CS101 Algorithms and Data Structures
Fall 2022
Homework 5

Due date: 23:59, October 23th, 2022

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your FULL name to your Chinese name and your STUDENT ID correctly in Account Settings.
4. If you want to submit a handwritten version, scan it clearly. `CamScanner` is recommended.
5. When submitting, match your solutions to the problems correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero points.

1. **(8 points) Multiple Choices**

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 1 point if you select a non-empty subset of the correct answers.

Write your answers in the following table.

(a)	(b)	(c)	(d)
BC	AB	B	C

(a) (2') Which of the following is/are true about Trees?

A. A full binary tree with n nodes has height of $O(\ln n)$.

B. Only given the depth of all leaf nodes in a tree, we can infer the height of this tree.

C. If T was transformed from a general tree T' through Knuth transform, which means T is a left-child right-sibling binary tree. Then the post-order traversal of T' identical to the in-order traversal of T.

D. None of the above.

(b) (2') Let's look at some magic properties about trees, which of the following statements is/are true?

A. A tree is a full binary tree if and only if every node has an odd number of descendants.

B. A rooted binary tree has the property that the number of leaf nodes equals to the number of full nodes plus 1.

C. There are 133 distinct shapes of ordered binary trees with 6 nodes.

D. None of the above.

Solution:
A. It is mentioned that each node has odd number of descendants including node itself, so all nodes must have even number of descendants 0, 2, 4, and so on. It means each node should have either 0 or 2 children. So this tree will be a full binary tree.
C. The total number of binary search trees with n nodes is:

$$T(n) = \sum_{i-1}^{n} = T(i-1)T(n-i) = \frac{(2n)!}{(n+1)!n!}$$

To compute the number of binary trees with n nodes, we can take one node as the root and discuss the number of nodes assigned to left-subtree and right-subtree. E.g.:

$$T(1) = 1, T(2) = T(0)T(1) + T(1)T(0) = 2, T(3) = T(0)T(2) + T(1)T(1) + T(2)T(0) = 5$$

$$T(4) = T(0)T(3) + T(1)T(2) + T(2)T(1) + T(3)T(0) = 14, T(5) = 42, T(6) = 132$$

(c) (2') Which of the following statements about the binary heap is/are true? Note that binary heaps mentioned in this problem are implemented by complete binary trees.

A. If a binary tree is a min-heap, then the post-order traversal of this tree is a descending sequence.

B. We have a binary heap of n elements and wish to add n more elements into it while maintaining the heap property. It can be done in $O(n)$.

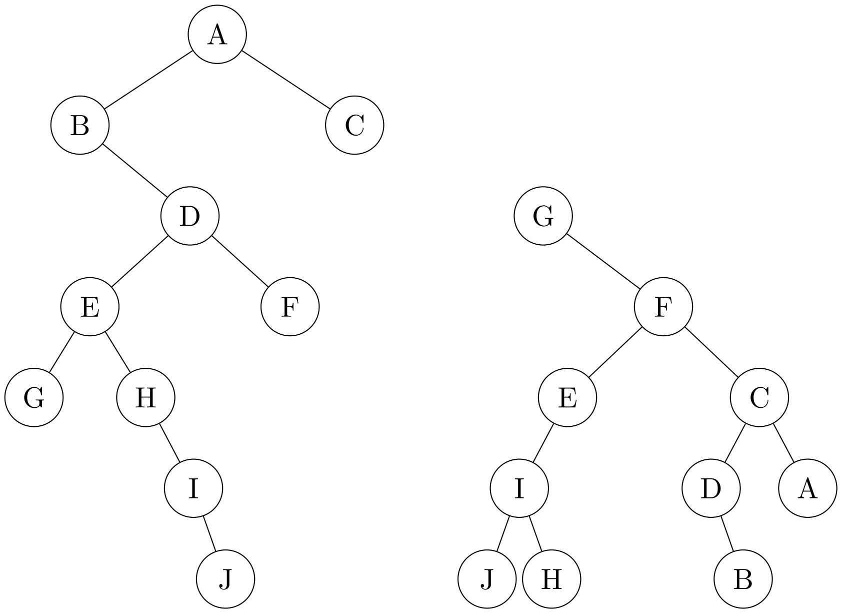
C. There exists a heap with seven distinct elements so that the in-order traversal gives the element in sorted order.

D. If item A is an ancestor of item B in a heap, then it must be the case that the Push operation for item A occurred before the Push operation for item B.

E. None of the above.

Solution:
C. The in-order traversal will not give elements in sorted order. As heap is implemented as either min-heap or max-heap, the root will have highest or lowest value than remaining values of the nodes. So this traversal will not give a sorted list.

(d) (2') Which traversals of the left tree and right tree, will produce the same sequence node name?



A. left: Post-order, right: Post-order

B. left: Pre-order, right: Pre-order

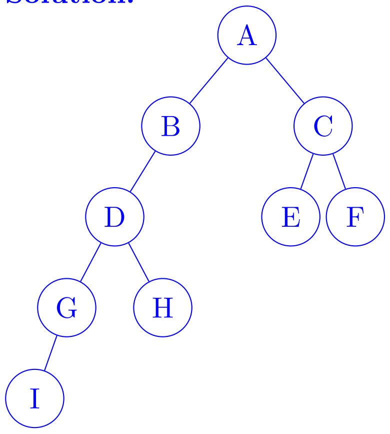
C. left: Post-order, right: In-order

D. left: In-order, right: In-order

2. (8 points) Draw a binary tree

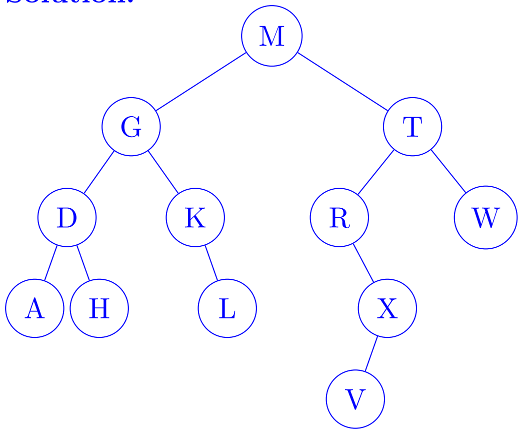
(a) (3') Given the in-order and pre-order traversal of a binary tree T are IGDHBAECF and ABDGIHCEF respectively.
Draw the tree T.

Solution:



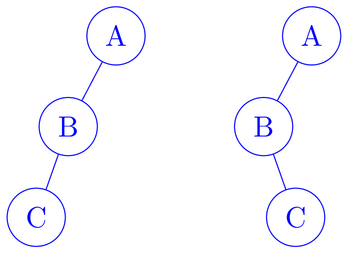
(b) (3') Given the in-order and post-order traversal of a binary tree T are ADHGKLMRUXTW and AHDLKGUXRWTM respectively.
Draw the tree T.

Solution:



(c) (2') Given the pre-order and post-order traversal of a binary tree T, can you decide the tree T? If yes, please describe an algorithm to construct T; if no, please provide a counterexample.

Solution:
No.


A
B
C
A
B
C

3. (12 points) Heap

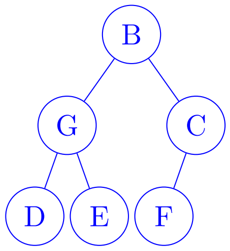
We are given the following array representing a min-heap where each letter represents a unique number. Assume the root of the min-heap is at index zero, i.e. A is the root. Note that there is no significance of the alphabetical ordering, i.e. we do not know whether B is less than or greater than C just because B precedes C in the alphabet.
Array: [A, B, C, D, E, F, G]
Six unknown operations are then executed on the min-heap. An operation is either a pop or a push. The resulting state of the min-heap is shown below
Array: [A, G, B, X, E, F, C]

(a) (8') The first and the final operations have already been filled in for you. Fill in the space below using the following options.
Note: You are free to use each of the following options 0 time, once or more than once.
(A) pop()
(B) push(B)
(C) push(C)
(D) push(X)
Hint: The number of elements in the heap doesn't change after 6 operations; In the final heap, X's position is in the left sub-heap.

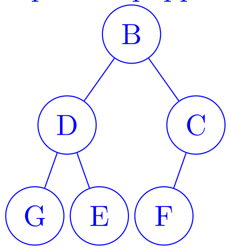
1. pop()
2. D push(X)
3. A pop()
4. A pop()
5. B push(B)
6. push(A)

(b) (4') Fill in the following comparisons with either >, <, or ? if unknown. We recommend considering which elements were compared to reach the final array.
1. X > B
2. X > C
3. G ? F
4. G > D

Solution:
1. Since the number of elements doesn't change, we must have two pops and two pushs. Since x is new in the final heap, we must have push(x). Since we only have two pushing options B and C, so the pop must be B or C.
2. First, after popping A, G becomes the top and needs to be percolated down. Since in the end G is in the left sub tree, it must be bigger than B so that it can be percolated down to the left sub tree, making B the top and now we get the tree:



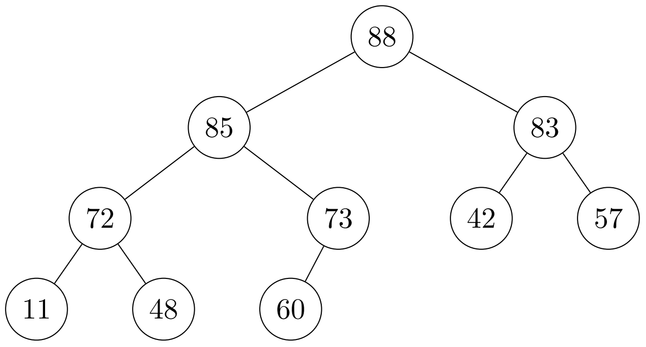
Since D needs to be popped, then G must be bigger than D so that it can later become the top to be popped. So now we get:



Since D needs to be popped, B must be popped first, and then pop D, and then push(B). Since in the end X is in the left sub tree, pop() must be taken after push(X) so that X can be percolated down in the left sub tree.
If we pop now, F will be the top. Since in the end it is in the right sub tree, C will become the top to make F percolated down in the right. In this case, D can't be popped. So the first step can only be push(X). The following steps are pop(),pop(),push(B).

4. (10 points) Heap Sort

(a) (10') You are given such a max heap like this:



Then you need to use array method to show each step of heap sort in increasing order. Fill in the value in the table below. Notice that the value we have put is the step of each value sorted successfully. For each step, you should always make you heap satisfies the requirement of max heap property.

index	1	2	3	4	5	6	7	8	9	10
value	88	85	83	72	73	42	57	11	48	60

Table 1: The original array to represent max heap.

index	1	2	3	4	5	6	7	8	9	10
value	85	73	83	72	60	42	57	11	48	88

Table 2: First value is successfully sorted.

index	1	2	3	4	5	6	7	8	9	10
value	83	73	57	72	60	42	48	11	85	88

Table 3: Second value is successfully sorted.

index	1	2	3	4	5	6	7	8	9	10
value	73	72	57	11	60	42	48	83	85	88

Table 4: Third value is successfully sorted.

index	1	2	3	4	5	6	7	8	9	10
value	72	60	57	11	48	42	73	83	85	88

Table 5: Fourth value is successfully sorted.

index	1	2	3	4	5	6	7	8	9	10
value	60	48	57	11	42	72	73	83	85	88

Table 6: Fifth value is successfully sorted.

index	1	2	3	4	5	6	7	8	9	10
value	57	48	42	11	60	72	73	83	85	88

Table 7: Sixth value is successfully sorted.

index	1	2	3	4	5	6	7	8	9	10
value	48	11	42	57	60	72	73	83	85	88

Table 8: Seventh value is successfully sorted.

index	1	2	3	4	5	6	7	8	9	10
value	42	11	48	57	60	72	73	83	85	88

Table 9: Eighth value is successfully sorted.

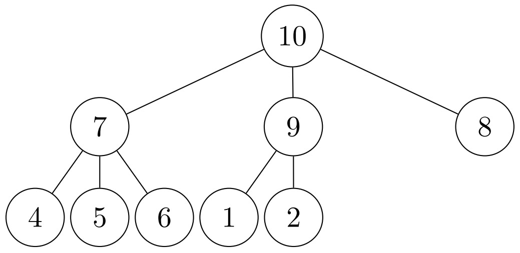
index	1	2	3	4	5	6	7	8	9	10
value	11	42	48	57	60	72	73	83	85	88

Table 10: Last 2 values are successfully sorted.

5. (12 points) k-ary Heap

In class, our professor has mentioned the method of the array storage of a binary heap. In order to have a better view of heap, we decide to extend the idea of a binary heap to a k-ary heap. In other words, each node in the heap now has at most k children instead of just two, which is stored in a complete binary tree.

For example, the following heap is a 3-heap.



(a) (4') If you are given the node with index i, what is the index of its parent and its jth child $(1 \leq j \leq k)$?

Notice: We assume the root is kept in $A[1]$. For your final answer, please represent it in terms of k, i and j. Please use flooring or ceiling to ensure that your final answer is as tight as possible if you need, i.e. $\lfloor m \rfloor$, $\lceil m \rceil$.

Solution:
A i-ary heap can be represented in a 1-dimensional array as follows. The root is kept in $A[1]$, its k children are kept in order in $A[2]$ through $A[k+1]$, their children are kept in order in $A[k+2]$ through $A[k^2+k+1]$, and so on. Two procedures that map a node with index i to its parent and to its jth child (for $1 \leq j \leq k$), respectively, are:
Parent: index $\lfloor (i-2)/k+1 \rfloor$
jth child: index $k(i-1)+j+1$

(b) (4') What is the height of a k-ary heap of n elements? Please show your steps.

Notice: For your final answer, please represent it in terms of i and n. Please use flooring or ceiling to ensure that your final answer is as tight as possible if you need, i.e. $\lfloor m \rfloor$, $\lceil m \rceil$.

Solution:

A k-ary heap would have a height of $\Theta(\log_k n)$. We know that:

$$\begin{aligned} 1 + k + k^2 + \cdots + k^{h-1} &< n \leq 1 + k + k^2 + \cdots + k^h \\ \frac{k^h - 1}{k - 1} &< n \leq \frac{k^{h+1} - 1}{k - 1} \\ k^h &< n(k-1) + 1 \leq k^{h+1} \\ h &< \log_k(n(k-1)+1) - 1 \leq h + 1 \end{aligned}$$

Therefore, $h = \lceil \log_k(n(k-1)+1) - 1 \rceil = \Theta(\log_k n)$.

(c) (4') Now we want to study the complexity of operations on k-ary heap. Suppose the Push, Pop and $\mathsf{Heap-sort}$ operations execute the same as what we mentioned in our lectures. Denote the height of the heap as h, please analyse the worst-case time complexity of Push and Pop operations on k-ary heap in terms of n, h, k with tight bound. For heap-sort, given a built heap, explain why the worst-case number of comparisons is $\Theta(nhk)$.

Solution:

For Push operation, one new element is pushed to the end of the heap and we compare it with its parent node to swap iteratively. The worst-case number of comparisons is the height of the heap, i.e., the worst-case time complexity is $\Theta(h)$.

For Pop operation, we delete the root element and copy the last element to the root. Then we compare it with its k children to swap iteratively. For the worst-case, the new root will be permeated from the root to the leaf and compare with each children. The worst-case time complexity is $\Theta(hk)$.

For the worst case of heap-sort, each element will be permeated from the root to the leaf. For each level, the inserted element needs to be compared to k children, and there are n elements in total. Therefore, the worst-case number of comparisons is $\Theta(nhk)$.