Data Structures - Assignment no. 6

Remarks:

- **Write both your name and your ID number very clearly on the top of the exercise. Write your exercises in pen, or in clearly visible pencil. Please write *very* clearly.**
- **Recall that 80% of the theoretical exercises must be submitted. The exercises can and must be worked on and submitted alone.**
- **Give correctness and complexity proofs for every algorithm you write.**
- **For every question where you are required to write pseudo-code, also explain your solution in words.**

1. In this question we discuss a model called the "extended comparison model", which is like the comparison model, except that you are allowed 5 types of questions: (i) "$a = b$?", (ii) "$a < b$?", (iii) "$a > b$?", (iv) "$a < b + 100$?", (v) "$a > b + 100$?". Prove a lower bound of $\Omega(n \log n)$ for sorting an array of size n in the extended comparison model.

2. (a) You are given two arrays, A and B, each of size n. Give an algorithm that returns an array C of size n, such that $C[i]$ is equal to the number of elements of A that are less or equal to $B[i]$. The algorithm should run in time $O(n \log n)$. Describe the algorithm and explain why the running time is $O(n \log n)$. You do not have to give pseudo-code.

   (b) Prove a lower bound of $\Omega(n \log n)$ for this problem in the comparison model.
   Hint: You can prove this lower bound directly. However, it is easier to give a reduction. To do this, you should: (i) Prove that if you can solve this problem in time $f(n)$ then you can sort an array of size n in time $O(f(n) + n)$; (ii) Deduce from this that if you can solve this problem in time $f(n)$ then $f(n) = \Omega(n \log n)$.

3. (a) You are given an array of size n, which contains $\lceil \log n \rceil$ distinct elements. Give an algorithm that sorts this array in time $O(n \log \log n)$. Describe the algorithm and explain why the running time is $O(n \log \log n)$. You do not have to give pseudo-code.

   (b) Prove a lower bound of $\Omega(n \log \log n)$ in the comparison model for this problem.

4. The QUICKSORT algorithm contains two recursive calls to itself. After the call to PARTITION, the left subarray is recursively sorted and then the right subarray is recursively sorted. The second recursive call in QUICKSORT is not really necessary; it can be avoided by using an iterative control structure. Consider the following version of quicksort.

```
QUICKSORT'($A, p, r$)
1   while $p < r$
2       do ▷ Partition and sort left subarray
3           $q \leftarrow$ PARTITION($A, p, r$)
4           QUICKSORT'($A, p, q$)
5           $p \leftarrow q + 1$
```

   (a) Argue that QUICKSORT'$(A, 1, length[A])$ correctly sorts the array A.

   (b) Compilers usually execute recursive procedures by using a stack that contains pertinent information, including the parameter values, for each recursive call. The information for the most recent call is at the top of the stack, and the information for the initial call is at the bottom. When a procedure is invoked, its information is pushed onto the stack; when it terminates, its information is popped. Since we assume that array parameters are actually represented by pointers, the information for each procedure call on the stack requires $O(1)$ stack space. The stack depth is the maximum amount of stack space used at any time during a computation.

   Describe a scenario in which the stack depth of QUICKSORT' is $\Theta(n)$ on an n-element input array.

   (c) Modify the code for QUICKSORT' so that the worst-case stack depth is $\Theta(\log n)$. Hint: apply the recursion on the side of A which contains less elements.