

## CSE 373 Spring 2009: Homework 3 Solutions

### 2.13

(a)  $B_3=1, B_5=2, B_7=5, B_n=0$  if  $n$  is even

(b) Consider an FBT (full binary tree) of size  $n$ . Removing the root gives us two FBTs of size  $(n-i)$  and  $(i-1)$  where  $i$  is some number between 1 and  $n-1$ . Thus,

$B_n = 0$ , if  $n$  is even.

Else,

$B_n = \sum (B_{n-i} B_{i-1})$  ; The summation is over  $i$ , where  $1 \leq i \leq n-1$ .

(c) Let  $c$  and  $m$  be some constants, such that  $m > 2^{c+1}$

Let,  $B_k > 2^{c.k}$  for all  $k$  less than  $n$  and greater than  $m$ .

Lets show the same for  $B_n$  for all  $n > m$ .

$$\begin{aligned} B_n &= \sum B_{(n-i)} B_{(i-1)} \\ &> \sum 2^{c.(n-i)} 2^{c.(i-1)} \\ &> \sum 2^{c(n-1)} \\ &> \sum 2^{c.n} / 2^c \\ &= n/2 * 2^{c.n} / 2^c \\ &> 2^{c.n} \end{aligned}$$

Note that there are  $n/2$  terms in the summation. The last inequality is true because we choose  $m > 2^{c+1}$  and  $n > m$ .

### 2.14

Sort all the numbers of the array in  $O(n \log n)$ . Scan the sorted array, copy only one of duplicate numbers to a different array  $\Rightarrow O(n)$ . Overall  $O(n \log n)$ .

### 2.15

Find an element in  $S$  with value to be  $v$ . Then exchange this element with the end of  $S$ . Use this element as pivot, if less than  $v$ , put it in the front of  $S$ , if large, put it in the back. Finally, exchange the first large element with the end element.

### 2.17

Use a binary tree. Check whether  $A[i] = i$  at a node. Check the right subtree if  $A[i] < i$ , or the left subtree if  $A[i] > i$ . This will work only because the array has distinct integers.

2.22 You can check whether an element  $A[i]$  is the  $k$ -th smallest in constant time. Suppose that the  $k$ -th smallest is  $A[i]$ . Since the array is sorted, it is greater than exactly  $i - 1$  values in array  $A$ . Then if it is the  $k$ -th smallest, it is also greater than exactly  $j = k - (i - 1)$  elements in  $B$ . It requires constant time to check if  $B[j] \leq A[i] \leq B[j + 1]$ . If  $A[i]$  is not the  $k$ -th smallest, then depending on whether  $A[i]$  is greater or less than  $B[j]$  and  $B[j + 1]$ , you know that  $A[i]$  is either greater than or less than the  $k$ -th smallest. Thus you can binary search for  $A[i]$  in  $O(\log mn)$  worst-case time.

2.23(a)

The idea is similar to the merge sort.

If  $a$  is a majority element in the array  $A[1 \dots, n]$ , then it must also be a majority element in either  $A[1 \dots, \text{ceil}(n/2)]$  or  $A[\text{ceil}(n/2 + 1) \dots, n]$ .

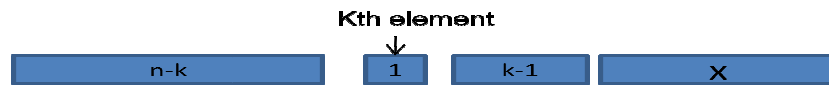
Thus, split array  $A$  into two sub-arrays, find majority (recursively) in each of the sub-arrays, and finally check if any of the two majorities in the subarray is actually the majority in the full array. The last check-step can be done in linear-time. Thus, the overall time complexity is  $O(n \log n)$ .

Q. Suppose you have a subroutine that can find the median of a set of  $n$  items (i.e., the  $(n/2)$  smallest) in  $O(n)$  time. Give an algorithm to find the  $k$ th biggest element (for arbitrary  $k$ ) in  $O(n)$  time.

Solution:

First, with one linear scan in time  $O(n)$ , we find the minimum and maximum of the array.

Now, if  $k \leq n/2$ , we add  $n - 2k + 2$  copies of  $\text{max} + 1$  to the array.



Otherwise, we add  $2k - n - 2$  copies of  $\text{min} - 1$  to the array.



Thus, what was previously the  $k$ th biggest element is the median ( $n'/2$  th smallest element) of the new array. Thus, after this modification (which took linear time), we can simply run the linear-time subroutine for finding the median, and it will return the right element.