CSC265 Homework 7 | Fall 2021
Name: Haojun Qiu | Student Number: 1006900622
Who I discussed with: Darcy Wang, Runshi Yang

Notation: $h(H)$, $h(P)$ denotes height of any heap H any pennant P, respectively.

Q1

I will be using a fact: Suppose $P_0, \ldots, P_m$ is a pennant forest, then $P_0, \ldots, P_i$ is a pennant forest for $i \in \{0, \ldots, m\}$.
That is, we can get a pennant forest by cutting off a portion of pennants at the end of a pennant forest. This is true since that property 3 and 4 still holds for $P_0, \ldots, P_i$. And, the order of pennants does not change from original pennant forest so property 1 and 5 still hold. Property 2 hold since we do not add any pennant.

Lemma 1. *Let $m \in \mathbb{N}$. Suppose $P_0, \ldots, P_m$ is a pennant forest, then for all $i \in \{0, \ldots, m\}$, $h(P_i) \in \{i-1, i\}$.*

Proof We prove that statement by cases on value of m. The latter case use the result from former case.

Case 1: Assume $m = 0$. Assume that P_0 is a pennant forest. Let $i = 0$. Then, by property 3, there are at least 1 pennant of height at most 0. That is, $h(P_i) = h(P_0) = 0 = i \in \{i-1, i\}$.

Case 2: Assume $m = 1$. Assume that P_0, P_1 is a pennant forest. Since, P_0 itself forms a pennant forest, $h(P_0) = h(P_0) = 0 = i \in \{-1, 0\}$ is shown in case 1 already. Let $i = 1$. Then, by property 3 on pennant forest P_0, P_1, there are at least $i + 1 = 2$ pennants of height at most 1, so $h(P_1) \leq i = 1$. By definition of tree height, $h(P_i) \geq 0$, hence we have $h(P_1) \in \{0, 1\} = \{i-1, i\}$.

Case 3: Assume $m \geq 2$. Assume that $P_0, \ldots, P_m$ is a pennant forest. Since P_0, P_1 is a forest pennant, by case 2, we have that $h(P_i) \in \{i-1, i\}$ for $i \in \{0, 1\}$. Let $i \in \{2, \ldots, m\}$.

- Firstly, we show $h(P_i) \leq i$. Since $0 \leq 2 \leq i \leq m$, by property 3 on the pennant forest $P_0, P_1, \ldots, P_i$, we know that there are at least (all) $i + 1$ pennants in $P_0, P_1, \ldots, P_i$ of height at most i. In particular, $h(P_i) \leq i$, as needed.
- We show that $h(P_i) \geq i - 1$. Suppose for a contradiction that $h(P_i) < i - 1$, i.e., $h(P_i) \leq i - 2$. Since $h(P_i) \leq i - 2$, by property 1, we have that $h(P_k) \leq h(P_i) \leq i - 2$ for all $k \in \{0, 1, \ldots, i-1\}$. That is, all $i + 1$ pennants in $P_0, P_1, \ldots, P_i$ have height at most $i - 2$. However, since $0 \leq i - 2 \leq m$, by property 4, we know that there are at most $(i - 2) + 2 = i$ pennants of height at most $i - 2$ in the pennant forest $P_0, P_1, \ldots, P_i$. This is a contradiction, so we conclude that $h(P_i) \geq i - 1$, as wanted.

So, for all $i \in \{0, \ldots, m\}$, $h(P_i) \in \{i-1, i\}$. ■

Proof of Q1: Let $m \in \mathbb{N}$. Assume that $P_0, P_1, \ldots, P_m$ is a pennant forest. Then, by lemma 1, since $h(P_i) \in \{i-1, i\}$ for all $i \in \{0, \ldots, m\}$, combining with definition of pennant, $|P_i| \in \{2^{i-1}, 2^i\}$ for all $i \in \{0, \ldots, m\}$. Therefore, combining with the fact that $h(P_0) = 0$ shown in lemma 1 (case 1), we have $|P_0| = 1$ and

$$1 + \sum_{i=1}^{m} 2^{i-1} \;\leq\; |P_0| + \sum_{i=1}^{m} |P_i| \;=\; \sum_{i=0}^{m} |P_i| \;\leq\; \sum_{i=0}^{m} 2^i.$$

And since $\sum_{i=0}^{m} 2^{i-1} = 1 + \sum_{i'=0}^{m-1} 2^{i'} = 1 + \frac{2^m - 1}{2-1} = 2^m$, and $\sum_{i=0}^{m} 2^i = \frac{2^{m+1}-1}{2-1} = 2^{m+1} - 1$, we have

$$2^m \;\leq\; \sum_{i=0}^{m} |P_i| \;\leq\; 2^{m+1} - 1 \quad \Longrightarrow \quad 2^m \;\leq\; \sum_{i=0}^{m} |P_i| \;<\; 2^{m+1},$$

as wanted. ■

Q2

We solve the problem using recursive algorithm. Let us call the algorithm CONSTRUCT-PF(F, H), where H is an input max heap of height h. We give a specification of our recursive algorithm.

Specification: Suppose F is a (perhaps empty) pennant forest, and H is a max-heap of height h. Then CONSTRUCT-PF(F, H) returns a pennant forest $P_0, \ldots, P_h$ with the elements in F and H.

Algorithm:

CONSTRUCT-PF(F, H)

```
1  Compute the height h of input H by ⌊lg(H.heapsize)⌋, this is correct by exercise 6.1-2 in CLRS.
2  if h = 0                          // Base Case
3      Construct a pennant P with root only using node in H (i.e., H.root)
4      Insert P to the start of pennant sequence in F
5      return F
6  else (h > 0)                      // Recursion
7      depth-h-size = H.heapsize − (2^h − 1)
8      if depth-h-size ≤ 2^h/2
9          Let P be the pennant with root be H.root, and root's subtree be H.root.right
10         H.root = H.root.left
11         H.heapsize = H.heapsize − (2^(h−1) − 1 + 1).      // Update the heap H to be its left subtree
12         F' = CONSTRUCT-PF(F, H)
13         Insert P at the end of pennant sequence in F'
14         return F'
15     else (depth-h-size > 2^h/2)
16         Let P be the pennant with root be H.root, and root's subtree be H.root.left
17         H.root = H.root.right
18         H.heapsize = H.heapsize − (2^h − 1 + 1)           // Update the heap H to be its right subtree
19         F' = CONSTRUCT-PF(F, H)
20         Insert P at the end of pennant sequence in F'
21         return F'
```

Invocation: Notice that if the specification holds, the call CONSTRUCT-PF(F, H) with H being the max-heap of height h and F being a empty pennant forest, will gives us the desired output.

Proof of specification: We prove it by induction on height h of input H. Since our initial invocation will input an empty pennant forest (empty sequence/list), it suffices to assume F is empty in prior to the algorithm.

Base case: If $h = 0$. Then, observe that since we expand F after the recursion call in the recursion step, so at the time of reaching base case, F will be the same as original input, i.e., still empty. Hence, adding a pennant of height 0 to it, we return a pennant forest P_0 with elements in F and H, as wanted.

Induction Step: If $h > 0$. We enter the recursion step (starts from line 6). Suppose F is a pennant forest and H is a max heap of height h. Then, assume the specification holds on heap with height $h - 1$.

The input H has height decrease by 1 (becomes $h-1$) in both recursion step (as complete binary tree H is being updated to its left or right subtree in lines 10 & 11 or 17 & 18). The heapsize attribute is updated correctly in lines 11 and 18 (in case 1 H lost its root plus a perfect binary tree of height $h - 2$, in case 2 H lost its root plus a perfect binary tree of height $h - 1$). Therefore, H is still a max-heap since subtree of a max heap is still a max-heap, so the induction hypothesis holds for the recursive call. That is, F' in line 12 and 19, is a pennant forest $P_0, \ldots, P_{h-1}$ with the elements in F and H.

Case 1: *depth-h-size* $\leq 2^h/2$. This means that the heap H is at most half full at the last level (depth h). We extract the root and right subtree (which is perfect binary tree) to be a new pennant (line 9), and put it to the end of F', and F' gets returned. We denote P in line 9 as P_h in resulting F'. Notice that $h(P_h) = h - 1$. We show that resulting F' satisfies all 5 properties of pennant forest.

- Since F' in line 12 is a pennant forest $P_0, \ldots, P_{h-1}$, by lemma 1, $h(P_{h-1}) \in \{h - 2, h - 1\}$. Hence, $h(P_h) = h - 1 \geq h(P_{h-1})$. Together with property 1 in original F', property 1 holds on the resulting F'.
- In F' in line 12 ($P_0, \ldots, P_{h-1}$), by lemma 1, the only pennant could have height $h-1$ is P_{h-1}, so there are at most 2 pennants with height $h - 1$ (P_{h-1} and P_h) in resulting F', together with property 2 on original

F', property 2 holds on resulting F'.

- Since we add a pennant to F' and property 4 is a lower bound on number of pennants, we remains to show statement with $i = h$. By property 3 on original F', there are at least h pennants of height at most $h-1$ in $P_0, \ldots, P_{h-1}$, hence there are at least $h+1$ pennants of height at most h (since $h(P_h) = h-1$) in $P_0, \ldots, P_{h-1}, P_h$, i.e., resulting F'.
- We add a pennant of height $h-1$, so we only need to show the statement with $i = h-1$ and $i = h$. There are exactly (thus at most) $(h-1)+2 = h+1$ pennants of height at most $h-1$ in resulting F'. It follows that there are at most $h+2$ pennants of height at most h as well.
- By the property 5 of original F', we remains to show that the root of $P_{h-1} \leq P_h$. This directly follows from the max heap property since root of P_{h-1} is in the left subtree of the root of P_h (root of original H), which all originally in a max-heap H before updating.

Case 2: *depth-h-size* $< 2^h/2$. This means that the heap H is more than half full at the last level (depth h). We extract the root and left subtree (which is perfect binary tree) to be a new pennant, and put it to the end of F', and F' gets returned. We denote P in line 9 as P_h in resulting F'. And, $h(P_h) = h$ in this case. Similarly, we show 5 properties of pennant forest on resulting F'.

- Replace $h(P_h) = h-1$ to be $h(P_h) = h$ in the argument in case 1 - property 1.
- Similarly, in F' in line 12 ($P_0, \ldots, P_{h-1}$), by lemma 1, the no pennant have height h. So, only $1 \leq 2$ pennants with height h (P_h) in resulting F', together with property 2 on original F', property 2 holds on resulting F'.
- Replace $h(P_h) = h$ to be $h(P_h) = h$ in the argument in case 1 - property 3.
- We add a pennant of height h, so we only need to show the statement with $i = h$. There are exactly $h+1$ pennants (thus at most $h+2$) of height at most h in resulting F'.
- Replace the word "left" to be "right" in the argument in case 1 - property 5.

■

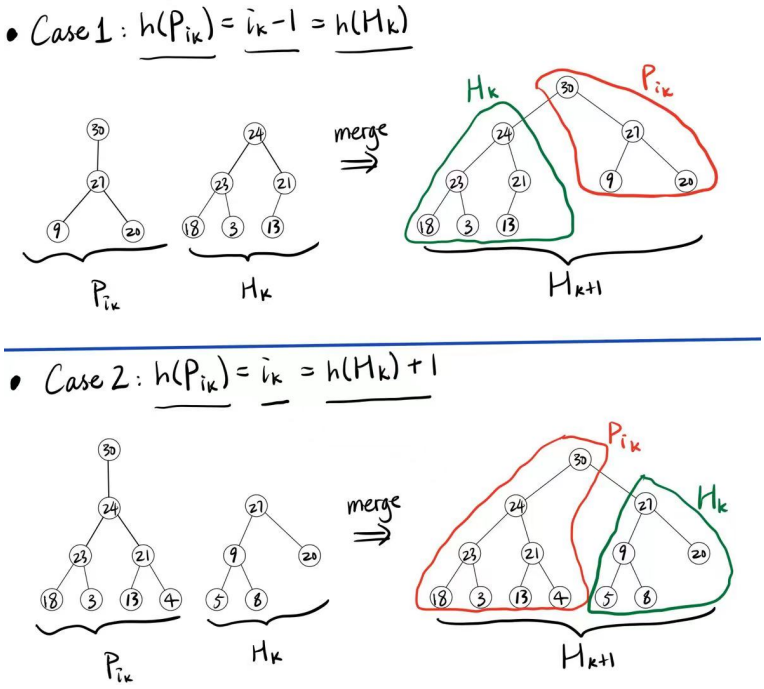
Running Time Analysis: As stated in the proof, the height of heap H is reduced by 1 in every recursive call, and we stops when it reaches base case of height being 0. So, there are $O(h)$ recursive calls. In every recursive call, only constant time operation being performed (calculating heapsize, re-set attributes, insertion into a sequence - assuming we use circular linked list to implement pennant forest so insertion at the end of sequence takes constant time). Hence, the running time of algorithm is $O(h)$.

Q3

The idea is to iteratively merge $P_0, \ldots, P_m$ in order.
Algorithm:

Set $m =$ the number of pennants in the pennant forest, determined by traversing through the sequence.
Let H to be a heap with root only constructed from P_0.
Let $i = 1$.
While $i \leq m$ **do**
  Determine the height $h(P_i)$ of pennant P_i by traversing a path (e.g., leftmost path) of P_i.
  If $h(P_i) = i-1$, then
    merge H and P_i by setting $H.root$ to be the root of P_i and left subtree of H to be original H and right subtree of H to be subtree rooted at P_i's child
  Else $(h(P_i) = i)$
    merge H and P_i by setting $H.root$ to be the root of P_i and right subtree of H to be original H and left subtree of H to be subtree rooted at P_i's child
  $i = i + 1$
Return H

Below is a picture illustration of the merging procedure in the loop, on 2 different cases of $h(P_{i_k})$.



Loop Invariant: Suppose $P_0, \ldots, P_m$ is a pennatn forest. For all $k \in \mathbb{N}$, if the loop is executed at least k times, H_k is a max-heap of height $i_k - 1$ with the same elements in $P_0, \ldots, P_{i_k - 1}$.

Proof of Loop Invariant: We prove by induction on m.

Base Case: Let $k = 0$. In prior to any loop iteration, $H_k = H_0$ is a max heap of height 0 with elements in P_0 (actually is just P_0), as wanted.

Induction Step: Let $k \in \mathbb{N}, k > 0$. Assume the loop invariant holds for k. That is, if the loop is executed at least k times, then H_k is a max heap of height $i_k - 1$ with the same elements in $P_0, \ldots, P_{i_k - 1}$. Assume the loop is executed at least $k+1$ times, I will show that H_{k+1} is a max heap of height $i_{k+1} - 1$ with the same elements in $P_0, \ldots, P_{i_{k+1} - 1}$. By lemma 1, There are only two cases on height of P_{i_k} ($i_k - 1$ or i_k).

Case 1: $h(P_{i_k}) = i_k - 1$. By merging procedure stated in the if-branch, H will be expanded so that H_{k+1} has H_k as left subtree which has height $i_k - 1$, and P_{i_k}'s subtree as right subtree which has height $i_k - 2$. Then, the height of H is increased by 1, i.e., $h(H_{k+1}) = i_k - 1 + 1 = i_{k+1} - 1$.

It remains to show the resulting H_{k+1} is a max-heap. Firstly, every nodes in H's right and left subtree satisfies the max-heap property (since left subtree is a max-heap and right subtree is the subtree of a pennant). And, the max-heap property holds on root of H_{k+1} as well, since H_{k+1}'s root is greater than or equal its left child (H_k) by property 5 and max-heap property on pennants, and is greater than equal to its right child because it was originally child of $H_{k+1}.root$ in pennant P_{k+1}. H is a complete binary tree since H_{k+1} is full at depth $0, 1, \ldots, i_k - 1$ (since P_{k+1} has perfect binary subtree and H_k is complete binary tree) and nodes at the last level comes from complete binary tree H_k and being put on the left side, so as left as possible. H_{k+1} contains elements in H_{k-1} and $P_{i_{k+1}-1}(P_{i_k})$, that is $P_0, P_1, \ldots, P_{i_k}$.

Case 2: $h(P_{i_k}) = i_k$. The formal argument is similar as case 1. As illustrated in the figure above, the difference is that in this case, the higher pennant's perfect binary tree is put on the left of the root, so resulting H (i.e., H_{k+1}) will be over half full at the last level. It will still be a max-heap, of height increased by 1 from H_k. ■

Correctness: The loop stops when $i_k = m + 1$ since i increase by 1 at each iteration. That is, $i_k - 1 = m$. So, by the loop invariant, we know that H being returned is a max-heap of height $i_k - 1 = m$ with the same elements in $P_0, \ldots, P_m$, as wanted.

Running Time Analysis: Determine the length of sequence takes $O(m)$ time. Construct H from P_0 takes constant time. For the while loop, it runs m iteration, at each iteration $i \in \{1, \ldots, m\}$ it determines the height of P_i, which takes $a+1$ steps where $a \in \{i-1, i\}$ by lemma 1, and do some constant operations. So, at iteration

i, it takes $O(i)$ time. Therefore, at total the loop takes times in

$$\sum_{i=1}^{m} O(i) = O\Big(\sum_{i=1}^{m}(i)\Big) = O(m^2).$$

The return statement takes constant time, so entire algorithm runs in $O(m) + O(m^2) + O(1) = O(m^2)$.

Q4

The idea is two make the P's root and it child's left subtree first pennant, and make the child of P_h' root and whose right subtree the second pennant.

Algorithm:

If $h(P) = 0$, then raise an error. We can not split it to two pennants.
Else $(h(P) > 0)$, then
Construct two pennants P_1 and P_2 containing root only, where $P_1.root$ contains element in $P.root$ and $P_2.root$ contains element in $P.root.child$.
$P_1.root.child = P.root.child.left$
$P_2.root.child = P.root.child.right$

Correctness: P_1 and P_2 root's subtrees are subtrees of perfect binary tree rooted at $P.root$, hence perfect as well. Also, the max-heap property holds on $P_1.root$ and $P_2.root$ since their child were under them when they were in P, which satisfies max-heap property globally. The other nodes has max-heap property holds as well (since it held when they in P and structure under those nodes do not change). Both P_1 and P_2 lost one level/depth at the top, so their height decrease by 1 from P_h, hence $h(P_1) = h(P_2) = h(P) - 1 = h - 1$, as wanted.

Running Time Analysis: All operation are constant time (when determine height we do not need to get the full height of P but only need to know it is 0 or not). So, it takes $O(1)$ to perform the algorithm.

Q5

Suppose P_1, P_2 are input pennants of height $h - 1$.

Algorithm:

1 Construct an empty pennant P.
2 Determine which of $P_1.key$, $P_2.key$ is larger, set the larger one to be the element in root of P (if tie just set any one to be). And the other is set to be the element in $P.root.child$.
3 Then, set $P.root.child.left = P_1.child$, and $P.root.child.right = P_2.child$.
4 Then, call Bubble-Down($P.root.child$) to restore max-heap property.

The helper procedure Bubble-Down(x) is indeed max heap Max-Heapify (CLRS 6.2) in the context of tree implementation instead of array, where input x is a node in a binary tree.

That is, suppose subtrees rooted at $x.left$ and $x.right$ are two max-heap, then after call Bubble-Down(x) the subtree rooted at x is a max-heap.

Let y points to the node we currently fix up on, So, $y = x$ initially. We update y along the way. The implementation will be iterative, at every iteration, if y has no child, then just return. If y has at least one child, determine the max element of y and its child(ren), if y contains the max element, then return. Else, exchange the element in y with the largest element (can be element in $y.left$ or $y.right$), and update y points to where original element in y goes to ($y.left$ or $y.right$).

Correctness: Firstly, right after merging P_1 and P_2 into P, P is has the pennant shape that has height 1 greater than P_1 and P_2. And, max heap property is preserved on the subtrees rooted at $P.child.left$ and $P.child.right$, so after call Bubble-Down($P.root.child$) (this call is neccssary since right after mergin max-heap property may not hold on node $P.root.child$), subtree rooted at $P.root.child$ satisfies max-heap property. The max heap property holds at root since it is the larger of $P_1.root$ and $P_2.root$ thus largest element in the P.

Running Time Analysis: The merging step takes constant time. BUBBLE-DOWN(x) takes time at most proportional to height of the subtree rooted at x. Thus, BUBBLE-DOWN($P.root.child$) use $O(h)$. Together, the algorithm runs in $O(h)$.

Q6

Algorithm:

Step 1: We adopt the algorithm from Q3, with one line added inside the loop (before the line $i = i+1$), that is, BUBBLE-DOWN($H.root$) (a helper algorithm from Q5). Let us call the modified algorithm CONSTRUCT-HEAP($P_0, \ldots, P_m$) where $P_0, \ldots, P_m$ is the sequence of pennants satisfies the first four properties of a pennatn forest. And, let H points to the output.

Step 2: Then, we construct a empty pennant forest (empty sequence) F. And we use the output H from above, together as input to algorithm in Q2, that is, call CONSTRUCT-PF(F, H).

Correctness: Notice that in our proof of loop invariant in Q3, we use property 5 of pennant forest to conclude that resulting heap at the end of each iteration is still a heap. Therefore, even if our input sequence of pennants does not satisfies property 5, if we can show that H at every iteration is still a heap (in our modified algorithm), the loop invariant still holds, and thus we can still conclude that the output H is a max-heap of height m with the same elements in $P_0, \ldots, P_m$. Below is the proof to H points to a max-heap at every iteration.

Proof: Initially, H is contains only its root (element in P_0) hence a max-heap.

Consider an arbitrary iteration, assume H before the iteration is a max-heap. Then, since in either case, we have shown in Q3 that after merging procedure, H has heap shape, and that subtrees rooted at $H.root.left$ and $H.root.right$ are both max-heap (without using property 5). Therefore, althought the max-heap property may not holds on the new root of H due to lost of property 5, after call BUBBLE-DOWN($H.root$), the resulting H is still a max-heap, as wanted. ■

Therefore, knowing output H from step 1 is a max-heap of height m and F is constructed to be a empty sequence, calling CONSTRUCT-PF(F, H) in step 2, by its specification, outputs a pennant forest $P_0, \ldots, P_m$ with the elements in F and H (i.e, the elements in H since F is empty, i.e., the elements in original input sequence $P_0, \ldots, P_m$ of pennants), as wanted.

Running Time Analysis: Step 1 of the algorithm, calling modified version of Q3 does not change its asymptotic upper bound of runtime. Since, at iteration i, we now have one more procedure BUBBLE-DOWN($H.root$), which takes $O(h(H)) = O(h(P_i)) = O(i)$. Therefore, at total the loop still takes $O(m^2)$, so does the entire algortihm.

Step 2 takes $O(h) = O(m)$ since height of input heap is m. Therefore, together with some constant operations, runtime of entire algorithm for Q6 is in $O(m^2) + O(m) + O(1) = O(m^2)$, as wanted.