Rishad Rahman (Deepak's Recitation) 2/25/2015

No Collaborators

Problem Set 3

3-1 a) No major parts of the pseudocode needs to be changed. This is true since all of INSERT, DELETE, + SUCCESSOR just rely on accessing clusters, summaries, + their elements none of which directly rely on the cluster size nor number of clusters. We do have to rewrite how low(x) + high(x) work by making it

$low(x) = x \bmod u^{2/3}$ $high(x) = \lfloor \frac{x}{u^{2/3}} \rfloor$

$index(i,j) = iu^{2/3} + j$

~~Otherwise the~~ other than the above, there are no other places which relied on $\sqrt{u}$ elements/cluster so we are good.

INSERT: V.summary contains $u^{1/3}$ elements + V.cluster[] has $u^{2/3}$ elements so at each step we either do $T(u) = T(u^{1/3}) + O(1)$ or $T(u) = T(u^{2/3}) + O(1)$. The latter is the worst case which gives us $O(\lg \log_{3/2} u) = O(\lg \lg u)$ which is the same as when had $\sqrt{u}$ elements/cluster.

DELETE: Again we either ~~recurse~~ do a ~~full~~ non-trivial delete on V.cluster[] or V.summary so we have the same recursion as the one in INSERT $\Rightarrow O(\lg \lg u)$ same as original.

SUCCESSOR: Same, we do either successor on v.cluster[] or v.summary $\Rightarrow O(\lg \lg u)$.

But we have a slightly higher constant factor since without rewriting we would have $\lg \log_{3/2} u > \lg \lg u$, but in the long run the performance is the same.

b) INSERT(V,x):

```
1   if V.min=None: V.min = V.max = x; return
2   if x < V.min: swap x <-> V.min
3   if x > V.max: swap x <-> V.max
4   if V.cluster[high(x)].min = None:
5       Insert(V.summary, high(x))
6       Insert(V.cluster[high(x)], low(x))
```

```
2 if V.min = V.max:
3   if x < V.min:
4     V.min = x; return
5   else:
6     V.max = x; return
```

return later

This is pretty much the same as in ∧ only maintaining the minimum except the O(1) operation for swapping the previous maximum element with x when need be. Otherwise we proceed as normal hence the runtime is still O(lg lg u). * We also needed to add lines 2-6 to make sure we don't try to insert V.min nor V.max

the INSERT operation

DELETE(V,x)

```
1   if x = V.min:
2     i = V.summary.min
      if i = None: V.min = V.max = None
        return
      x = V.min = index(i, V.cluster[i].min)
3     if i = None:
4       if x = V.max: V.min = V.max = None
5         return
6       else V.min = V.max
7         return
8     x = V.min = index(i, V.cluster[i].min)
9   if x = V.max:
10    i = V.summary.max
11    if i = None:
12      if x = V.min: V.min = V.max = None
13        return
14      else V.max = V.min
15        return
16    x = V.max = index(i, V.cluster[i].max)
17  Delete(V.cluster[high(x)], low(x))
18  if V.cluster[high(x)].min = None:
19    Delete(V.summary, high(x))
```

} Tree becomes empty

} only one element remaining, the max which is in its own field

} empty tree

} only previous min is remaining

This is pretty much the same as the code presented in the lecture notes except with a few modifications. First if V.summary.min is none when x=V.min, this doesn't rule out the possibility that we have another element, V.max, which isn't in the tree if we check this in line 6 when x was not the only element in our structure. Second we do the analogous case for when x=V.max in lines 9-16 & the logic is exactly the same. The rest is exactly from the pseudocode in the lecture notes, this means that the two calls to Delete in lines 17 & 19 work in the same fashion & the additional checks when maximum field is not in the tree contribute constant time $\Rightarrow O(\lg \lg u)$.

```
SUCCESSOR(V,x):
    ~~HIGH(x)~~
1   if x < V.min: return V.min
2   if x ≥ V.max: return None
3   i = high(x)
4   if low(x) < V.cluster[i].max:
5       j = Successor(V.cluster[i], low(x))
6   else: i = Successor(V.summary, high(x))
        ~~if V.cluster[i].min~~
7       if i = None
8           return V.max
9       else
10          ~~return~~ j = V.cluster[i].min
11  return index(i,j)
```

Exactly the same pseudocode but the checks for $x < V.min$ & $x > V.max$ in lines 1-2 & the check for No successor in line 7 which means we return the max. These add constant time hence our runtime is still $O(\lg \lg u)$.