

# Major Week Assignment 1

## IADS 2005, Episode 4<sup>1</sup>

(from the Alstrup–Rauhe Heritage)

This set consists of 10 exercises in total. The exercise can either be handed in individually or in groups of two persons on or before **March 9, 2005, 13.00**. Groups should hand in exercises that sum to at least 120 points. For hand-ins by single individuals, the requirement is 100 points. There should be a maximum of 6 exercises in the hand-in.

- The front page of the hand in must clearly state who the author(s) are.
- Solutions should be described as briefly as possible.
- The algorithms that you design need not be described in pseudocode.
- If an exercise asks for the “construction of a heap that supports the operations”, it should be described how to construct a data structure that supports the operations. It does not matter whether it is similar to the heap in CLRS.
- In some exercises we provide examples of the operations that should be supported. These only serve the purpose of being examples. For instance, if you are asked to support the operation  $+$ , and the example is  $2 + 3 = 5$ , then a solution which is only able to add 2 and 3 is not a good solution.

### Exercises

1. **(40 points)** Construct a heap (or a heap-like data structure) that supports  $\text{HEAP-INSERT}(A, \textit{key})$  and  $\text{HEAP-EXTRACT-MAX}(A)$  in  $O(\lg n)$  time and  $\text{HEAP-UPDATE-ALL}(A, k)$  in  $O(1)$  time.  $\text{HEAP-UPDATE-ALL}(A, k)$  adds  $k$  to all values. As an example, assume the heap is empty and the following operations are executed:  
     $\text{HEAP-INSERT}(A, 10)$ ,  
     $\text{HEAP-UPDATE-ALL}(A, 100)$ ,  
     $\text{HEAP-INSERT}(A, 55)$ ,  
     $\text{HEAP-UPDATE-ALL}(A, 1000)$ ,  
     $\text{HEAP-INSERT}(A, 30)$ ,  
     $\text{HEAP-EXTRACT-MAX}(A)$ ,  
     $\text{HEAP-EXTRACT-MAX}(A)$  and  
     $\text{HEAP-EXTRACT-MAX}(A)$ .  
Then the values returned should be 1110, 1055 and 30 (in that order).
2. **(38 points)** Give a correctness proof for the algorithm  $\text{COUNTING-SORT}$  described on page 168 in CLRS.

---

<sup>1</sup>Revised version, created by correcting errors pointed out by Esben.

3. **(24 points)** Chapter 10.1 describes the data structure called *queue*. A queue supports two operations, one which inserts an element into the queue, and one which extracts the element that has been in the queue for the longest time. Both operations work in  $O(1)$  time. Assume we are given a queue that contains  $n$  elements where each element is a list of integers. Initially each list only consist of a single integer, so that the lists are initially sorted. Note that given two lists with a total of  $\ell$  elements, we can merge them into a single sorted list in  $O(\ell)$  time. Given a queue of lists, we iteratedly execute the following code until the queue only contains a single sorted list:
- Extract two lists form the queue;
  - Merge these two lists in to the new list  $L$ ;
  - Insert the list  $L$  into the queue.

*Example:*  $((1),(7),(8),(5))$ . We have 4 lists in the queue. The 4 lists contain the numbers 1, 7, 8 and 5 respectively. We execute the program:

- (1) and (7) are extracted from the queue which then is  $((8),(5))$ ;
- (1) and (7) are merged to (1,7);
- (1,7) is inserted into the queue which is then  $((8),(5),(1,7))$ ;
- (8) and (5) are extracted and the queue is then  $((1,7))$ ;
- (8) and (5) are merged into (5,8);
- (5,8) is inserted into the queue which now is  $((1,7),(5,8))$ ;
- (1,7) and (5,8) are extracted from the queue which then is the empty queue  $()$ ;
- (1,7) and (5,8) are merged into (1,5,7,8);
- (1,5,7,8) is inserted into the queue which is then  $((1,5,7,8))$ .

The queue now only consists of a single sorted list.

Analyse the worst-case complexity of this sorting algorithm.

4. **(21 points)** Let  $\text{HALF}(A, x)$  be a procedure which as arguments takes  $k$  integers stored in an array  $A$ , and among these search for a specific integer  $x$  (which is assumed to be present among the integers in  $A$ ). If  $k = 1$  then  $\text{HALF}$  simply returns the single integer that is given as argument. For  $k > 1$ ,  $\text{HALF}$  returns (an array containing) half of the integers (i.e.  $\lceil k/2 \rceil$  integers), where  $x$  is guaranteed to be among these. If we wish to find  $x$  among the  $2k$  integers, we can simply repeat the call to  $\text{HALF}$  with the integers that  $\text{HALF}$  reduces the solution to, until the integer  $x$  is the only one left. Assume a call of  $\text{HALF}$  for  $n$  integers takes time  $O(n)$ . How much time does it take to find  $x$  starting with  $n$  integers? That is, find the sum of time costs of all the iterated calls of  $\text{HALF}$ .
5. **(24 points)** We call  $\text{BUILD-MAX-HEAP}$  with  $n$  elements (page 133) and then execute  $n/\lg n$  many  $\text{HEAP-EXTRACT-MAX}$  (page 139) operations. What is the total time cost of this?
6. **(24 points)** One of the disadvantages of the heaps we have seen until now is that we need to know the maximal number of elements in the heap at any time in advance. In some situations this is not satisfactory. We will address this problem in the following way. Initially we start with a heap that has a limit of only one

element. Now suppose the user would like to insert a new element but, alas, there is no room for this. Suppose the heap at this stage has  $k$  elements. Here is what we do: We build a new heap, with a limit of  $2k$  elements and copy the  $k$  elements from the old heap into the first  $k$  entries in the new heap so that the heap property is preserved. Let us call this heap operation  $\text{CONSTRUCT-HEAP}(j)$ , and let complexity of this operation be  $O(j)$ . The argument is the size limit of the new heap to be constructed. A sequence of heap operations would look like this:

```

CONSTRUCT-HEAP(1); ...;
CONSTRUCT-HEAP(2); ...;
CONSTRUCT-HEAP(4); ...;
CONSTRUCT-HEAP(8); ...

```

Assume the last  $\text{CONSTRUCT-HEAP}$  call is  $\text{CONSTRUCT-HEAP}(2^m)$ . What is the total time cost of all the  $\text{CONSTRUCT-HEAP}$  operations?

In the exercises below we will use the following terminology: The element of order  $i$  from a set of  $n$  elements is the  $i$ th smallest element. For instance, the minimum of a set of  $n$  elements has order 1, while the maximum element is the element of order  $n$ . The *median* of a set of  $n$  elements, where  $n$  is odd, is the element of order  $(n + 1)/2$ . For  $n$  even, the median is the element of order  $n/2$ .

7. **(49 points)** Assume we have an algorithm that finds the median of distinct elements in an unsorted array of  $n$  integers in time  $O(n)$ .

Design an algorithm that for an array with  $n$  distinct elements returns the elements from order  $i$  to order  $\min\{i + \lfloor \sqrt{n} \rfloor - 1, n\}$  in sorted order. The time cost should be  $O(n)$ .

Suppose that instead of returning  $\lfloor \sqrt{n} \rfloor$  elements we want to return  $k > \sqrt{n}$  elements. How large can we set  $k$  expressed in terms of  $n$  while keeping the total time cost  $O(n)$ ?

8. **(33 points)** In this exercise, we assume that we have access to an algorithm that finds the median in an unsorted array of  $n$  distinct elements in time  $O(n)$ .

A *median-queue* is a type of a heap with an  $\text{EXTRACT}$  operation that extracts the median instead of the maximal element. Formally, the median-queue supports the operations:

- $\text{INIT}(X, S)$ : Initialises the queue  $S$  to contain the elements from the unsorted input array  $X$  of distinct elements.
- $\text{INSERT}(S, x)$ : Inserts  $x$  into the queue  $s$ .
- $\text{EXTRACT}(S)$ : Extracts and returns the median from  $S$ .

Construct a median-queue. The time cost must be  $O(\lg n)$  for  $\text{INSERT}$  and  $\text{EXTRACT}$ , while  $\text{INIT}$  must work in linear time.

*Hint:* Use 2 (two) heaps and exploit the fact that  $\text{BUILD-HEAP}$  has time cost  $O(n)$ .

9. **(24 points)** CLRS Exercise 2.3–7, page 37. Assume that *set* means “unsorted array”.
10. **(30 points)** CLRS Problem 6–1, page 142.

*Hint:* in 6–1b, (3.18) on page 55 in CLRS might be useful for proving the  $\Omega$ -part of the  $\Theta$