CS101 Algorithms and Data Structures

Fall 2019

Homework 4

Due date: 23:59, October 20, 2019

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 according problem numbers correctly.
6. No late submission will be accepted.
7. Violations to any of above may result in zero score.

Problem 1: Binary Tree & Heap

Binary Tree and Heap have a ton of uses and fun properties. To get you warmed up with them, try working through the following problems.

Multiple Choices: Each question has one or more correct answer(s). Select all the correct answer(s). For each question, you get 0 point if you select one or more wrong answers, but you get 0.5 point if you select a non-empty subset of the correct answers.

Note that you should write you answers of Problem 1 in the table below.

Q(1)	Q(2)	Q(3)	Q(4)	Q(5)
BCD	B	C	AB	120, 140, 90, 80 50, 70, 100, 60, 40

(1) Which of the following statements about the binary tree is true?

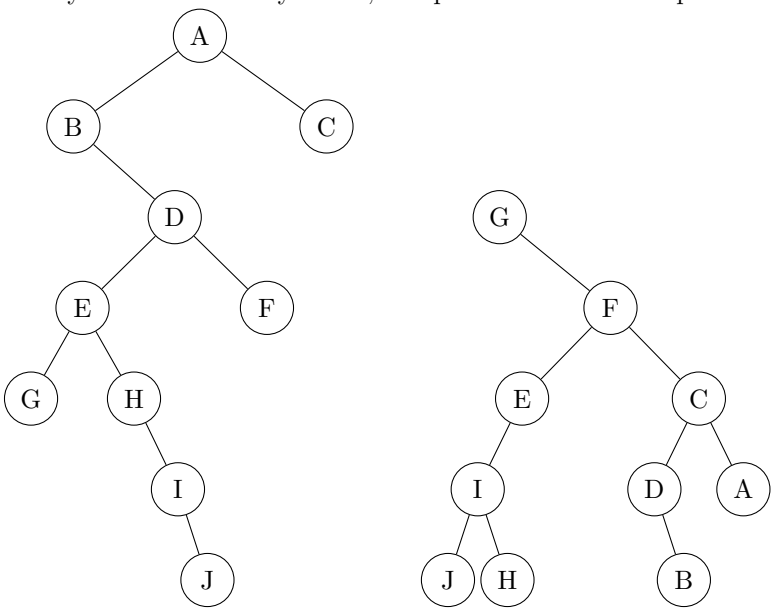
A. Every binary tree has at least one node.

B. Every non-empty tree has exactly one root node.

C. Every node has at most two children.

D. Every non-root node has exactly one parent.

(2) Which traversals of binary tree 1 and binary tree 2, will produce the same sequence node name?



A. Postorder, Postorder

B. Postorder, Inorder

C. Inorder, Inorder

D. Preorder, Preorder

(3) Which of the following statements about the binary tree is **not** true?

A. A rooted binary tree has the property that $n_0 = n_2 + 1$, where n_i denotes the number of nodes with i degrees.

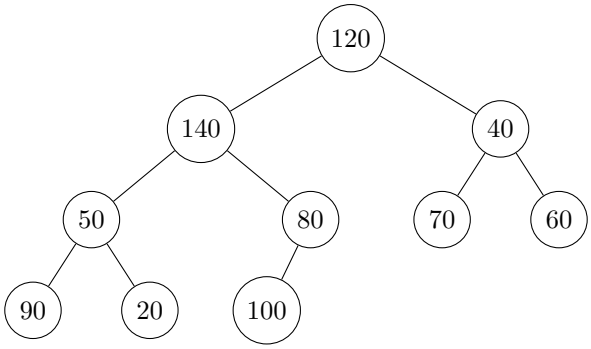
B. Post-order traverse can give the same output sequence as a BFS.

C. BFS and DFS on a binary tree always give different traversal sequences.

D. None of the above.

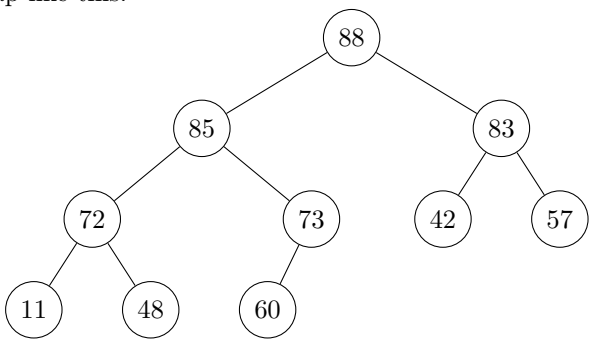
(4) Which of the following statements about the binary heap is **not** true?
A. There exists a heap with seven distinct elements so that the in-order traversal gives the element in sorted order.
B. If item A is an ancestor of item B in a heap (used as a Priority Queue) then it must be the case that the Insert operation for item A occurred before the `Insert` operation for item B.
C. If array A is sorted from smallest to largest then A (excluding A[0]) corresponds to a min-heap.
D. None of the above.

(5) Suppose we construct a min-heap from the following initial heap by Floyd's method. After the construction is completed, we delete the root from the heap. What will be the post-order traversal of the heap? Write down your answer in the table above directly.



Problem 2: Heap Sort

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	0	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	0	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	0	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	0	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	0	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	0	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	0	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	0	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	0	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	0	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.

Problem 3: Median Produce 101

Nowadays the hotest variety show *Produce 101* has a new rule to judge all the singers: for all 5 judgers, use the median score value among all the judgers to set her score. Previously, the programme group has a calculator to calculate the score for each singer. Accidentally, the calculator is broken one day. And the fans of famous star Yang Chaoyue are eagerly waiting for the score. So they want to help the programme group to get the correct score.

Recall that the median of a set is the value that separates the higher half of set's values from the set's lower values.

For example, given the set with **odd** numbers of elements:

$$\{78, 94, 17, 87, 65\}$$

The median score is 78.

For another example, given the set with **even** numbers of elements:

$$\{78, 94, 17, 87, 65, 76\}$$

The median score is $(78 + 76)/2 = 77$.

In Yang's fans group, the crazy fans have quarrelled for the following two opinions to get this median score:

- Use only one min heap.
- Use both min heap and max heap.

Now they are asking you for your help. Please help them solve this problem.

So first, let's try the case with only one min heap to get the median score.

Consider a set S of arbitrary and distinct integer scores (not necessatily the set shown above). Let n denote the size of set S, and assume in this whole problem that n can be odd or even. **Assume that we have inserted all the elements in set S to the minheap.**

(1) **Using natural language**, describe how to implement the algorithm that returns the median from set S. Analyze your time complexity. (Suppose the total number of element in S is given, which is n. And the minheap has been built.)

You will receive full credit only if your method runs in $O(n\log n)$ time.

Solution

You are free to use any methods in the API for `MinHeap`:

1. `insert(int n)` // Where you insert the integer n.
2. `int extractMin()` // Where you can get the minimum value among the heap and delete the value to maintain to heap property. The return value is an integer.
3. `bool isEmpty()` // Check whether the heap is empty. If it is empty, return bool value `true`. Otherwise, return bool value `false`.

```
1  int findMedian(MinHeap heap, int n){
2      if (n % 2 == 1){
3          for (i == 1; i <= (n+1)/2 ; i++){
4              int median = extractMin();
5          }
6      }else{
7          for (i == 1; i <=n/2 + 1; i++){
8              if (i == n/2): int median1 = extractMin();
9              if (i == n/2+1): int median2 = extractMin();
10         }
11         median = median1 + median2;
12     }
13     return median;
14 }
```

Because for each `extractMin()`, the time complexity is $O(\log n)$. The total time is:

$$\frac{n+1}{2}\log n = O(n\log n).$$

And now, let's try the case with both max heap and min heap to get the median score.

Now, another fancy fan Wang Xiaoming finds a data structure for storing a set S of numbers, supporting the following operations:

- `INSERT(x):` Add a new given number x to S
- `MEDIAN():` Return a median of S.

Assume no duplicates are added to S. He proposes that he can use a maxheap A and a minheap B to get the median score easily. These two heaps need to always satisfy the following two properties:

- Every element in A is smaller than every element in B.
- The size of A equals the size of B, or is one less.

To return a median of S, he proposes to return the minimum element of B. **Assume that we have inserted all the elements in set S to the two heaps.**

(2) Using two properties above, argue that this is correct, partially correct or wrong (i.e., that a median is returned). If it is not correct, can we find a strategy that calculates the median in $O(1)$ time complexity? Explain the reason and strategy (if we need) briefly. (Suppose the total number of element in S is given, which is n. And heap A, B have been built.)

Solution

Let n be the size of S.

By property 1, we know that the minimum element e of B is larger than all the elements in A. Furthermore, by its minimality, e is smaller than all the other elements in B.

By property 2, we know that A has to have exactly $\lceil (n+1)/2 \rceil - 1$ elements. So e is larger or equal to exactly $\lceil (n+1)/2 \rceil$ elements. But the above case is correct only when n is odd. Therefore, the algorithm is partially correct.

Therefore, the time complexity of this algorithm is $\Theta(1)$, which is much better than (1).

(3) **Using natural language**, explain how to implement `INSERT(x)` operation. You should notice that these two properties should always be held. Analyze the most efficient running time of `INSERT` algorithm in terms of n. (Suppose the total number of element in S is given, which is n.)

Solution

Note that since initially property 2 holds, $|A| = \lceil (n+1)/2 \rceil - 1$, and $|B| = \lfloor (n+1)/2 \rfloor$, where n is the size of S.

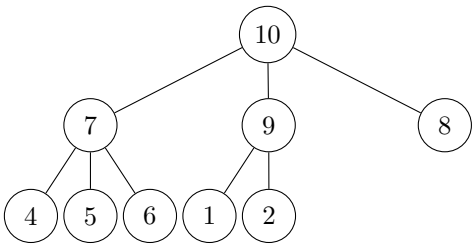
Let x be the element to be inserted. We start by comparing x to the minimum element b of B (we can do it in $O(1)$ time since B is a min-heap). If $x > b$ we do a heap-insertion of x into B, otherwise, w do a heap-insertion of x into A. Either of steps will take $O(\log n)$ time. This way of inserting x ensures that the property 1 still holds.

However, after this insertion, property 2 might be violated by either $|A|$ becoming to equal $|B|+1$, or $|B|$ becoming equal to $|A|+2$. In the first case we extract the maximum element of A and heap-insert it into B, since A is a max-heap and B is a heap, this can be done in $O(\log n)$ time. Similarly, in the second case we extract the minimum element of B and heap-insert it into A, this can be done in $O(\log n)$ time. After this operation, property 2 holds again, and we did not violate property 1 while doing it.

Problem 4: k-ary Heap

In class, Prof. Zhang 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-ary max-heap.



(1) 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, j and i. Please use flooring or ceiling to ensure that your final answer is as tight as possible if you need, i.e. $\lfloor x \rfloor$, $\lceil x \rceil$.

Solution

A k-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 $1 \leq j \leq k$), respectivelt, are:

Parent: index $\lfloor (i-2)/k + 1 \rfloor$.
Child: index $k(i-1) + j + 1$.

(2) 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 k and n. Please use flooring or ceiling to ensure that your final answer is as tight as possible if you need, i.e. $\lfloor x \rfloor$, $\lceil x \rceil$.

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

(3) Now we want to study which value of k can minimize the comparison complexity of heapsort. For heapsort, given a built heap, the worst-case number of comparisons is $\Theta(nhk)$, where $h = \Theta(\log_k n)$ is the height of the heap. Suppose the worst-case number of comparisons is $T(n, k)$. You need to do:

- Explain why $T(n, k) = \Theta(nhk)$.
- Suppose n is fixed, solve for k so that $T(n, k)$ is minimized.

Notice: k is an integer actually. In this problem, we only consider the complexity of comparision, not the accurate number of comparision.

Solution

For the worst case, 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, $T(n, k) = \Theta(nhk)$.

To find the value of k that minimizes the expression, notice that $nk \log_k n = nk\frac{\ln n}{\ln k}$. The derivative of $\frac{k}{\ln k}$ with respect to k is $\frac{1}{\ln k} - \frac{1}{(\ln k)^2}$. Setting this to zero yields, $k = e = 2.718\cdots$ as the minimizing value. But in reality, k is an integer. Therefore, it is easy to varify that $k = 3$.

(4) TA Yuan is motivated by professor, and he has a new idea. He wants to use k-ary heap to implement the heapsort algorithm. Because he wants to loaf on the job, he chooses $k = 1$. And he argues that we only need to do the `BUILD-HEAP` operation in the heapsort algorithm if $k = 1$. Since we know from the lecture that `BUILD-HEAP` takes $O(n)$ time, he thinks it can actually sort in $O(n)$ time!

From the above, we can conclude his two statements:

- When $k = 1$, the only operation required in heapsort algorithm is `BUILD-HEAP`.
- When $k = 1$, the `BUILD-HEAP` operation will run in $O(n)$ time.

Now you are the student of TA Yuan, and you need to judge his two statements are true or false **respectively**.

- If his statement is true, please explain the reason and prove its correctness.
- If his statement is false, please help him find the fallacy in his argument with your own reason. In the heapsort he proposes, what sorting algorithm is actually performed? What is the worst running time of such a sorting algorithm?

Solution

He is correct when he argues that for $k = 1$ the only operation required in the heapsort algorithm is `BUILD-HEAP`. Since each node in the heap has only one child, the heap will be comprised of a single root-to-leaf chain containing the sorted elements. Since our array representation of a heap guarantees that a parent will always precede its children in the array, the elements in the array representing this heap will be correctly sorted.

His argument fails when he claims that this BUILD-HEAP operation will run in $O(n)$ time. Look at the code for BUILD-HEAP:

```
1 BUILD-HEAP(A):
2   heap-size[A] = length[A]
3   for i= length[a]/k-1 downto 1:
4     do HEAPIFY(A,i)
```

Since the elements in the subarray $A[\lceil n/k \rceil \cdots n]$ are leaves in the tree, BUILD-HEAP goes through the remaining positions and runs HEAPIFY on each one. Before each call to HEAPIFY(A, i), the subarray $A[i+1 \cdots n]$ is sotred. Since the unary heap has only one path from the root down to the leaf, the next call to HEAPIFY simply walks along the subarray $A[i+1 \cdots n]$ and inserts $A[i]$ so that subarray $A[i \cdots n]$ is now sorted. This is exactly the operation of INSERT-SORT (but from back-to front, instead of front-to-back) which has worst case running time $\Theta(n^2)$.