Assignment 3 [30 marks max]

General reminder: solutions without any justification / explanation on how you got the answer are considered insufficient. Providing the code written in some programming language (e.g., Python) is not expected and does not typically count as sufficient justification / explanation (especially, when you should demonstrate the behaviour of some algorithm).

Compulsory [5 marks max]

1. Find the reordering of array A = {0, 1, 2, 3, 4, 5, 6, 7, 8} such that the simple D&C method for calculating the median (i.e., the one without any sophisticated choice of pivots) required the maximum number of steps. Demonstrate the behaviour of the method for the reordered array on its input. **[5 marks]**

Elective [up to 25 marks]

2. Use the sophisticated D&C method for calculating the median (the method using the median of medians as the pivot) to calculate median of integers in A = { 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. Show individual steps to demonstrate you understood how the method works. **[5 marks]**

   Hint: be sure to return from recursion and continue.

3. Design an algorithm for calculation of square root of non-negative integer n with precision to k decimal places that will run in $\Theta(\log n)$ complexity. **[5 marks]**

   Hint: use divide & conquer.

4. Let A be an array of different integers. If there is a pair of indices i, j (i≠j) such that i<j and but A[i] > A[j], the pair (A[i], A[j]) is called an inversion.
   - **a.** Write all inversions in A = {2, 3, 8, 6, 1}. **[1 mark]**
   - **b.** Find a permutation of integers {1, 2, …n} such that the number of inversions is maximized. How many inversions this permutation has? **[1 mark]**
   - **c.** Design an algorithm that can count the number of inversions in the input array with $\Theta(n \cdot \log n)$ complexity. **[3 mark]**

     Hint: use divide & conquer + consider sorting.

5. BONUS 1 **[5 marks]**

   Given an unordered array A and an integer *k*, design an algorithm that can split the array into *k* consecutive chunks such that the largest sum of the values in chunks is minimal in the expected time better than $\Theta(n \cdot k)$. For example, if $k = 2$ we are searching for the index *i* where the array will be split into two parts (left and right) so that the maximum of the sum of left part and the sum of the right part is the minimal of all possible divisions.

   Hint: first design the algorithm for *k = 2* and only the try to generalise it. Exploit binary search to avoid testing every possible division

6. BONUS 2

Let *R* be a set of *N* rectangles in E2 (with sides parallel to axes x and y), whereas these rectangles may intersect. Let *S* be a set of *M* points (in E2) such that *M* >> *N*. For each point *p* of the set *S*, we want to find a subset *R*' $\subset$ *R* such that it contains all rectangles containing the point *p*. **[2 marks]**

a. Design an algorithm to solve this problem without any additional memory requirement effectively (i.e., all you need is the memory for the input sets *R*, *S* and the output subset *R*'). What is the time complexity of your algorithm? **[3 marks]**

b. Design an algorithm to solve this problem in time $\Theta(m \cdot \log n)$, i.e., the subset *R*' is found in time $\Theta(\log n)$. What is the time complexity of the pre-processing? What is the memory complexity of your algorithm?

Hint: use a balanced binary tree constructed in the pre-processing.