

ECE 537 - Foundations of Computing  
Prof. Sen  
**Homework #10**  
**Solutions**

1. Draw a diagram which illustrates the arrangement of all the pointers and associated data structures needed in:

- (a) A binomial heap implementation.
- (b) A Fibonacci heap implementation.

**Solution:** See Figures 19.3 (b) and 20.1 (b) in the Cormen et al. text.

2. Exhibit a sequence of MELDABLE PRIORITY QUEUE ADT operations that lead to the construction of an  $n$ -vertex Fibonacci heap containing a single tree having a height of exactly  $n - 1$ .

**Solution:** Try the following at the site

<http://www.cse.yorku.ca/~aaw/Jason/FibonacciHeapAnimation.html>:

- First insert nine items with the following keys in the order given:  $Insert(H,9)$ ,  $Insert(H,8)$ ,  $Insert(H,7)$ ,  $Insert(H,6)$ ,  $Insert(H,5)$ ,  $Insert(H,4)$ ,  $Insert(H,3)$ ,  $Insert(H,2)$ ,  $Insert(H,1)$ . This leads to a Fibonacci heap that consists of nine single-vertex trees.
- Now perform two  $Extract-Min(H)$  operations. This will leave you with a Fibonacci heap that contains three trees: one with one four vertices, one with two vertices, and one with one vertex.
- Next perform a  $Delete(H,7)$  operation. This deletes from the four-vertex tree the leaf that is also a child of the root, leaving two trees: one with five vertices, and one with one vertex that is also the minimum key.
- Next perform another  $Extract-Min(H)$ . This results in a Fibonacci heap containing a single tree with five vertices.
- Finally, perform a  $Delete(H,5)$ , deleting the child of the root that is a leaf. This results in a four-vertex tree that has a height of three.

3. Prove that the number of children at a depth of  $i$  in binomial tree  $B_k$ ,  $0 \leq i \leq k$ , is  $\binom{k}{i}$ . The proof is by induction on  $k$ . The base case is trivial to verify. Now assume the statement holds for  $B_{k-1}$ , and let  $D(k, i)$  denote the number of children at a depth of  $i$  in binomial tree  $B_k$ . Recall that  $B_k$  is composed of two trees from  $B_{k-1}$ , where one of the trees from  $B_{k-1}$  is made a subtree of the root vertex of the other. Thus, a vertex at a depth of  $i$  in  $B_{k-1}$  appears in  $B_k$  once at a depth of 5 of

$i$ , and once at a depth of  $i + 1$ . Therefore, using the induction hypothesis we can write

$$\begin{aligned}
 D(k, i) &= D(k - 1, i) + D(k - 1, i - 1) \\
 &= \binom{k - 1}{i} + \binom{k - 1}{i - 1} \\
 &= \frac{(k - 1)(k - 2) \cdots (k - i)}{i!} + \frac{(k - 1)(k - 2) \cdots (k - i + 1)}{(i - 1)!} \\
 &= \frac{(k - 1)(k - 2) \cdots (k - i)}{i!} + \frac{i(k - 1)(k - 2) \cdots (k - i + 1)}{i!} \\
 &= \frac{k[(k - 1)(k - 2) \cdots (k - i + 1)] - i[(k - 1)(k - 2) \cdots (k - i + 1)]}{i!} \\
 &\quad + \frac{i(k - 1)(k - 2) \cdots (k - i + 1)}{i!} \\
 &= \frac{k(k - 1)(k - 2) \cdots (k - i + 1)}{i!} \\
 &= \binom{k}{i}
 \end{aligned}$$

4. Read Chapter 21 in the Cormen et al. text, and then answer problem 21-1.

**Solution:**

- For the input sequence:  
4,8,E,3,E,9,2,6,E,E,E,1,7,E,5,  
the values in the *extracted* array would be: 4,3,2,6,8,1.
- We want to show that the array *extracted* returned by OFF-LINE-MINIMUM is correct, meaning that for  $i = 1, 2, \dots, m$ ,  $extracted[j]$  is the key returned by the  $j$ -th *Extract-Min* operation.

We start with  $n$  *Insert* operations and  $m$  *Extract-Min* operations. The smallest of all the elements will be extracted in the first *Extract-Min* after its insertion. So we find  $j$  such that the minimum element is in  $K_j$ , and put the minimum element in  $extracted[j]$ , which corresponds to the *Extract-Min* after the minimum element insertion.

Now we reduce to a similar problem with  $n - 1$  *Insert* operations and  $m - 1$  *Extract-Min* operations in the following way: the *Insert* operations are the same but without the insertion of the smallest element that was extracted, and the *Extract-Min* operations are the same but without the extraction that extracted the smallest element. The idea involves uniting  $I_j$  and  $I_{j+1}$ , removing the extraction between them and also removing the insertion of the minimum element from  $I_j \cup I_{j+1}$ . Uniting  $I_j$  and  $I_{j+1}$  is accomplished on line 6. We need to determine which set is  $K_l$ , rather than just using  $K_{j+1}$  unconditionally, because  $K_{j+1}$  may have been destroyed when it was united into a higher-indexed set by a previous execution of line 6.

Because extractions are processed in increasing order of the minimum value found, the remaining iterations of the for loop correspond to solving the reduced problem.

There are two things to observe here. First, if the smallest remaining element had been inserted after the last *Extract-Min* (i.e.,  $j=m+1$ ), then no changes occur, because the element is not extracted. Second, there may be smaller elements within the  $K_j$  sets than the one we are currently looking for. These elements do not affect the result, because they correspond to elements that were already extracted, and therefore they have no further effect on the execution of the algorithm.

- To implement this algorithm, start by placing each element in a disjoint-set forest. Each root has a pointer to its  $K_i$  set, and each  $K_i$  set has a pointer to the root of the tree representing it. All the valid sets  $K_i$  are stored in a linked list. Before *Off-Line-Minimum*, there is an initialization that builds the initial set  $K_i$  according to the  $I_i$  sequences:
  - Line 2 “determine  $j$  such that  $i \in K_j$ ”) becomes  $j \leftarrow \text{Find-Set}(i)$ .
  - Line 5 “let  $l$  be the smallest value greater than  $j$  for which set  $K_l$  exists” becomes  $K_l \leftarrow \text{next}[K_j]$ .
  - Line 6 “ $K_l \leftarrow K_j \cup K_l$ , destroying  $K_j$ ” becomes  $l \leftarrow \text{Link}(j, l)$  and remove  $K_j$  from the linked list.

In order to analyze the running time, note that there are  $n$  element and the following disjoint-set operations:

- $n$  *Make-Set* operations,
- at most  $n - 1$  *Union* operations before starting,
- $n$  *Find-Set* operations,
- at most  $n$  *Link* operations.

Thus the overall number of operations,  $m$ , is  $O(n)$ , and the total running time is therefore  $O(m\alpha(n)) = O(n\alpha(n))$ .

(Note: the “tight bound” wording in the phrasing in the problem statement does not mean an asymptotically tight bound, rather, they are asking for a bound that is not too “loose”.)