

1. 对于递归式 $T(n) = T(2n/3) + 2^{\lceil \lg n \rceil}$, 给出 $T(n)$ 的一个渐近紧确界。

(注: $2^{\lg n} = n$)

解:

定义 $F(n) = F(2n/3) + n, G(n) = G(2n/3) + 2n$, 则由主定理显然有: $F(n) = \Theta(n), G(n) = \Theta(n)$, 而 $n = 2^{\lg n} \leq 2^{\lceil \lg n \rceil} \leq 2^{(\lg n)+1} = 2n$, 因此 $F(n) \leq T(n) \leq G(n)$, 故 $T(n) = \Theta(n)$

2. 给出算法 RANDOMIZED-QUICKSORT 的伪代码，并分析 RANDOMIZED-QUICKSORT($A, 1, n$) 运行过程中，在最坏和最好情况下，RANDOMIZED-QUICKSORT 分别被调用了多少次？（以 Θ 符号形式给出答案）

在最坏和最好情况下，RANDOMIZED-QUICKSORT 分别被调用了 $\Theta(n)$ 次。

Algorithm 1 PARTITION(A, p, r)

```

1:  $x = A[r]$ 
2:  $i = p - 1$ 
3: for  $j = p$  to  $r - 1$  do
4:   if  $A[j] \leq x$  then
5:      $i = i + 1$ 
6:      $A[i] \leftrightarrow A[j]$ 
7:  $A[i + 1] \leftrightarrow A[r]$ 
8: Return  $i + 1$ 

```

Algorithm 2 RANDOMIZED-PARTITION(A, p, r)

```

1:  $i = \text{RANDOM}(p, r)$ 
2:  $A[r] \leftrightarrow A[i]$ 
3: Return PARTITION( $A, p, r$ )

```

Algorithm 3 RANDOMIZED-QUICKSORT(A, p, r)

```

1: if  $p < r$  then
2:    $q = \text{RANDOMIZED-PARTITION}(A, p, r)$ 
3:   RANDOMIZED-QUICKSORT( $A, p, q - 1$ )
4:   RANDOMIZED-QUICKSORT( $A, q + 1, r$ )

```

3. 给定 n 个已经排好序的元素 $a_0, \dots, a_{n-1}$, 请给出时间复杂度为 $O(n)$ 的算法来判断是否存在元素对满足 $a_i = -a_j, i \neq j$ 。给出伪代码, 并说明算法工作原理。

解：

做向内搜索。基本思想是：如果 $a_{left} + a_{right} > 0$ ，那么 $right--$ ，否则如果 $a_{left} + a_{right} < 0$ ，那么 $left++$ 。

Algorithm 4 向内搜索

```
1:  $l = 0; r = n - 1$ 
2: while  $l < n$  do
3:   if  $a_l + a_r > 0$  then
4:      $n = n - 1$ 
5:   else
6:     if  $a_l + a_r < 0$  then
7:        $l = l + 1$ 
8:   else
9:     Return True
10: Return False
```

4. 对于图一所给的斐波那契堆, 黑色节点表示结点的 mark 属性为 true:
- 1) 抽取最小结点 (需要考虑 CONSOLIDATE 操作), 画出操作后最后的结果。
 - 2) 在 1) 的基础上, 把关键字 25 减为 3, 画出操作后的最后的结果 (需考虑级联操作)。

Given a Fibonacci heap below. the mark attribute of black node is true:

- 1) Please give the result of extracting the minimum node (CONSOLIDATE should be considered).
- 2) Please give the result of decrease the value of key 25 down to 3, based on the result of 1).

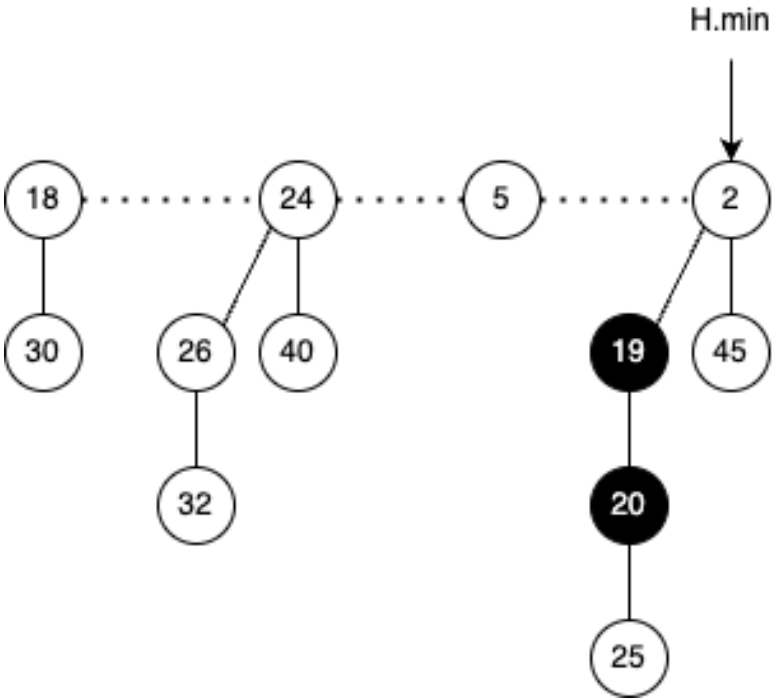
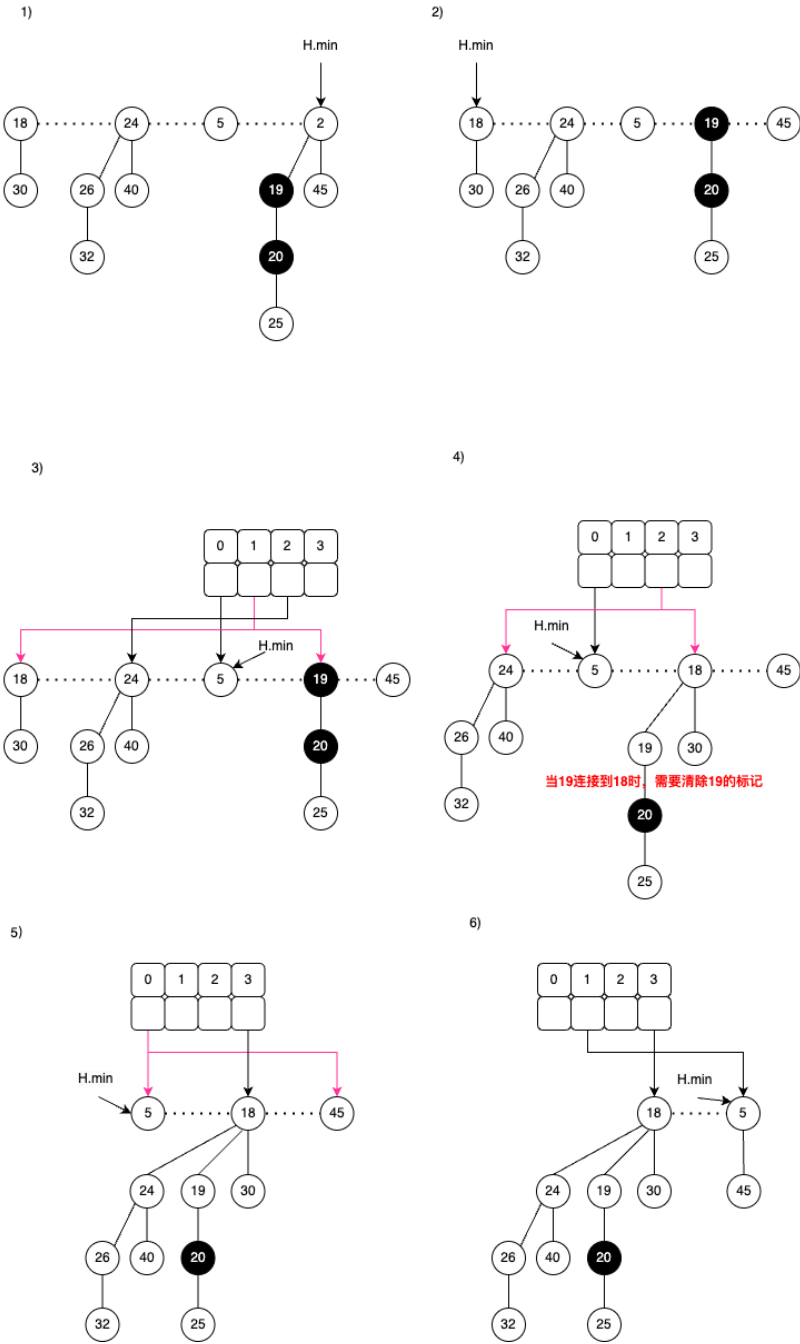


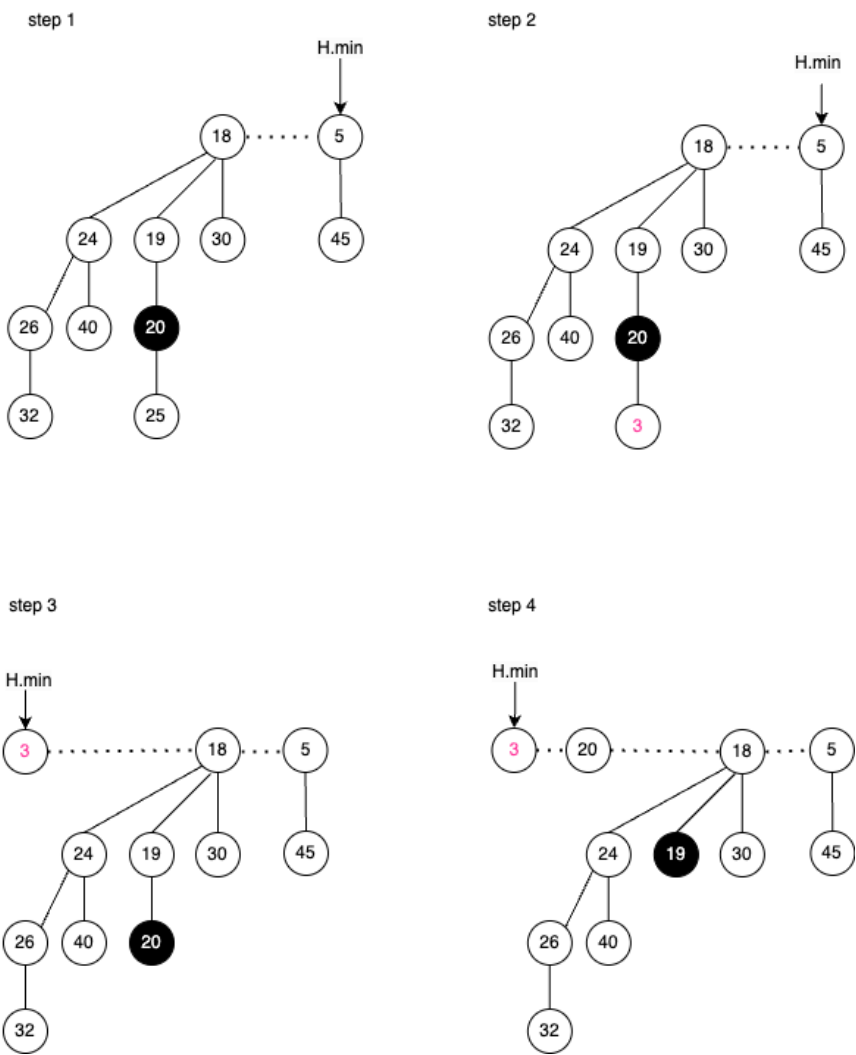
图 1: 斐波那契堆
Fig. 1 Fibonacci Heap

1)

本题度数为1的树的组合方式共有3种。所以有三种解法。其中19为根结点的解法，19和20均为marked，其余的做法中，仅20为marked结点。



2)



3. 对于递归式 $T(n) = T(n/3)/2 + T(2n/3)/2 + \lg n$, 请给出一个好的渐近上界, 并用代入法验证。

解：

每一层的代价可以表示为 $\frac{1}{2^i} \lg \frac{3^{i-1}-1}{2} n^{2^i}$ ，简单放缩后得到每层的代价约为 $O(\lg n)$ ，因此估计它的渐近上界为 $O(\lg^2 n)$ ，用代入法验证：

$$\begin{aligned}
 T(n) &= T(n/3)/2 + T(2n/3)/2 + \lg n \\
 &= \frac{1}{2}c(\lg^2(n/3)) + \frac{1}{2}c(\lg^2(2n/3)) + \lg n \\
 &= c\lg^2 n + (-\lg 3 + \lg 2/3 + (1/c))c\lg n + ((1/2)\lg^2 3 + (1/2)\lg^2(2/3))c \quad (c > 1/(\lg 3 + \lg(3/2))) \\
 &\leq c\lg^2 n
 \end{aligned}$$

其中，常数项 $C = ((1/2)\lg^2 3 + (1/2)\lg^2(2/3))c$ 被省略。

同时，使用 Akra-Bazzi 方法也可以求出该解。

1. 有 n 个木块，大小分别为 $a_1, \dots, a_n$ ；另有 m 个大小为 V 的容器。定义一个容器被填满当且仅当其中放入了两个木块 a_i, a_j ，且 $a_i + a_j \leq V$ 。每个木块只能使用一次。请给出时间复杂度为 $O(n \log n)$ 的算法来判断是否存在一种木块分配方案，使得这 m 个容器都被填满。给出伪代码，并说明算法的工作原理。

注意，你的算法不需要输出方案，只需要输出是或者否。

解：

先给 $a_1, a_2, \dots, a_n$ 排序，然后维护两个指针 $left, right$ 和一个计数器 num 表示当前 $a_1, a_2, \dots, a_{left}$ 中有多少个木块没有被使用过，得到最多能够填满多少个容器。基本思想是：从大到小枚举 $right$ ，对于目前最大的 $right$ ，找到使得 $a_{left} + a_{right} \leq V$ 成立的最大 $left$ ，那么 $a_1, a_2, \dots, a_{left}$ 中没有用过的木块和 a_{right} 一起就都是一个合法的可以放到容器里的木块配对方案。如果这样的 $a_1, a_2, \dots, a_{left}$ 存在，那么就任选一个和 a_{right} 配对。（这是因为：如果 $a_1, a_2, \dots, a_{left}$ 可以和 a_{right} 配对，那么也一定能和比 a_{right} 小的木块配对，且 $a_{right+1}, \dots, a_n$ 的配对方案都已经决定过了，那么现在和 a_{right} 配对一定是最优解之一。）之后 $num--$, $right--$ ，前一个 $right$ 求出的 $left$ 值就可以作为初始值继续往上搜索下一个 $right$ 对应的 $left$ 。直到 $right \leq left$ ，这表明剩下的 num 个没有用到的木块都可以两两组合（这是因为上一轮的搜索保证了 $a_{left} + a_{right+1} \leq V$ ），那么再给答案加上 $\lfloor \frac{num}{2} \rfloor$ 。最后判断填满的容器数是否 $\geq m$ 即可。

时间复杂度：排序时间复杂度 $O(n \log n)$ ，向内搜索时间复杂度 $O(n)$ ，总时间复杂度 $O(n \log n)$ 。

Algorithm 1 向内搜索

```

1: sort a
2: l = 0; r = n; num = 0; ans = 0;
3: while l < r do
4:   while l + 1 < r and al+1 + ar ≤ V do
5:     l ++; num ++;
6:   if num > 0 then
7:     num --; ans ++;
8:   r --;
9: ans = ans + num/2;
10:
11: if ans ≥ m then
12:   Return True
13: else
14:   Return False

```

2. 扩张红黑树使得它能以 $O(1)$ 的时间给出动态数集 Q 中两个最接近的数之间的差值 (MIN-GAP)。例如, $Q = \{1, 5, 9, 15, 18, 22\}$, MIN-GAP 返回 $18 - 15 = 3$, 因为 15 和 18 是 Q 中两个最接近的数。写出该数据结构并说明 MIN-GAP 如何计算。

解: 将红黑树的 key 设为动态数集 Q 中的值。结点 x 的附加值包含以 x 为根的子树中两个最接近的数之间的差值 ($x.min_gap$), 以 x 为根的子树中的最大值 ($x.max$), 和以 x 为根的子树中的最小值 ($x.min$)。叶子结点的 min_gap 设为 ∞ , 非叶子结点 x 的 $x.min_gap = \min\{x.left.min_gap, x.right.min_gap, x - x.left.max, x.right.min - x\}$ 。因此直接求根结点的 min_gap 即可在 $O(1)$ 的时间给出 MIN-GAP。

4. 对于图一所给的斐波那契堆, 黑色节点表示结点的 mark 属性为 true:
- 1) 抽取最小结点, 画出操作后最后的结果 (需要考虑 CONSOLIDATE(合并) 操作,)。
 - 2) 在 1) 的基础上, 把关键字为 35 的节点的值降低为 4, 画出操作后的最后的结果 (需考虑 CASCADING-CUT(级联切断) 操作)。

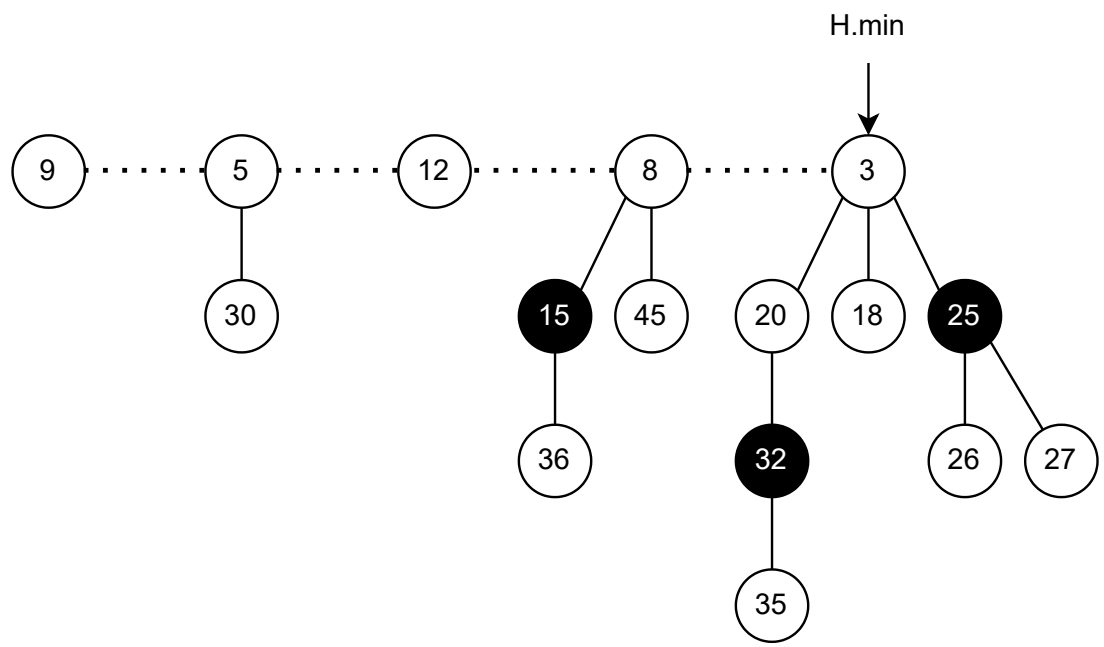
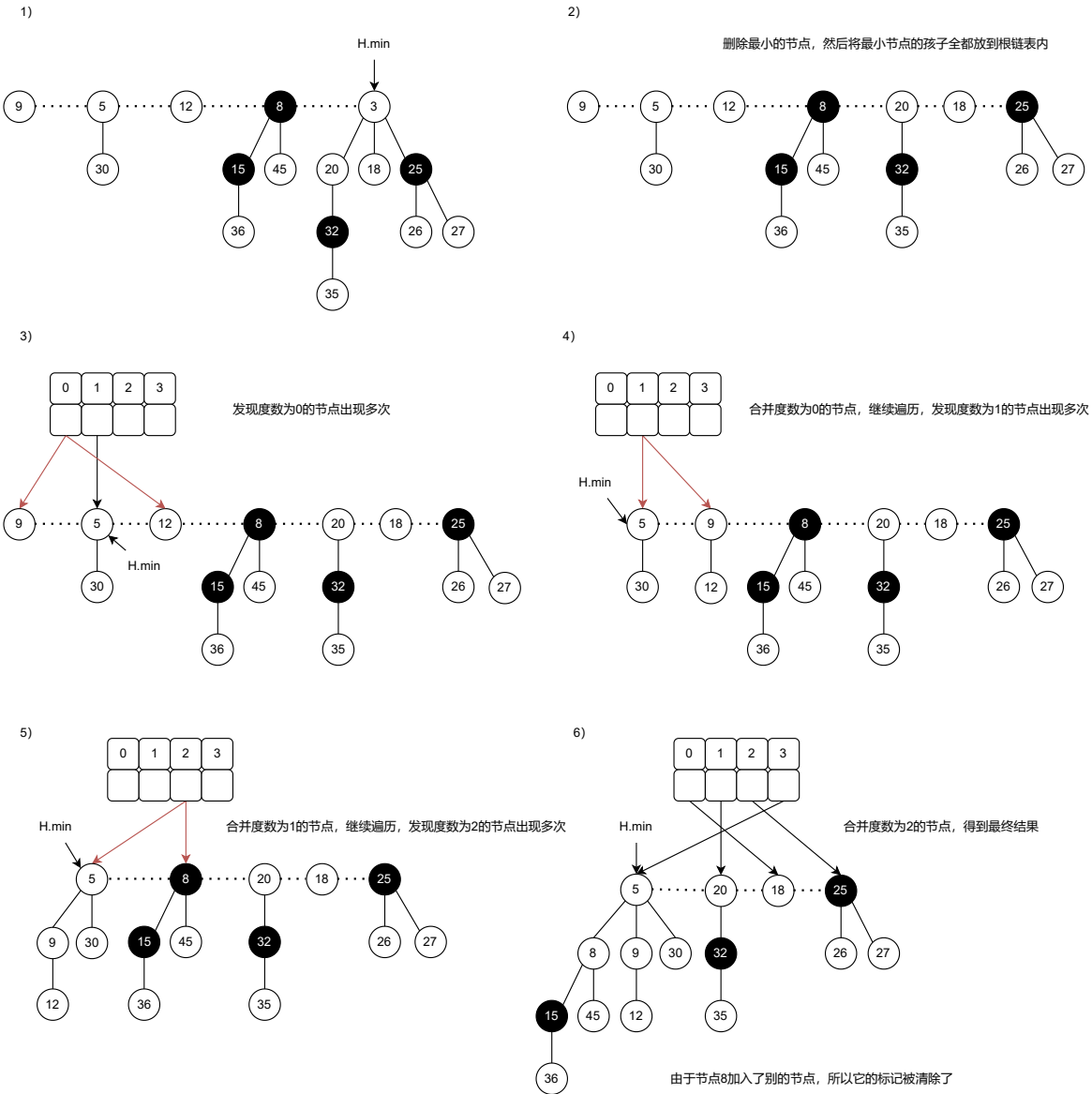
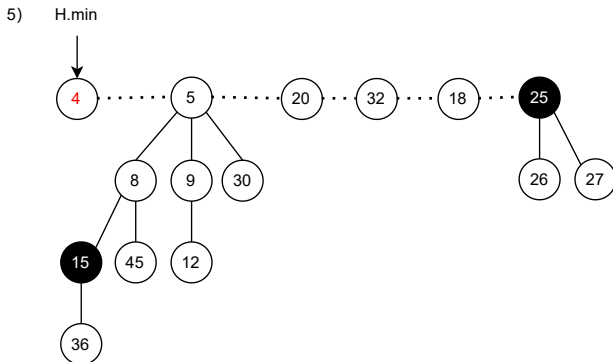
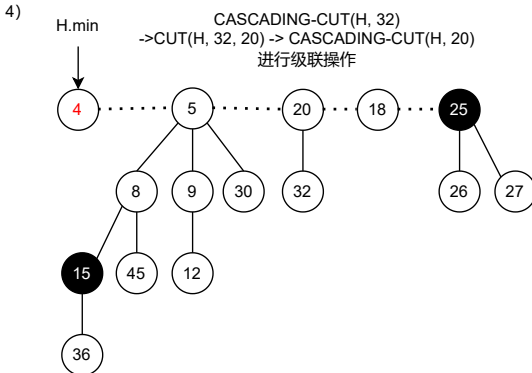
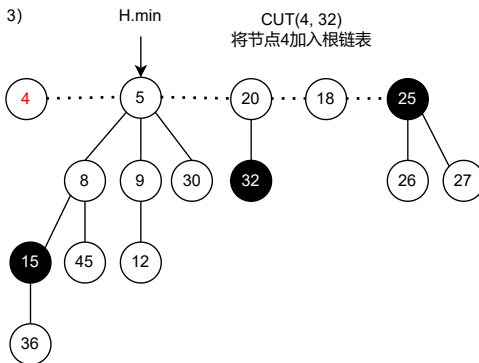
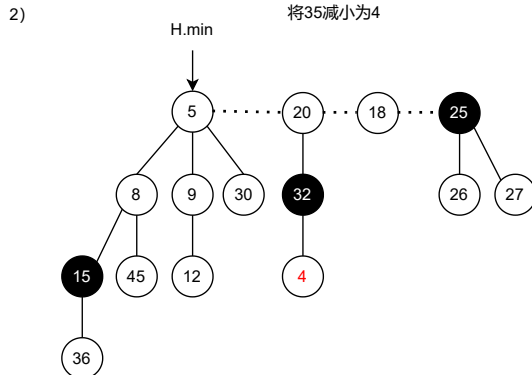
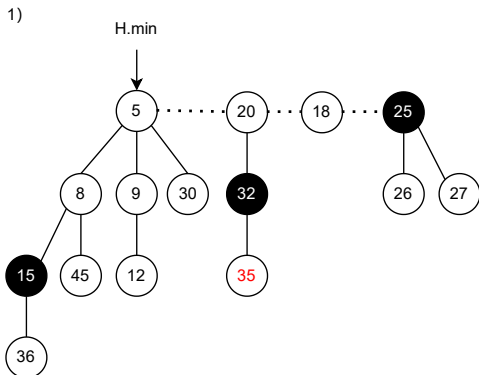


图 1: 斐波那契堆

1)



2)



1. 对于递归式 $T(n) = 2T(\frac{n}{2}) + \frac{n}{\log_2 n}$, 请给出渐近确界。

解:

$$T(n) = 2T(n/2) + n/\log n$$

由于 $n^{\log_2 2} = n$, 而 $f(n) = n/\log n$, 此时主定理不适用。考虑递归树分析:

$$\begin{aligned} T(n) &= 2T(n/2) + \frac{n}{\log n} \\ &= 4T(n/4) + \frac{n}{\log n} + \frac{n}{\log \frac{n}{2}} \\ &= n \sum_{i=0}^{\log n - 1} \frac{1}{\log \frac{n}{2^i}} \end{aligned}$$

假设存在整数 k 使得 $2^k \leq n < 2^{k+1}$ 。这意味着 $k \leq \log_2 n < k+1$, 由于 $\log_2 n \geq k$, 我们有 $\log_2 n - i \geq k - i$ 。我们来分析求和部分 $S(n) = \sum_{i=0}^{k-1} \frac{1}{\log_2 n - i}$ 的上界:

$$S(n) = \sum_{i=0}^{k-1} \frac{1}{\log_2 n - i} \leq \sum_{i=0}^{k-1} \frac{1}{k - i} = \sum_{j=1}^k \frac{1}{j} = H_k$$

其中 H_k 是调和级数, 我们知道 $H_k = \Theta(\log k)$ 。所以, $T(n) \leq n \cdot \Theta(\log k) = O(n \log k)$ 。而另一侧的下界同理, 可以得到 $T(n) = \Omega(n \log k)$ 。综合得到 $T(n) = \Theta(n \log k)$ 。

使用 Akra-Bazzi 方法也可以求出该解, 具体解法可参照课本。

2. 有一排共 n 桶染料，每桶燃料的颜色用一个 $[1, n]$ 之间的数字表示。画家小 S 想要使用至少 k 种颜色作画，但是他只能使用**连续的**若干桶染料（也就是若干桶相邻的染料）。现在请你求出：他最少需要用多少桶染料，才能够满足至少 k 种颜色的要求。给出伪代码，并说明算法的工作原理。你的算法时间复杂度需要是 $O(n)$ 的，超出会酌情扣分。

注意，你的算法不需要考虑无解的情况。

解：

考虑使用双指针 + 桶计数。我们使用 $left$ 和 $right$ 维护一个滑动的窗口，也就是一个连续子数组 $[left...right]$ ，同时用桶维护当前子数组中不同的颜色数量 $total$ 。我们的目标是最小化这个区间的长度，也就是 $right - left + 1$ 。

首先初始化 $left = 1, right = 0, total = 0, answer = n, cnt[] = 0$ ，然后每一步中：

- 不断向右扩展 $right$ ，直到当前窗口包含 k 种不同的颜色
 - $cnt[color[r]] \leftarrow cnt[color[r]] + 1$ ，如果此时 $cnt[color[r]]$ 从 0 变成 1，说明出现了一种新的颜色，那么 $total \leftarrow total + 1$
 - $r \leftarrow r + 1$ ，如果 $r > n$ ，则退出循环
- 更新答案 $answer \leftarrow \min(answer, right - left + 1)$
- $cnt[color[l]] \leftarrow cnt[color[l]] - 1$ ，如果此时 $cnt[color[l]]$ 从 1 变成 0，说明将当前窗口右移一位后，少出现了一种颜色，那么 $total \leftarrow total - 1$
- $left \leftarrow left + 1$ ，表示窗口的左端点向右移动一步。

Algorithm 1 最短连续 k 色子数组

Require: 颜色数组 $colors[1\dots n]$, 目标颜色数 k

```
1:  $n \leftarrow \text{length of } colors$ 
2:  $counts \leftarrow$  new array of size  $(n + 1)$  initialized to 0
3:  $left \leftarrow 1$ 
4:  $right \leftarrow 0$ 
5:  $total \leftarrow 0$ 
6:  $answer \leftarrow n$ 
7: while  $left \leq n$  do
8:   while  $total < k$  and  $right < n$  do
9:      $right \leftarrow right + 1$ 
10:     $c_{right} \leftarrow colors[right]$ 
11:    if  $counts[c_{right}] = 0$  then
12:       $total \leftarrow total + 1$ 
13:     $counts[c_{right}] \leftarrow counts[c_{right}] + 1$ 
14:  if  $total < k$  then
15:    break
16:   $answer \leftarrow \min(answer, right - left + 1)$ 
17:   $c_{left} \leftarrow colors[left]$ 
18:   $counts[c_{left}] \leftarrow counts[c_{left}] - 1$ 
19:  if  $counts[c_{left}] = 0$  then
20:     $total \leftarrow total - 1$ 
21:   $left \leftarrow left + 1$ 
22: return  $answer$ 
```

3. 已知在一个区间集合中，被最多数量区间覆盖的点被称为最大重叠点，且它一定可以是某个区间的端点。请你简述一个数据结构的设计思路，使它能够支持动态查询一个动态区间集合中的最大重叠点（任意一个符合要求的端点即可），并分析各操作的时间复杂度。

解：

利用扩展红黑树，维护区间集合内的区间端点。其中左端点权重为 +1，右端点权重为 -1，任意位置的覆盖数量可以表示为从最左端点到该位置的权重前缀和。最大重叠点即为整个序列中权重前缀和的最大值所对应的端点。

在红黑树中，每个节点存储以下三个辅助信息：

- 子树权重和 $\text{sum}(x)$ ：以该节点为根的子树中所有权重之和；
- 最大前缀和 $\text{max_prefix}(x)$ ：该子树内从最左端点出发可达到的最大前缀权重和；
- 最大前缀位置 $\text{argmax}(x)$ ：达到该最大前缀和的端点位置。

当红黑树结构发生变化（如插入、删除或旋转）时，需要在回溯过程中重新计算每个受影响节点的辅助信息，即：

首先计算该节点的子树权重和：等于左子树权重和 + 当前节点权重 + 右子树权重。

然后计算最大前缀和：取三种可能的情形中的最大者：

- 左子树内部的最大前缀和
- 经过当前节点（左子树权重和 + 当前节点权重）
- 跨过当前节点并延伸到右子树的最大前缀（左子树权重和 + 当前节点权重 + 右子树的最大前缀和）。

与最大前缀和对应的位置 $\text{argmax}(x)$ 同步更新为产生该最大值的端点。

因此，每当插入或删除一个区间的端点时，只需在回溯更新过程中依次执行上述“重新计算节点信息”的过程，复杂度为 $O(\log n)$

最终，红黑树根节点的 argmax 即为当前区间集合的最大重叠点，查询时间为 $O(1)$

4. 对于图一所给的斐波那契堆，黑色节点表示结点的 mark 属性为 true:
- 1) 抽取最小结点，画出操作后最后的结果（需要考虑 CONSOLIDATE(合并) 操作,）。
 - 2) 在 1) 的基础上，把关键字为 30 的节点的值降低为 4, 画出操作后的最后的结果（需考虑 CASCADING-CUT(级联切断) 操作）。

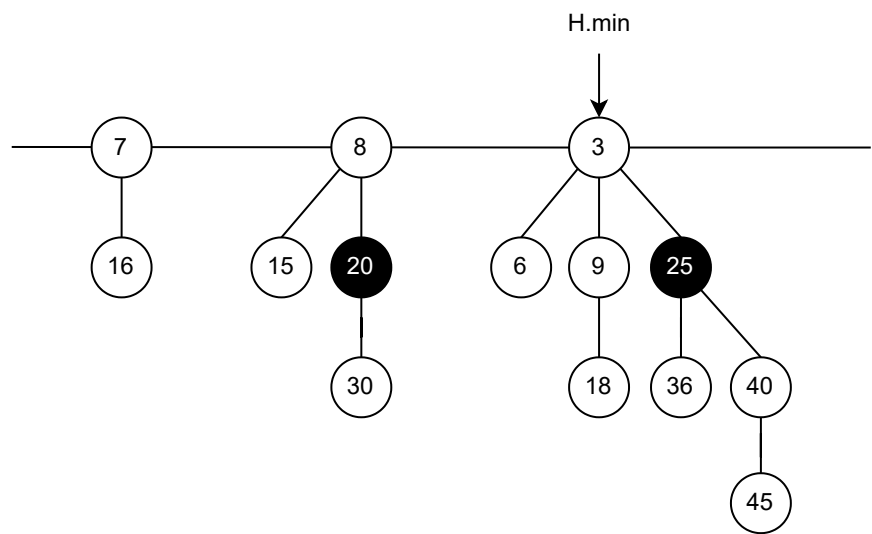
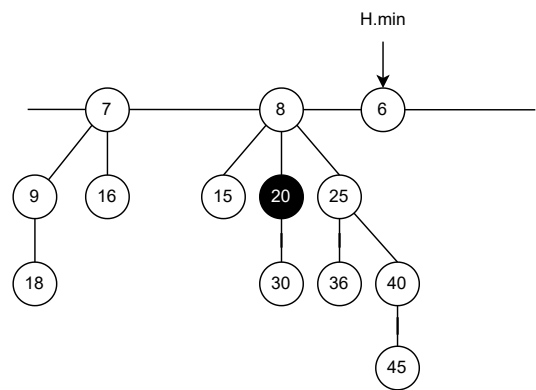


图 1: 斐波那契堆

1)



2)

