Homework #3

(Due: May 04)

GROUP NUMBER: 18

Group Members		
Name	SBU ID	% Contribution
Manish Reddy Vadala	114190006	33
Ranjith Reddy Bommidi	114241300	33
Venkata Ravi Teja Takkella	113219890	33

COLLABORATING GROUPS

Group Number	Specifics (e.g., specific group member? specific task/subtask?)
NA	NA

EXTERNAL RESOURCES USED

	Specifics (e.g., cite papers, webpages, etc. and mention why each was used)
1.	NA
2.	
3.	
4.	
5.	

Task 1 (a)

We merge two min heap ordered trees $\hat{T}_k$ to produce one $\hat{T}_{k+1}$. So, inorder to merge two $\hat{T}_k$ trees we first compare the root nodes of the two trees and the tree with minimum node will become root node of $\hat{T}_{k+1}$ tree, here cost of comparing is 1 operation. The larger root of $\hat{T}_k$ right child will be joined to immediate child of smaller root node's $\hat{T}_k$ tree. The child to the tree with minimum node is cut and now it points to the root of larger $\hat{T}_k$ node tree. The new tree produced after linking two $\hat{T}_k$ trees looks similar to $\hat{T}_{k+1}$ tree, but it need not follow heap property as we see in fig1 node 8 does not follow heap property so we try to heapify the right tree of the node 8, the left tree of node 8 already follows heap property originally. The cost of heapifying is k in the worst case for merging two $\hat{T}_k$ trees.

Hence cost of merging two $\hat{T}_k$ trees = k + 1

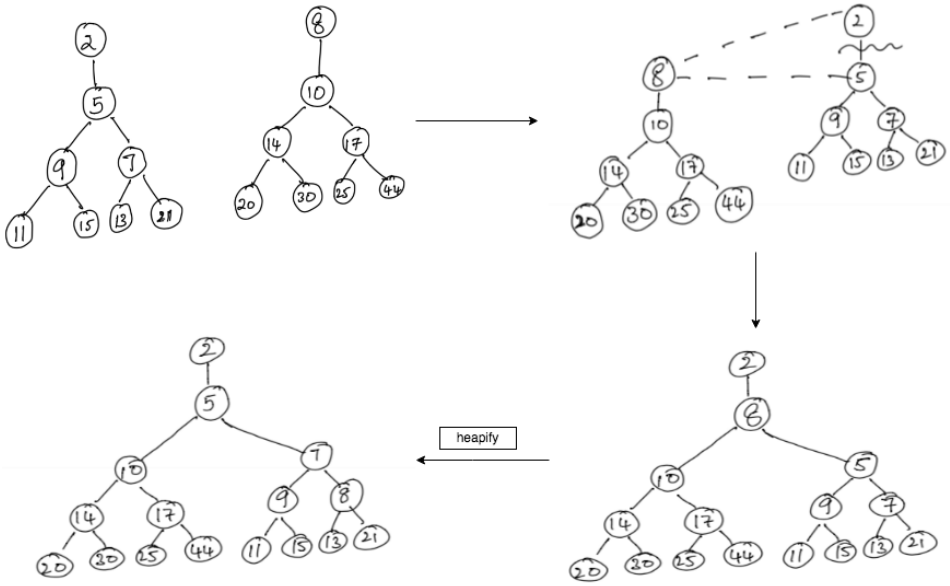

Figure 1: Figure shows an efficient way to merge two min heap ordered $\hat{T}_k$ trees to produce a new $\hat{T}_{k+1}$ tree

Task 1 (b)

Here we are asked to find the amortized costs of MAKE-HEAP, MINIMUM, INSERT, EXTRACT-MIN and UNION under eager union. Also mentioned that the data structure will never contain more than N items during it's entire lifetime.

Let us define the potential function over here as

$$\Phi(D_i) = c \times \sum_{\hat{T}_k \in D_i} (k+2)$$

Here D_i is the state of our Data Structure after i^{th} operation and k explains the height of the tree $\hat{T}_k$ which belongs to the Heap.

Amortized Cost($\hat{c}_i$) for each operation is sum of actual cost(c_i) and potential difference(Δ_i)

$$\hat{c}_i = c_i + \Delta_i$$

MAKE-HEAP

Here we are creating a new binary heap with just 1 index and contains only $\hat{T}_0$. For creating a binary heap

$$actual\ cost,\ c_i = 1$$

$$potential\ change,\ \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = 2c$$

(as #trees increases by 1)

$$amortized\ cost,\ \hat{c}_i = c_i + \Delta_i = 1 + 2c = \Theta(1)$$

MINIMUM

Since we store the Min Pointer, we can get this with the cost of one. So for getting Minimum Value of the Heap

$$actual\ cost,\ c_i = 1$$

$$potential\ change,\ \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = 0$$

(as #trees doesn't change)

$$amortized\ cost,\ \hat{c}_i = c_i + \Delta_i = 1 + 0 = \Theta(1)$$

UNION(Eager-Union)

Union of Perfect Binomial Heaps works exactly the same way as Union of Binomial Heaps which is Binary addition. But in Binomial Heaps linking two binomial trees of order k can be achieved in $\Theta(1)$ time since we just need to link the two root nodes of the trees and we get a new binomial tree of order k+1. But in our case we might need to Heapify again after linking two extended perfect binomial trees to get a new perfect extended binomial tree of order k+1. This can be achieved with cost of k+1 from **1)a**, where k is the height of the tree.

So, in Union(H_1, H_2). Let n_1 , n_2 be the number of nodes in Heaps respectively and $n_1 + n_2 \leq N$. Lets assume that we have performed k scans in both the heaps and among which we have linked l trees. For linking these l trees the cost would be $\sum_{\hat{T}_i \in l}(i+1)$ and for linking two $\hat{T}_s$ trees the potential change would be $\Delta_i = (s+1+2-2(s+2)) = -(s+1)$. Therefore the potential change for linking l trees would be $\sum_{\hat{T}_i \in l} -(i+1)$. Finally we update the min pointer by comparing the min pointers of two Heaps H_1, H_2 which costs 1.

$$actual\ cost,\ c_i = k + \sum_{\hat{T}_i \in l}(i+1) + 1$$

$$potential\ change,\ \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = c\sum_{\hat{T}_i \in l} -(i+1)$$

$$amortized\ cost,\ \hat{c}_i = c_i + \Delta_i = k + 1 + \sum_{\hat{T}_i \in l}(i+1) + c\sum_{\hat{T}_i \in l} -(i+1)$$

(for c = 1)

$$amortized\ cost,\ \hat{c}_i = k + 1 = O(k)$$

$$And\ this\ is\ upper\ bounded\ by\ O(\log N)$$

(we know k $\leq \log N$, k is the total number of cells)

INSERT

In the Operation Insert(H, x) to insert x into the Heap, we first create a MAKE-HEAP with x. Which takes $\Theta(1)$ time and then perform Union(H,H') where H' is the new Heap with one node(x).

Let H be our perfect binomial heap which contains k trees. Let l be the number of link operations we have performed while inserting H'. From **1a)**, we have that the cost of merging two $\hat{T}_i$ tress take $i+1$ time. Then the cost of linking l trees starting with $\hat{T}_0$, would be $\sum_{i=0}^{l-1}(i+1) = l(l+1)/2$. Finally, we need to update the min pointer by checking whether the new inserted node is smaller than the present min node. This comparison and updating the min pointer would take $\Theta(1)$ time. And let's assume that there are k trees in the data structure at this point.

$$actual\ cost,\ c_i = Cost\ for\ MAKEHEAP\ + Cost\ for\ Merging + Cost\ for\ Updating\ Min$$

$$actual\ cost,\ c_i = 2 + \frac{l(l+1)}{2} + 1 = 3 + l^2/2 + l/2$$

(performing l link operations takes i+1 time for linking each $\hat{T}_i$ tree with other $\hat{T}_i$)

$$potential\ change,\ \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = c(\sum_{i=0}^{k-1}(i+2) - \sum_{i=0}^{l-1}(i+2) - \sum_{i=0}^{k-1}(i+2)) = c(-\sum_{i=0}^{l-1}(i+2))$$

$$= -3cl/2 - cl^2/2$$

$$amortized\ cost,\ \hat{c}_i = c_i + \Delta_i = 3 + l^2/2 + l/2 - 3cl/2 - cl^2/2 = \Theta(1)$$

(for c = 1)

EXTRACT-MIN

We can Perform Extract-Min operation in 5 Steps, as follows:

Step 1: Locate the Min Pointer.

Step 2: Sever the Min Pointer from the Heap.

Step 3: Create a new Heap with the children of the severed Min Pointer Node.

Step 4: Do the Union Operation with the Original Heap and the new Heap.

Step 5: Update the Min Pointer.

The cost for step 1 is 1, since we already have a Min pointer which points to the root node of the extended perfect binary tree. The cost for step 2 is also 1, after we find the min pointer we just need to remove the link of the root node from the original Heap.

Let H be our Original Heap and our min pointer is pointing to the root node of $\hat{T}_k$ tree. The cost for step 3 here would be creating a Heap for k children of $\hat{T}_k$ tree which are $\{\hat{T}_{k-1}, \hat{T}_{k-2}, ..., \hat{T}_1, \hat{T}_0\}$. The cost would be k for linking all the trees to the new Heap. Let the new Heap be H'.

In step 4 we perform Union(H,H'). The cost here would be $\sum_{\hat{T}_i \in l}(i+1)$ which is when we perform at l link operations for the Union of both the heaps. In the last step we update the Min pointer by checking all the root nodes of the cells which costs k.

$$actual\ cost,\ c_i = 1 + 1 + k + \sum_{\hat{T}_i \in l}(i+1) + k = 2 + 2k + \sum_{\hat{T}_i \in l}(i+1)$$

(in step 4 performing l link operations takes i+1 cost for linking each $\hat{T}_i$ tree with other $\hat{T}_i$)

$$potential\ change,\ \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = c(\sum_{i=0}^{k-1}(i+2) - \sum_{\hat{T}_i \in l}(i+2))$$

(k-1 extra trees were formed after removing min node in worst case, l trees were reduced after l link operations)

$$amortized\ cost\ \hat{c}_i = c_i + \Delta_i = \ 2 + 2k + \sum_{\hat{T}_i \in l}(i+1) + c(\sum_{i=0}^{k-1}(i+2) - \sum_{\hat{T}_i \in l}(i+2))$$

$$amortized\ cost\ \hat{c}_i = 2 + 2k + (1-c)\sum_{\hat{T}_i \in l}(i+1) - cl + c(\sum_{i=0}^{k-1}(i+2)) = O(k^2)$$

(for c = 1)

$$And\ this\ is\ upper\ bounded\ by\ O(\log^2 N)$$

(we know k $\leq \log N$, k is the total number of cells)

Task 1 (c)

We need to find out amortized time complexity for Make-Heap, Minimum, Insert,Union and Extract-Min operations under Lazy Union.

We define potential function for the data structure as:

$$\Phi(D_i) = c \times \sum_{\hat{T}_k \in D_i} (k+2)$$

Similar to the lazy union in Binomial Heap we too have circular doubly linked list in our data structure and we don't perform the merge operations until Extract-Min is called. We just add trees to the data structure for both insert and union operations.

MAKE-HEAP

Cost of creating a singleton heap is $c_i = 1$. Our Potential has increased since we have created a new tree $\hat{T}_0$ and that would be 2c.

$$\Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = 2c$$

Hence the amortized cost $\hat{c}_i$,

$$\hat{c}_i = c_i + \Delta_i = 1 + 2c = \Theta(1)$$

MINIMUM

We maintain min-pointer in our data structure so the cost of finding the element is 1. Since there are no newly formed trees in the data structure, potential change is 0.

$$Actual\ cost, c_i = 1$$

$$Amortized\ cost\ \hat{c}_i = c_i + \Delta_i = 1 + 0 = \Theta(1)$$

UNION(Lazy Union)

For Union since we don't do any merging, we just combine the linked lists just as in Binomial Heaps and cost of linking two double linked list is 1 which is actual cost. And there is no change in the number of trees so the potential change is zero.

$$Actual\ cost, c_i = 1$$

$$Amortized\ cost\ \hat{c}_i = c_i + \Delta_i = 1 + 0 = \Theta(1)$$

INSERT

First we create a new heap and then do union on H so the actual cost is $c_i = 2$ which is cost of make-heap and union. As there is 1 new tree created in our system, the potential change is:

$$\Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = 2c$$

$$Amortized\ cost\ \hat{c}_i = c_i + \Delta_i = 2 + 2c = \Theta(1)$$

EXTRACT-MIN

Unlike before, in Lazy Union we have several trees of same rank. We create an array of length $\log N$ with each location containing a nil pointer. We can use this array to transform the linked list version to array version.

Step 1: Transform the linked list to array version.

Step 2: Create a new array with the children of Min-Pointer node.

Step 3: Union with newly created array.

Step 4: Transform the array version to linked list and update min-pointer.

In step 1 the cost of transforming from linked list version to array version with t trees and performing l links is $t + \sum_{\hat{T}_i \in l}(i+1)$ lets say this array created be g. In this step since we have performed l links, number of trees would decrease by l hence potential would drop by $\sum_{\hat{T}_i \in l}(i+2)$. Cutting min node and creating an another array with the children trees would cost 2 + k, lets say the new array be f. When new trees are increased, the potential would increase by k-1 new trees i.e. $\sum_{i=0}^{k-1}(i+2)$.

In step3 for the union of g,f let's say we perform l_1 links hence the actual cost will be k + $\sum_{\hat{T}_i \in l_1}(i+1)$. The potential change here would decrease because of linking l_1 trees by $\sum_{\hat{T}_i \in l_1}(i+2)$

Transforming to linked list version from the union of arrays and updating min pointer would cost t.

$$actual\ cost,\ c_i = t + (\sum_{\hat{T}_i \in l}(i+1)) + (2+k) + (\sum_{\hat{T}_i \in l_1}(i+1)) + k + t = 2 + 2t + 2k + (\sum_{\hat{T}_i \in l}(i+1)) + (\sum_{\hat{T}_i \in l_1}(i+1))$$

$$potential\ change,\ \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = c(-\sum_{\hat{T}_i \in l}(i+2)) + \sum_{i=0}^{k-1}(i+2) - \sum_{\hat{T}_i \in l_1}(i+2))$$

$$amortized\ cost,\ \hat{c}_i = c_i + \Delta_i = 2 + 2t + 2k + (\sum_{\hat{T}_i \in l}(i+1)) + (\sum_{\hat{T}_i \in l_1}(i+1)) + c(-\sum_{\hat{T}_i \in l}(i+2)) + \sum_{i=0}^{k-1}(i+2) - \sum_{\hat{T}_i \in l_1}(i+2))$$

$$amortized\,cost,\ \hat{c}_i = c_i + \Delta_i = 2+2t+2k-cl-cl1+(1-c)(\sum_{\hat{T}_i \in l}(i+1))+(1-c)\sum_{\hat{T}_i \in l_1}(i+2)+\sum_{i=0}^{k-1}(i+2)$$

(for c=1)

$$amortized\,cost,\ \hat{c}_i = c_i + \Delta_i = 2 + 2t + 2k - l - l1 + \sum_{i=0}^{k-1}(i+2) = \Theta(k^2)$$

$$Since\ here\ t, k, l\ are\ all\ order\ of\ k$$

$$And\ this\ is\ upper\ bounded\ by\ = O(\log^2 N)$$

(we know k $\leq \log N$, k is the total number of cells)

Task 1 (d)

We need to find out amortized time complexity for Make-Heap, Minimum, Insert and Extract-Min operations.

We define potential function same as before:

$$\Phi(D_i) = c \times \sum_{\hat{T}_k \in D_i} (k+2)$$

The only thing which we need to keep in mind over here is that, it's given that there are already n items in the data structure (take it as N over here). So if there's a case where this n exceeds the maximum number of elements possible in the data structure. We will be needing to increase the size of the data structure to give space for more elements. Same works for multiple Extract Mins too.

When this happens we would be needing $O(\log(n))$ time to re-build the heap from the base with the new maximum possible number of nodes i.e. $2N$. This process just needs the root nodes being pointed to the new array created.

Apart from this functionality, rest should be the same as **1b)**. Even the complexities match with **1b)**. We won't be making any modifications to the data structure or the sequence of operations for each specified operation.

Given, we can keep apart the UNION operation.

MAKE-HEAP

This remains same as **1b)** and this doesn't alter the maximum possible size of nodes in the array.

$$actual\ cost,\ c_i = 1$$

$$potential\ change,\ \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = 2c$$

(as #trees increases by 1)

$$amortized\ cost,\ \hat{c}_i = c_i + \Delta_i = 1 + 2c = \Theta(1)$$

MINIMUM

Finding the Minimum doesn't change too, as we already have a min-pointer and this is not altering the size of the data structure. So the amortized cost of Minimum also remains at $\Theta(1)$.

INSERT

While for insert there's a possibility such that we might be needing to do a build-heap and as we know this will take an extra cost of $O(\log(n))$, which is nothing but number of trees (k)

$$actual\ cost,\ c_i = Cost\ for\ MAKEHEAP + Cost\ for\ Build\ Heap + Cost\ for\ Merging + Cost\ for\ Updating\ Min$$

$$actual\ cost,\ c_i = 2 + k + \frac{l(l+1)}{2} + 1 = k + 3 + l^2/2 + l/2$$

(performing l link operations takes i+1 time for linking each $\hat{T}_i$ tree with other $\hat{T}_i$)

$$potential\ change,\ \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = c(\sum_{i=0}^{k-1}(i+2) - \sum_{i=0}^{l-1}(i+2) - \sum_{i=0}^{k-1}(i+2)) = c(-\sum_{i=0}^{l-1}(i+2))$$

$$= -3cl/2 - cl^2/2$$

$$amortized\ cost,\ \hat{c}_i = c_i + \Delta_i = k + 3 + l^2/2 + l/2 - 3cl/2 - cl^2/2$$

(for c = 1)

$$amortized\ cost,\ \hat{c}_i = 3 + k - l$$

(But we perform Build Heap Operation when the array is full, and l becomes k in this case)

$$amortized\ cost,\ \hat{c}_i = \Theta(1)$$

EXTRACT-MIN

Here the whole sequence of steps followed are similar to 1b), but there might be a case again where the number of elements deleted are much lesser than half of initial elements. In such a case, we need to build-heap the entire data structure to a smaller max number of nodes available data structure i.e. N/2. This operation of build-heap would again take $O(\log(n))$ i.e. the number of trees in the data structure (k) time similar to what we've written as part of Insert.

Let's take a case where such a build-heap functionality is also required and calculate the amortized cost. The equations are straight taken up from 1b) with an extra k added up in the front for build-heap.

$$actual\ cost,\ c_i = k + 1 + 1 + k + \sum_{\hat{T}_i \in l}(i+1) + k = 2 + 3k + \sum_{\hat{T}_i \in l}(i+1)$$

(in step 4 performing l link operations takes i+1 cost for linking each $\hat{T}_i$ tree with other $\hat{T}_i$)

$$potential\ change,\ \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = c(\sum_{i=0}^{k-1}(i+2) - \sum_{\hat{T}_i \in l}(i+2))$$

(k-1 extra trees were formed after removing min node in worst case, l trees were reduced after l link operations)

$$amortized\ cost\ \hat{c}_i = c_i + \Delta_i = \ 2 + 3k + \sum_{\hat{T}_i \in l}(i+1) + c(\sum_{i=0}^{k-1}(i+2) - \sum_{\hat{T}_i \in l}(i+2))$$

$$amortized\ cost\ \hat{c}_i = 2 + 3k + (1-c)\sum_{\hat{T}_i \in l}(i+1) - cl + c(\sum_{i=0}^{k-1}(i+2)) = O(k^2)$$

(for c = 1)

$$And\ this\ is\ again\ upper\ bounded\ by\ O(\log^2 n)$$

(we know k $\leq \log n$, k is the total number of cells)

So even in such a case, we obtained the amortized cost of Extract-Min as $O(\log^2 n)$

Task 1 (e)

Since we are already computing Insert Operation in $O(1)$ amortized cost in **1)d**. We just need to maintain the data structure here. This doesn't change our asymptotic costs for other operations as we are just maintaining the data structure.

And again we are following the same potential function as in **1)d**. Which is:

$$\Phi(D_i) = c \times \sum_{\hat{T}_k \in D_i} (k+2)$$

Here k is the height of the extended perfect binomial tree which belongs to the Heap.

MAKE-HEAP

This remains same as **1d)** and this doesn't alter the maximum possible size of nodes in the array. Here actual cost would be 1, potential change would be 2c. Therefore the amortized cost would be $\Theta(1)$

MINIMUM

Finding the Minimum doesn't change too, as we already have a min-pointer and this is not altering the size of the data structure. So the amortized cost of Minimum also remains at $\Theta(1)$.

INSERT

Inserting would be still $\Theta(1)$, as we haven't changed our data structure. It remains the same as in **1)d**.

EXTRACT-MIN

Amortized cost for extracting the minimum is still $O(\log^2 n)$.