Exercises for Units 16 & 17

Problem 2.

a) After the DeleteMin operation, we create at most MAX-RANK new trees. We need to merge these with the existing trees.

Let's create a vector $[0, \ldots, \text{MAX-RANK}]$ that stores at position i a pointer to a linked list containing all trees of rank i. This can be done by traversing all our trees and putting them in the appropriate bucket in time $O(t)$, where t is the total number of trees:

```
For all trees t:
    add t in R[rank(t)].
```

Now we traverse t backwards, link all possible pairs of trees and move them to the next bucket. Doing this backwards ensures that a linked tree does not participate in another linking:

```
for j from MAX-RANK downto 1 do:
    while size(R[j]) ≥ 2 do:
        T ← obtained by linking T1 and T2 in R[j].
        add T in R[j+1].
        remove T1, T2 from R[j].
        update min.
```

→ time $O(t + \text{MAX-RANK})$.

b) We use the same potential as in class:

$\Phi(Q) = c \cdot \#\text{trees}$

#trees before: t

Let x_i be the #trees with rank i. Then $t = \sum_{i=0}^{\text{MAXRANK}} x_i$.

After linking in bucket i we have x_i' trees with $x_i' \leq 1 + \frac{x_{i-1}}{2}$ (1 ↑ left over after linking rank i; $\frac{x_{i-1}}{2}$ ← obtained from linking rank $i-1$).

$\Rightarrow$ #tree after linking $t' = \sum_{i=0}^{\text{MAXRANK}} \left(1 + \frac{x_{i-1}}{2}\right) = \frac{t}{2} + \text{MAX-RANK} + 1$.

$\Rightarrow$ amortized cost = actual cost + $\Delta\Phi = O(t + \text{MAX-RANK}) + c\left(\text{MAX-RANK} + 1 - \frac{t}{2}\right)$.

If we pick c large enough to cover the constant hidden inside $O(t)$, the amortized cost becomes: $O(\text{MAX-RANK})$.

Exercises for Unit 16 & 17

(3.) a) start with all n nodes being in one tree of rank $\log n$

1. delete_min → the big tree loses the root and falls into $\log n - 1$ trees: $B_0, B_1, \dots, B_{\log n - 1}$
2. insert old minimum back
3. insert dummy value
4. delete dummy → triggers merging of all trees into the big tree we started from

→ 2 inserts and 2 deletes that together cost $\Omega(\log n)$ (more precisely $\Theta(\log n)$)

- we can repeat these 4 steps as much as we want

⇒ $4 \cdot k$ operations with $\Theta(2 \cdot k \cdot \log n) = \Theta(k \log n)$ cost

⇒ cost per operation is $\Theta(\log n)$, so in particular $\Omega(\log n)$

b) answer: no

- otherwise, we could sort n elements in $O(n)$ time
  → insert them into the queue one by one and then do delete_min n times

Algorithms and Data Structures (WS15/16)

Example Solutions for Units 16 & 17

Problem 4

a) Take $k = 0$ as induction base. We have

$$F_{0+2} = 1 + 0 = 1 = (\frac{1+\sqrt{5}}{2})^0$$

Our induction hypothesis is that for $0 \leq i \leq k+1$, $F_i \geq (\frac{1+\sqrt{5}}{2})^{i-2}$. We get for F_{k+2}

$$\begin{aligned} F_{k+2} &= F_{k+1} + F_k \\ &\geq (\frac{1+\sqrt{5}}{2})^{k-1} + (\frac{1+\sqrt{5}}{2})^{k-2} \\ &= (\frac{1+\sqrt{5}}{2})^{k-2}(1 + \frac{1+\sqrt{5}}{2}) \end{aligned}$$

Note that

$$(\frac{1+\sqrt{5}}{2})^2 = \frac{6+2\sqrt{5}}{4} = \frac{3+\sqrt{5}}{2} = (1 + \frac{1+\sqrt{5}}{2})$$

We use this in the equation above to get

$$F_{k+2} \geq (\frac{1+\sqrt{5}}{2})^{k-2}(\frac{1+\sqrt{5}}{2})^2 = (\frac{1+\sqrt{5}}{2})^k$$

b) Proof by induction on k. If $k = 0$, then

$$1 + \sum_{i=0}^{k} F_i = 1 + F_0 = 1 + 0 = 1 = F_2$$

Our induction hypotheses is that $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$, and we have that

$$F_{k+2} = F_k + F_{k+1} = F_k + 1 + \sum_{i=0}^{k-1} F_i = 1 + \sum_{i=0}^{k} F_i$$

c) We show by induction on k that $s_k \geq F_{k+2}$ for all nonnegative integer k. The bases, for $k = 0$ and $k = 1$, are trivial. For the inductive step, we assume that $k \geq 2$ and that $s_i \geq F_{i+2}$ for $i = 0, 1, \ldots, k-1$. We have

$$
\begin{aligned}
s_k &\geq 2 + \sum_{i=0}^{k-2} s_i \\
&\geq 2 + \sum_{i=0}^{k-2} F_{i+2} && \text{by induction hypothesis} \\
&= 2 + F_2 + F_3 + \ldots + F_k \\
&= 1 + \sum_{i=0}^{k} F_i \\
&= F_{k+2} && \text{by b)}
\end{aligned}
$$

Problem 5

We show how to achieve this by induction.

Base case: Consider inserting two keys, namely x and $x+1$. Then insert $-\infty$ and call DeleteMin, which would immediately delete this element and force linking. This would create the linear chain of 2 elements.

Induction: Suppose we have a linear chain of i elements, the smallest being x and the largest being $x+k$, $x, k > 0$. Consider inserting the elements $x-2$ and $x-1$. Then insert $-\infty$ and call DeleteMin. The two new inserted nodes would merge to form a chain of two elements, which is then merged with the longer chain. The new tree is such that the left child of the root is a linear chain of i elements and the right child is just the a single node. Now invoking the remove$(x-1)$, we get a linear chain of $i+1$ elements.

Hence, following this scheme we can always create a Fibonacci heap of height n in $O(n)$ operations.