > 0; --j) arr[k][j] = arr[j][k];

else

for(int j = 0; j < Math.sqrt(n); ++j) arr[j\*j][k] = arr[k-1][j];

}

}

1. static int fun4(int n)

{

if(n <= 5)

return 55; int r = 0;

for(int i = 0; i < n; ++i) r = (n + 98 \* r) % 23;

return (fun4(n/3) + fun4(n/3 - 1) \* r) % 55;

}

# Question 2: problem

Call a user a *star* on a social media if they follow no one, but everyone else follows them. We want to find an algorithm such that, assuming that we are given as input a *n × n* matrix with true in cell (*i, j*) if user *i* follows user *j* and false otherwise, returns a user who is a star, or *−*1 if there is not any (by convention, let us say that users can’t follow themselves so cells (*i, i*) can only contain false).

1. Give two examples of possible inputs with *n ≥* 3 users, one in which there is a star, and another where there is no star. **[2 marks]**
2. Is it ever possible to have two stars? Why? **[2 marks]**
3. Write an algorithm (in pseudo-code or in java, up to you) that solves the problem. The more efficient (asymptotically) your solution is, the higher the mark. **[5 marks]**
4. Run your algorithm step-by-step on the two examples you provided. **[2 marks]**
5. Analyze the asymptotic time complexity of your algorithm in function of the dimension of the input matrix. **[2 marks]**

# Question 3: dual quicksort

The goal of this question is to investigate an implementation of a variant of quicksort used in some standard libraries where two pivots are used instead of one and the problem is reduced to three sub-problems instead of two.

1. Discuss the asymptotic time complexity of the quicksort algorithm in the best, average and worst case. Consider two variants of the pivot selection: one where the pivot is taken to be a random index in the array, the other where it taken to be the median of the collection of values in the array thanks to a linear-time algorithm. **[4 marks]**

The Java implementation of our variant of the quicksort algorithm would be the following

static void dualQuickSortInner(int[] arr, min, max)

{

if(max - min < 17)

return insertSort(arr, min, max);

putPivotsAtTheFrontAndBack(arr, min, max); int pivotsPos[] = pivot2(arr, min, max); dualQuickSortInner(arr, min, pivotsPos[0]);

dualQuickSortInner(arr, pivotsPos[1], pivotsPos[2]); dualQuickSortInner(arr, pivotsPos[2], max);

}

static void dualQuickSort(int[] arr)

{

dualQuickSortInner(arr, 0, arr.length - 1);

}

To simplify the rest of the exercise, let us just assume that all the elements of arr are pairwise distincts. We will not specify the implementation of insertSort and putPivotsAtTheFrontAndBack, but will assume the following:

* putPivotsAtTheFrontAndBack(arr, min, max) swaps up to 4 elements of the array between min and max, in such a way that arr[min] *≤* arr[max-1]. The informal idea is that it puts two special pivots elements at the front and at the back of the array
* insertSort(arr,min,max) is a naive sorting algorithm that permutes the elements of arr between min and max so that they end up sorted
* pivot2(arr, min, max) should permute elements between min and max and re- turns an array pivots with two positions

min *≤* pivotsPos[0] *<* pivotPos[1] *≤* max

such that, assuming we originally had *x* = arr[min] and *y* = arr[max] with *x < y*

* + between min and pivotsPos[0] (inclusive), all elements of arr are strictly smaller than *x*
  + from pivotsPos[0] to pivotsPos[1], all elements of arr are equal to *x*
  + from pivotsPos[1] to pivotsPos[2]-1, all elements of arr are in ]*x, y*[
  + from pivotsPos[2] to pivotsPos[3]-1, all elements of arr are equal to *y*
  + all the remaining elements are strictly greater than *y*

Below is a picture synthesizing how the requirements above on the output between

min and max will look like:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *< x* |  | = *x* | *x* | *< − <* | *y* | = *y* | *y* | *<* |
|  | | | | | | | | | |

min max

1. Write a java implementation or pseudo-code for

static int[] pivot2(int[] arr, min, max);

as specified above. Your solution should work in-place (i.e., without creating a new array) and in time *O*(max *−* min). **[5 marks]**

1. Show that on input arrays that are already sorted, what is the time complexity of this sorting algorithm, assuming that putPivotsAtTheFrontAndBack does nothing? Justify briefly your answer by exhibiting the shape of a recurrence relation satisfied by this function. **[3 marks]**
2. Now assume that the function putPivotsAtTheFrontAndBack is smarter and manages to swap the*n* rd highest element between min and max at the position min and the 2*n* rd

3 3

highest element at the position max *−* 1, and that it runs in linear time, i.e., *O*(max*−*min).

Give the recursive equation satisfied by the time complexity then. Then deduce from the Master theorem the time complexity of dualQuickSort. **[5 marks]**

1. Is it clear whether there is a superior solution between quicksort and that one through the asymptotic analyses carried out in previous questions? If not, suggest some ways to determine which is the better candidate.

# [2 marks]