Solution:

1. As defined, for integers \( k \geq 0 \) and \( j \geq 1 \),
\[
A_k(j) = \begin{cases} 
  j + 1, & \text{if } k = 0, \\
  A_{k-1}(j), & \text{if } k \geq 1
\end{cases}
\]
we only consider cases when \( j \geq 1 \) here.

   • When \( j = 1 \), \( A_3(1) = A_2^2(1) = A_2(A_2(1)) = 2047 \), \( tower(1) = 2^{tower(0)} = 2 \). Obviously we have \( A_3(1) > tower(1) \).

   • Assume when \( j = i - 1 \), we have \( A_3(i - 1) \geq tower(i - 1) \); now we prove for \( j = i \), we still have \( A_3(i) \geq tower(i) \). According to Lemma 21.3, we have \( A_2(j) = 2^{j+1}(j + 1) - 1 \). Also we know the function \( A_k(j) \) strictly increases with both \( j \) and \( k \). So:
\[
A_3(i) = A_2^{i+1}(i) = A_2(A_2^i(i)) > A_2(A_2^i(i - 1)) = A_2(A_3(i - 1)).
\]
\[
\geq A_2(tower(i - 1)) = 2^{tower(i-1)+1}(tower(i - 1) + 1) - 1 > 2^{tower(i-1)} = tower(i)
\]

By induction hypothesis, we have proved that \( A_3(j) \geq tower(j), \forall j \geq 1 \).

2. \textbf{Binomial-Heap-Extract-Min}(H)
   1. \( x = \text{Binomial-Heap-Minimum}(H) \)
   2. Remove \( x \) from the root list of \( H \)
   3. \( H' = \text{Make-Binomial-Heap}() \)
   4. Reverse the order of the linked list of \( x \)'s children, and set \( H'.head \) to point to the head of the resulting list.
   5. \( H = \text{Binomial-Heap-Union}(H, H') \)
   6. \textbf{return } x

\textbf{Binomial-Heap-Minimum}(H)
   1. \( y = \text{NIL} \)
   2. \( x = H.head \)
   3. \( min = \infty \)
   4. \textbf{while } x \neq \text{NIL} \textbf{ do}
   5. \textbf{if } x.key < min \textbf{ then}
   6. \( min = x.key \)
   7. \( y = x \)
   8. \( x = x.sibling \)
   9. \textbf{return } y
Binomial-Heap-Union($H_1, H_2$)
1: $H = \text{Make-Binomial-Heap}()$
2: $H.head$ merges the root lists of $H_1$ and $H_2$ into a single linked list that is sorted by degree into monotonically increasing order.
3: Free the objects $H_1$ and $H_2$ but not the lists they point to
4: if $H.head = \text{NIL}$ then
5: \text{return } H
6: prevx = NIL
7: $x = H.head$
8: nextx = $x.sibling$
9: while $\text{nextx} \neq \text{NIL}$ do
10: \text{if } ($x.\text{degree} \neq \text{nextx.\text{degree}}$) or ($\text{nextx.sibling} \neq \text{NIL}$ and $\text{nextx.sibling.\text{degree}} = x.\text{degree}$) then
11: \text{prevx } = x
12: \text{x } = \text{nextx}
13: else
14: \text{if } $x.\text{key} \leq \text{nextx.\text{key}}$ then
15: $x.\text{sibling} = \text{nextx.\text{sibling}}$
16: \text{Binomial-Link}($\text{nextx}, x$)
17: else
18: \text{if } prevx = \text{NIL} then
19: $H.\text{head} = \text{nextx}$
20: else
21: prevx.\text{sibling} = nextx
22: \text{Binomial-Link}($x, \text{nextx}$)
23: $x = \text{nextx}$
24: nextx = $x.\text{sibling}$
25: \text{return } H

Binomial-Link($y, z$)
1: $y.p = z$
2: $y.sibling = z.\text{child}$
3: $z.\text{child} = y$
4: $z.\text{degree} = z.\text{degree} + 1$

Binomial-Heap-Decrease-Key($H, x, k$)
1: if $k > x.\text{key}$ then
2: \text{error } "\text{new key is greater than current key}" 
3: $x.\text{key} = k$
4: $y = x$
5: $z = y.p$
6: while $z \neq \text{NIL}$ and $z.\text{key} > y.\text{key}$ do
7: exchange $z$ with $y$
8: $y = z$
9: $z = y.p$
**Binomial-Heap-Delete**($H, x$)

1. **Binomial-Heap-Decrease-Key**($H, x, -\infty$)
2. **Binomial-Heap-Extract-Min**($H$)

3. **Problem 19-3 on page 529**

   **a.** The algorithm is given below. According to the analysis in CLRS, page 519, the amortized running time of the implementation of **Fib-Heap-Change-Key** is still $O(\lg(n))$

   **Fib-Heap-Change-Key**($H, x, k$)
   1. if $k < x.key$ then
   2. **Fib-Heap-Decrease-Key**($H, x, k$)
   3. else $k > x.key$
   4. **Fib-Heap-Delete**($H, x$)
   5. **Fib-Heap-Insert**($H, x, k$)

   **b.** In order to implement the **Fib-Heap-Prune**($H, r$), we modify the structure by adding double linked list among all the leaf nodes, which helps us easily extract any leaf node. To do the prune operation, we randomly choose a leaf node, and remove it from both the leaves list and its parent’s list, as shown below:

   **Fib-Heap-Prune**($H, r$)
   1. for $i \leftarrow 1$ to $\min(r, H.n)$ do
   2. $x \leftarrow$ random leaf node
   3. remove $x$ from parent’s children list
   4. remove $x$ from leaves list

   Due to such structural modification, we add one more cost $s(H)$ to the original potential function, which is related to the size of heap. Thus, the new potential function is

   $$\Phi'(H) = \Phi(H) + s(H) = t(H) + 2m(H) + s(H)$$

   Assume we remove $q = \min(r, H.n)$ nodes in this operation, and removing such $q$ nodes brings $c$ times cascading cuts. Similar to decrease key analysis in CLRS, page 521, the actual cost in this operation should be $c + q$: $c$ times cascading cut and adding new leaf nodes, $q$ times removing nodes.

   Thus, the potential change of original part $\Phi(H)$ is still $4 - c$, while the additional potential change is $-q$.

   Thus, the amortized cost should be $c + q + 4 - c - q = 4 = O(1)$. 