*Write your solutions in this sheet below the problem statement. Mark clearly your solutions by corresponding letters A, B, C, D and separate visually each solution from the other ones by empty space or by a line. Use the other side of the sheet or ask for an additional blank sheeet if necessary.*

*Each problem 1. – 4. is worth 0 – 4 points, each answer to a particular question A, B, C, D contributes at most 1 point to the total.*

*Example solutions/hints are listed below.*

1. Four algorithms process a square matrix M of size *n*×*n*. The indices of rows and columns of M start with 1. Each of the algorithms processes all matrix elements. Determine the asymptotic complexity of each algorithm.

A. Algorithm A processes each element of M in time proportional to log(*n*×*n*).

[Θ( *n*2 ∙ log(*n*)), yes, log(*n*) without the second power of *n*. ]

B. Algorithm B processes each element in the row *r* in time *k*∙*r*, where *k* is a positive constant.

[Totalling all row times ( *k* + 2 *k* +3 *k* + … + *n k* ) yields *k* ∙ *n*∙ (*n* +1)/2, and thus the complexity is Θ( *k* ∙ *n* 2) = Θ( *n* 2).]

C. Algorithm C processes each element at position [*r*][c] (*r* and *c* are row and column indices, respectively) in time *k*∙*r*∙*c*, where *k* is a positive constant.

[Evaluate the sum $\sum\_{r=1}^{n}\sum\_{c=1}^{n}krc=k\sum\_{r=1}^{n}(r∙\sum\_{c=1}^{n}c)=k\sum\_{r=1}^{n}(r∙\frac{n\left(n+1\right)}{2} )=…continue yourself$. It yields Θ(*n*4).]

D. Algorithm D processes each element in constant time. However, before processing the element, the algorithm checks the value of each element in the row in which the processed element is located. Each check takes constant time to complete.

[Checking any element takes time Θ( *n*) , there are is *n* 2 elements, the total time is thus is Θ( *n* 3).]

2. Insert sort processes an array A with an even number of values, array length is N. The array contains elements of only two values, 10 and 20. The first value in the array is 10 and the values in the array strictly alternate: 10, 20, 10, 20, 10, etc.

A. Determine how many comparisons between two element values are performed when the length of the array is 8:

 10 20 10 20 10 20 10 20

[0 + 1 + 2 + 1 + 3 + 1 + 4 + 1 = 13]

B. Determine how many comparisons between two element values are performed when the last element with value 10 is being inserted to its final position in array A. Solve the problem for the general value of N ≥ 4.

[The solution in A (0 + 1 + 2 + 1 + 3 + 1 + 4 + 1 = 13) suggests a pattern. The last 10 has to be compared to N/2–1 values 20 in front of it and finally to the last 10 in the sorted part of the array, and that yields N/2–1+1 = N/2 comparisons.]

C. Write a function *f*(N) which returns the total number of comparisons made during the sort of array A. Suppose N  ≥  4. Explain how did you derive *f*(N).

[The solution in A (0 + 1 + 2 + 1 + 3 + 1 + 4 + 1 = 13) suggests a pattern. Each of 20’s is associated with one comparison, the number of comparisons associated with 10’s is 0 + 2 + 3 + 4 + … + N/2. Thus, in total, there are

N/2 + (N/2)( N/2+1)/2 – 1 = (N2+6N–8)/8 comparison, f(N) = (N2+6N–8)/8. ]

D. Decide to which complexity class belongs *f*(N). The available classes are Θ(1), Θ(log N), Θ(N), Θ(N ∙ log N), Θ(N2).

[ Clearly, *f*(N) belongs to Θ(N 2), term 6N–8 and factor 1/8 can be neglected. ]

3. AVL tree T is in the picture. The four particular subtrees with roots in D, E, F, G are shown schematically and the value given in the subtrees represents the depth of the subtree. The depth is the number of edges on the path from the subtree root to the deepest node in the subtree.

A. T is depicted immediately after a new key (not depicted) was inserted into the tree. To complete correctly the insert operation, a rotation hast to be applied in T. Draw the shape of T after this rotation and determine the type of the rotation (L, R, LR or RL)

[ L rotation in node A.

Tree shape is coded at node N as follows: <L child of N>--<node N>--<R child of N>:

<D>--<B>--<E>, <B>--<A>--<F>, <A>--<C>--<G>. ].

B. Insert keys 10, 30, 20, 50, 40, in this order, into an originally empty AVL tree, using Insert operation. Draw the resulting AVL tree and determine the number and the type (L, R, LR, RL) of each rotation used in the process.

[ Shape: <10>--<20>-- <<30>--<40>--<50>> (R subtree of root 20 has 3 nodes). Two RL rotations, after insertion of 20 and 40. ]

C. Delete the key 10 from the AVL tree built in C. Which rotation, if any, was applied in the deletion process?

[ Tree root 20 is left with empty L subtree, only L rotation can help. 40 becomes new root with L child 20, parent of 30 is now 20. ]

D. Suppose that an AVL tree contains *n*2 keys and that another *n* keys are inserted, one by one, into the tree using the Insert operation. Use the asymptotic notation (O/Θ/Ω) to specify the maximum number of simple (L or R) rotations which are performed during the whole process. Explain your reasoning.

[ After the insertion, the tree is either balanced or it is not, and in that case Insert stops after the firs rotation, single or double. Thus, each insertion results in at most two simple rotations. This number is not related to the size of the tree, thus the maximum number of simple rotations in the whole process is 2∙*n* ∈ Θ(*n*), *n* ≥ 2. Why n ≥ 2? Explain it yourself.]

4. A binary heap is given in the picture.

A. Determine all possible integer values of element Z in the heap.

[10 ≤ Z ≤ 40, Z ∈$Z$]

B. Substitute Z by its smallest possible value (calculated in the answer to question A) .

Next, use the Insert operation to insert values 55, 45, 35 into the heap, in the given order. Draw the resulting heap.

[ Z = 10, the heap contents is then 10 10 30 40 50 35 70 80 90 99 55 60 45, heap top is on the left, draw the picture yourself.]

C. Apply twice operation ExtractMin to the heap obtained in B. Draw the resulting heap.

[ 30 40 35 45 50 35 70 80 90 99 55 ]

D. Suppose that a binary heap H contains exactly 2*n* leaves. Operation ExtractMin is applied *n* times on H. What is the asymptotic complexity of the whole process? Explain your reasoning.

[2*n* leaves means 2 *n* – 1 internal nodes, that gives 4*n* – 1 nodes in the heap in total.

After the last ExtractMin, there are 3 *n* – 1 nodes in the heap. Thus, for the complexity of each extraction, it hods:

 Θ(log(3*n* –1)) ≤ (ExtractMin complexity) ≤ Θ(log(4*n* –1)).

It should be obvious that Θ(log(A*n* –B)) = Θ(log(*n*)), for any positive constants A, B. Therefore, the inequality (1) becomes:

Θ(log(*n*)) ≤ (ExtractMin complexity) ≤ Θ(log(*n*)),

And therefore, ExtractMin complexity = Θ(log(*n*)).

The operation is performed *n* times, the whole process takes time Θ( *n* ∙ log(*n*)). ]