WORKSHEET, WEEK 08: SORTING

8.1 Problems

Problem 1:

(A) Find the $O(g(n))$ for the following function: $\log_2 n!$.

(B) Some algorithm receives n items as its input and then calls function $f(x_1, x_2, x_3, x_4)$ for any ordered quadruplet x_1, x_2, x_3, x_4 received in the input. Assume that $f(\ldots)$ runs in constant time. Find the time complexity of the whole algorithm.

(C) Some algorithm receives n items as its input and then calls a function f on all subsets of the received items having size $\lfloor n/4 \rfloor$. Assume that $f(\ldots)$ runs in constant time. Find the time complexity of the whole algorithm.

(D) What is the lower bound of comparisons needed to sort an array of 5 elements (assume they are all different)?

Problem 2: An array of 10 elements is used to initialize a minimum heap (as the first stage of the Heap sort algorithm):

$$\{5, 3, 7, 10, 1, 2, 9, 8, 6, 4\}$$

Assume that the minimum heap is initialized in the most efficient way (inserting elements level by level – starting from the bottom levels). All slots are filled in with the elements of the 10-element array in the order they arrive.

(A) How many levels will the heap tree have? (The root of the heap is considered L_0 – level zero. the last level is denoted by L_{k-1}. Just find the number k for this array.)

(B) Draw the intermediate states of the heap after each level is filled in. Represent the heap as a binary tree. (If some level L_k is only partially filled and contains less than 2^k nodes, please draw all the nodes as little circles, but leave the unused nodes empty.)

(C) What is the total count of comparisons ($a < b$) that is necessary to build the final minimum heap? (In this part you can assume the worst case time complexity – it is not necessarily achieved for the array given above.)

QuickSort Algorithm: This variant of Quicksort uses the leftmost element of the input area as a pivot. It is taken from the lecture slides. There are other Quicksort flavors (randomized or choosing a pivot differently).

```
QUICKSORT(A[ℓ ... r]) :
if l < r :
    i = ℓ          (i increases from the left and searches elements ≥ than pivot)
    j = r + 1      (j decreases from the right and searches elements ≤ than pivot.)
    v = A[ℓ]       (v is the pivot.)
    while i < j :
        i = i + 1
        while i < r and A[i] < v :
            i = i + 1
        j = j − 1
        while j > ℓ and A[j] > v :
            j = j − 1
        A[i] ↔ A[j]  (Undo the extra swap at the end)
    A[i] ↔ A[j]  (Undo the extra swap at the end)
    A[j] ↔ A[ℓ]  (Move pivot to its proper place)
    QUICKSORT(A[ℓ ... j − 1])
    QUICKSORT(A[j + 1 ... r])
```

Problem 3:

(A) Run this pseudocode for one invocation QUICKSORT$(A[0..11])$, where the table to sort is the following:

$$13, 0, 23, 1, 8, 9, 29, 16, 8, 24, 6, 11.$$

Draw the state of the array every time you swap two elements (i.e. execute $A[k_1] \leftrightarrow A[k_2]$ for any k_1, k_2).

(B) Continue with the first recursive call of QUICKSORT() (the original call QUICKSORT$(A[0..11])$ is assumed to be the 0^{th} call of this function). Draw the state of the array every time you swap two elements.

(C) Decide which is the second recursive call of QUICKSORT() and draw the state of the array every time you swap two elements. Show the end-result after this second recursive call at the very end.

Problem 4:

```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat
        swapped := false
        for i := 1 to n-1 inclusive do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
                /* swap them and remember something changed */
                swap(A[i-1], A[i])
                swapped := true
            end if
        end for
    until not swapped
end procedure
```

The image shows Bubble sort pseudocode for a 0-based array $A[0] \ldots A[n-1]$ of n elements.

(A) How many comparisons (`A[i-1] > A[i]`) in this algorithm are used to sort the given array. Show the state of the array after each `for` loop in the pseudocode is finished.

$$A[0] = 9,\ 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ A[9] = 8.$$

(B) How many comparisons (`A[i-1] > A[i]`) in this algorithm are used to sort the following array:

$$A[0] = 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ A[9] = 0.$$

Problem 5:

We have a 1-based array with 11 elements: $A[1], \ldots, A[11]$. We want to sort it efficiently. Consider the following Merge sort pseudocode:

```
MERGESORT(A, p, r):
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGESORT(A, p, q)
4      MERGESORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

Assume that initially you call this function as MERGESORT(A,1,11), where $p = 1$ and $r = 11$ are the left and the right endpoint of the array being sorted (it includes both ends).

(A) What is the total number of calls to MERGESORT for this array (this includes the initial call as well as the recursive calls on lines 3 and 4 of this pseudocode).

(B) How many comparisons are needed (in the worst case) to sort an array of 11 items by the MergeSort algorithm?

(C) Evaluate $\log_2 11!$ using Stirling's formula or a direct computation. What is the theoretical lower bound on the number of comparisons to sort 11 items?