

CS5800: Algorithms — Spring '21 — Virgil Pavlu

Homework 9

Submit via [Gradescope](#)

Name: Ketaki Nitin Kolhatkar

Collaborators:

Instructions:

- Make sure to put your name on the first page. If you are using the \LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3rd edition. While the 2nd edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3rd edition.

1. (25 points) Exercise 17.3-3. (Hint: a reasonable potential function to use is $\phi(D_i) = kn_i \cdot \ln n_i$ where n_i is the number of elements in the binary heap, and k is a big enough constant. You can use this function and just show the change in potential for each of the two operations.)

Solution:

If the potential function is $\phi(D_i) = \sum_{k=1}^i \lg k$, then the cost would be:

$$c'_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

$$c'_i \leq \lg i + \sum_{k=1}^i \lg k - \sum_{k=1}^{i-1} \lg k$$

$$c'_i \leq \lg i + \lg i$$

$$c'_i \leq 2\lg i$$

Therefore, we would get $c'_i = \theta(\lg n)$

Now we calculate for EXTRACT MIN:

$$c'_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

$$c'_i \leq \lg i + 1 + \sum_{k=1}^{i-1} - \sum_{k=1}^i \lg k$$

$$c'_i \leq 1$$

$$c'_i = \theta(1)$$

2. (25 points) Exercise 17.3-6.

Solution:

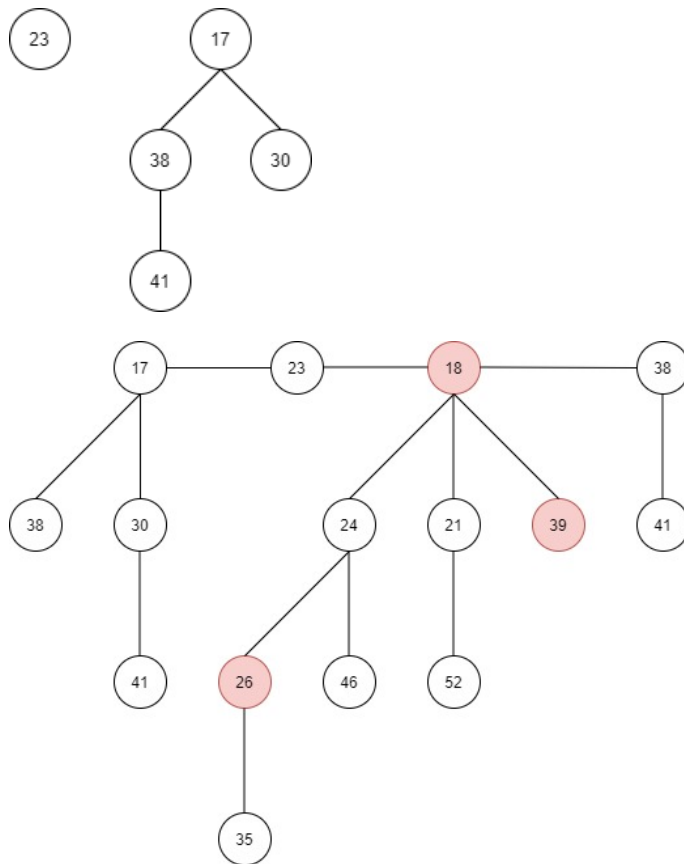
During Enqueue we get a time complexity of $O(1)$ to push the elements using 2 stacks. During dequeue, we check if stack 2 is empty. If it is, we pop the elements of stack 1 to 2. This would take a time complexity of $O(n)$. The rest would take a time complexity of $O(1)$. Overall it takes 1 unit to push the elements to stack1. Then we would pop the elements from stack 1 and push it to stack 2 which would take 2 units. Lastly, it takes 1 unit to pop an element from stack 2. Hence, in all it takes 4 units to get in and then out of the queue implemented stack.

To accommodate the pop from S1 and push to S2 we assign another extra cost of 2. Using aggregate analysis we get the amortized cost of $O(1)$ of Enqueue and Dequeue.

3. (25 points) Exercise 19.2-1.

Solution:

We have the subtree with the roots at 24, 17, 23 added to the root list. We the set H.min to 18. This has a degree of 2 with the subtree with a root of 18. The subtree with root 38 has a degree of 1. When we consider the subtree with the root 24, we get a degree of 2. We place 24 under 18 for making a subheap. Next we check the heap with root 17. We place the heap with root 38 below 17. Next we consider the heap with root 23. H.min would be set at 17. The 3 distinct heaps in the root list are:



4. (50 points) Implement binomial heaps as described in class and in the book. You should use links (pointers) to implement the structure as shown in the fig 1.

Your implementation should include the operations: *Make-heap, Insert, Minimum, Extract-Min, Union, Decrease-Key, Delete*

```
Make-Binomial-Heap()
```

```
head[H] = NIL
```

```
return H
```

```
Binomial-Heap-Minimum(H)
```

```
y := NIL
```

```
x := head[H]
```

```
min := infinity
```

```
while x != NIL
```

```
    do if key[x] < min
```

```
        then min := key[x]
```

```
            y := x
```

```
            x := sibling[x]
```

```
return y
```

```
Binomial-Link(y,z)
```

```
p[y] := z
```

```

sibling[y] := child[z]
child[z] := y
degree[z] := degree[z] + 1

```

```

Binomial-HeapMerge(H1,H2)
a = head[H1]
b = head[H2]
head[H1] = Min-Degree(a, b)
if head[H1] = NIL
    return
if head[H1] = b
    then b = a
a = head[H1]
while b != NIL
    do if sibling[a] = NIL
        then sibling[a] = b
        return
    else if degree[sibling[a]] < degree[b]
        then a = sibling[a]
        else c = sibling[b]
        sibling[b] = sibling[a]
        sibling[a] = b
        a = sibling[a]
        b = c

```

```

Binomial-Heap-Union(H1,H2)
H := Make-Binomial-Heap()
head[H] := Binomial-Heap-Merge(H1,H2)
free the objects H1 and H2 but not the lists they point to
if head[H] = NIL
    then return H
prev-x := NIL
x := head[H]
next-x := sibling[x]
while next-x != NIL
    do if (degree[x] != degree[next-x]) or
        (sibling[next-x] != NIL
         and degree[sibling[next-x]] = degree[x])
    then prev-x := x
        x := next-x
    else if key[x] <= key[next-x]
        then sibling[x] := sibling[next-x]
        Binomial-Link(next-x,x)
    else if prev-x = NIL
        then head[H] = next-x

```

```

        else sibling[prev-x] := next-x
        Binomial-Link(x,next-x)
        x := next-x
    next-x := sibling[x]
return H

Binomial-Heap-Insert(H,x)
H' := Make-Binomial-Heap()
p[x] := NIL
child[x] := NIL
sibling[x] := NIL
degree[x] := 0
head[H'] := x
H := Binomial-Heap-Union(H,H')

Binomial-Heap-Extract-Min(H)
find the root x with the minimum key in the root list of H,
    and remove x from the root list of H
H' := Make-Binomial-Heap()
reverse the order of the linked list of x's children
    and set head[H'] to point to the head of the resulting list
H := Binomial-Heap-Union(H,H')
return x

Binomial-Heap-Decrease-Key(H,x,k)
if k > key[x]
    then error "new key is greater than current key"
key[x] := k
y := x
z := p[y]
while z != NIL and key[y] < key[z]
    do exchange key[y] and key[z]
        if y and z have satellite fields, exchange them, too.
        y := z
        z := p[y]

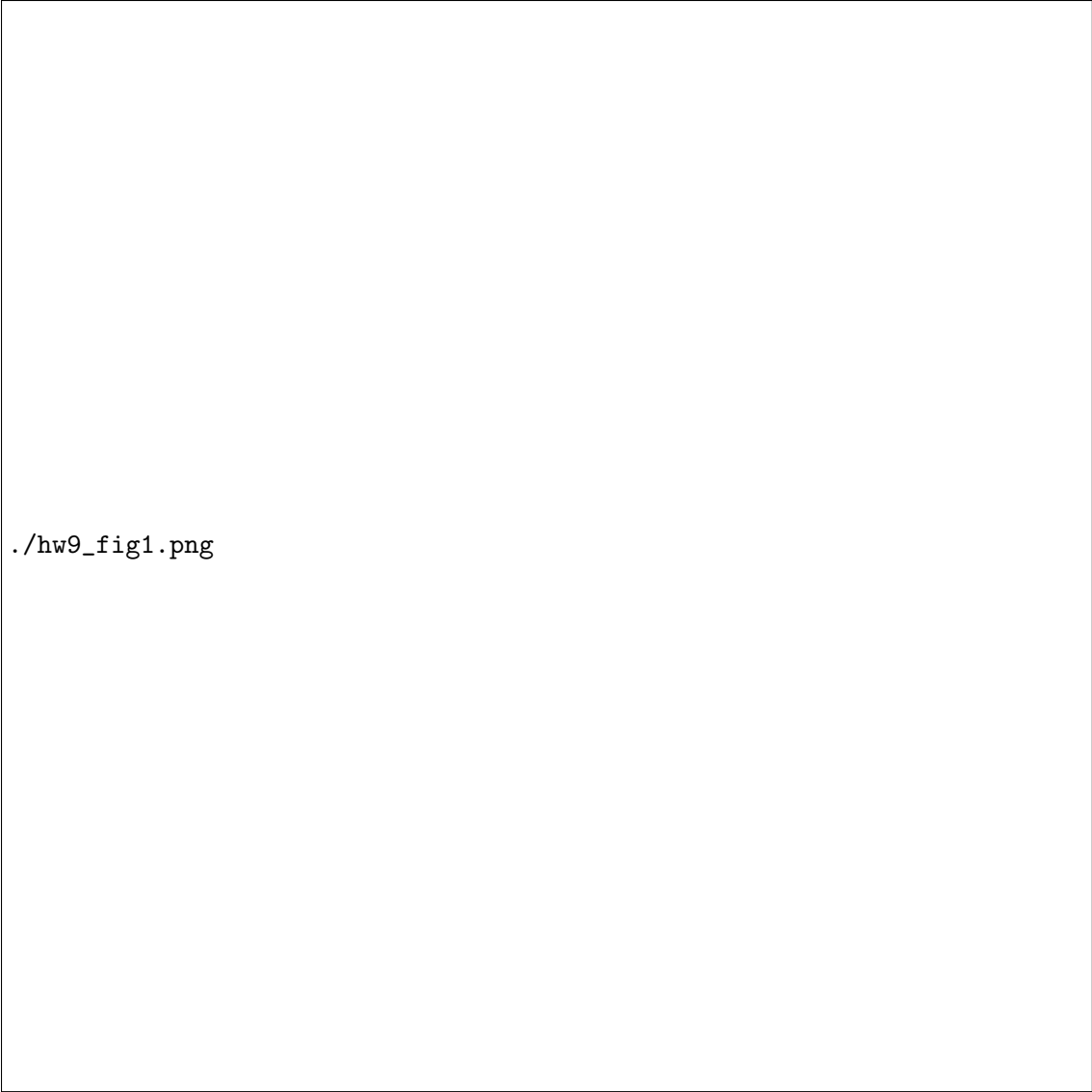
Binomial-Heap-Delete(H,x)
Binomial-Heap-Decrease-Key(H,x,-infinity)
Binomial-Heap-Extract-Min(H)

```

Make sure to preserve the characteristics of binomial heaps at all times:

- (1) each component should be a binomial tree with children-keys bigger than the parent-key;
- (2) the binomial trees should be in order of size from left to right. Test your code several arrays set of random generated integers (keys).

Solution:



./hw9_fig1.png

Figure 1: Binomial Heaps

5. **(Extra Credit)** Find a way to nicely draw the binomial heap created from input, like in the figure.

Solution:

6. **(Extra Credit)** Write code to implement Fibonacci Heaps, with discussed operations: ExtractMin, Union, Consolidate, DecreaseKey, Delete.

Solution:

7. **(Extra Credit)** Figure out what are the marked nodes on Fibonacci Heaps. In particular explain how the potential function works for FIB-HEAP-EXTRACT-MEAN and FIB-HEAP-DECREASE-KEY operations.

Solution: