

## CS 430 - Homework 2, solution sketches

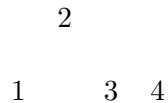
**Problem 1** Let  $T$  be a heap storing  $n$  keys. Assume for this problem that the heap is stored in an array  $A$  starting from 1 - that is,  $A[1]$  is the biggest key in the heap. Give an efficient algorithm for reporting all the keys in  $T$  that are greater than or equal to a given query key  $x$  (which is not necessarily in  $T$ ). Note that the keys do not need to be reported (printed) in sorted order. Analyse the running time. An  $O(k)$  algorithm is needed for full grade, where  $k$  is the number of elements printed. That is, the algorithm should do a constant number of elementary operations per printed element.

### Solution Sketch:

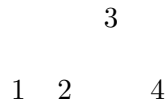
```
HEAP-GREATER-EQUAL(A,i,x)
1  if A[i] ≥ x
2    print (A[i])
3    if 2i ≤ n
4      HEAP-GREATER-EQUAL(A,2i,x)
5    if 2i + 1 ≤ n
6      HEAP-GREATER-EQUAL(A,2i+1,x)
```

**Problem 2** Professor Amongus claims that a (2,3) tree storing a set of items will always have the same structure, regardless of the order in which the items are inserted. Show that Professor Amongus is wrong, by giving two orders of the same set of items giving distinct trees.

Consider the set of keys (1,2,3,4). If the keys are inserted into a 2-3 tree in this order, it generates the tree

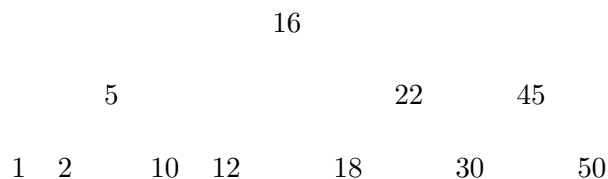


However, if the keys are inserted in the order (4, 3, 2, 1), it generates the following 2-3 tree.



**Problem 3** Consider the following sequence of keys: (5,16,22,45,2,10,18,30,50,12,1) Consider the insertion of items with this set of keys, in the order given, into an initially empty (2,3) tree  $T$ . Draw  $T$  after each insertion.

### Final tree:



**Problem 4** Suppose we are given two sorted arrays  $S$  and  $T$ , each with  $n$  items. Describe an  $O(\log n)$ -time algorithm for finding the  $k$ th smallest key in the union of the keys from  $S$  and  $T$  (assuming no duplicates). Present pseudocode, argue correctness and analyze the running time.

**Solution Sketch:**

```

FIND-K-SMALLEST(S,T,n,k)
1  if k == 1
2    then return Min(S[1],T[1])
3  if k == 2n
4    then return Max(S[n],T[n])
5  left ← 1
6  right ← n
7  do while left ≤ right
8    k1 = ⌊(left + right)/2⌋
9    k2 = k - k1 /*Kth smallest is in S[1..k1] or T[1..k2]*/
10   if (k2 ≥ n)
11     then left ← k1+1 /* k ≥ k1+n, need to increase k1 */
12     continue
13   if (k2 ≤ 0)
14     then right ← k1-1 /* k ≤ k1, need to decrease k1 */
15     continue
16   if (S[k1] < T[k2+1])
17     then left ← k1
18   if (S[k1+1] > T[k2])
19     then right ← k1
20   if (S[k1] < T[k2+1] AND S[k1+1] > T[k2])
21     then return (Max(S[k1],T[k2]))
22 end while
23 return (Max(S[k1],T[k2]))

```

Binary search is used in array  $S$  to locate the  $k$ th smallest item. This is guaranteed by three conditions:

- 1)  $K = k_1 + k_2$
- 2)  $S[k_1] < T[k_2+1]$  and  $S[k_1+1] > T[k_2]$

Because binary search is used, the running time is  $O(\log n)$ .

**Problem 5** Suppose we are given two  $n$ -element sorted sequences  $A$  and  $B$  that should not be viewed as sets (that is,  $A$  and  $B$  may contain duplicate entries). Describe an  $O(n)$ -time method for computing a sequence representing the set  $A \cup B$  (with no duplicates). Present pseudocode and analyze the running time.

**Solution Sketch:**

```
UNION(A,B)
1  k ← 0
2  i ← 0
3  j ← 0
4  while i < n and j < n
5      if A[i] < B[j]
6          then x ← A[i]
7              i ← i + 1
8      else if A[i] == B[j]
9          then x ← A[i]
10             i ← i + 1
11             j ← j + 1
12     else if A[i] > B[j]
13         then x ← B[j]
14             j ← j + 1
15     if k == 0 or x > result[k-1]
16         then result[k] ← x
17             k ← k+1
18     while i < n
19         if A[i] > result[k-1]
20             then result[k] ← A[i]
21                 k ← k + 1
22                 i ← i + 1
23     while j < n
24         if B[j] > result[k-1]
25             then result[k] ← B[j]
26                 k ← k + 1
27                 j ← j + 1
```