CS101 Algorithms and Data Structures
Fall 2022
Homework 4

Due date: 23:59, October 16th, 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.

Notes:

1. Some problems in this homework requires you to design Divide and Conquer algorithm. When grading these problems, we will put more emphasis on how you reduce a problem to a smaller size problem and how to combine their solutions with Divide and Conquer strategy.
2. Your answer for these problems **should** include:
   (a) Algorithm Design
   (b) Time Complexity Analysis
   (c) Pseudocode (Optional)
3. Your answer for these problems is not allowed to include real C or C++ code.
4. In Algorithm Design, you should describe each step of your algorithm clearly.
5. Unless required, writing pseudocode is optional. If you write pseudocode, please give some additional descriptions if the pseudocode is not obvious.
6. You are recommended to finish the algorithm design part of this homework with LaTeX.

1. (0 points) Binary Search Example

Given a sorted array a of n elements, design an algorithm to search for the index of given element x in a.

Solution:

Algorithm Design: We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with the middle element, return the middle index.
3. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
4. Otherwise (x is smaller) recur for the left half.

Pseudocode(Optional):

left and right are indecies of the leftmost and rightmost elements in given array a respectively.

```
1: function BINARYSEARCH(a, value, left, right)
2:     if right < left then
3:         return not found
4:     end if
5:     mid ← ⌊(right − left)/2⌋ + left
6:     if a[mid] = value then
7:         return mid
8:     end if
9:     if value < a[mid] then
10:        return binarySearch(a, value, left, mid-1)
11:    else
12:        return binarySearch(a, value, mid+1, right)
13:    end if
14: end function
```

Time Complexity Analysis: During each recursion, the calculation of mid and comparison can be done in constant time, which is $O(1)$. We ignore half of the elements after each comparison, thus we need $O(\log n)$ recursions.

$$T(n) = T(n/2) + O(1)$$

Therefore, by the Master Theorem $\log_b a = 0 = d$, so $T(n) = O(\log n)$.

2. (9 points) Trees

Each question has **one or more than one** correct answer(s). Please answer the following questions **according to the definition specified in the lecture slides**.

(a)	(b)	(c)
ABD	A	A

(a) (3') Which of the following statements is(are) **false**?

A. Nodes with the same depth are siblings.

B. Each node in a tree has exactly one parent pointing to it.

C. Given any node a within a tree, the collection of a and all of its descendants is a subtree of the tree with root a.

D. The root node cannot be the descendant of any node.

E. Nodes with degree zero are called leaf nodes.

F. Any tree can be converted into a forest by removing the root node.

(b) (3') Given the following pseudo-code, what kind of traversal does it implement?

```
1: function ORDER(node)
2:     visit(node)
3:     if node has left child then
4:         order(node.left)
5:     end if
6:     if node has right child then
7:         order(node.right)
8:     end if
9: end function
```

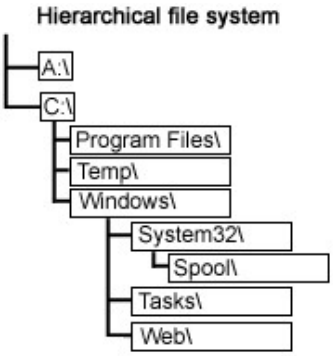
A. Preorder depth-first traversal

B. Postorder depth-first traversal

C. Inorder depth-first traversal

D. Breadth-first traversal

(c) (3') Which traversal strategy should we use if we want to print the hierarchical structure?



A. Preorder depth-first traversal

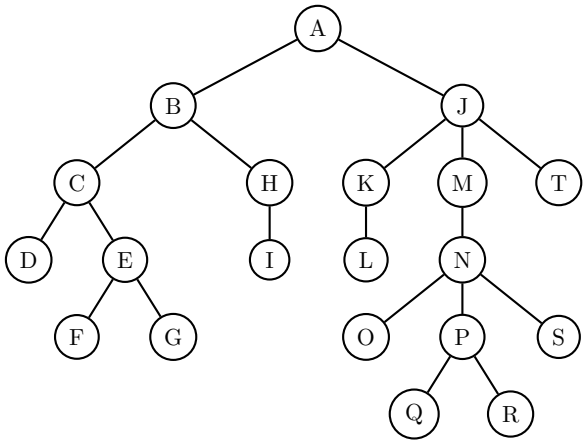
B. Postorder depth-first traversal

C. Inorder depth-first traversal

D. Breadth-first traversal

3. (12 points) Tree Properties

Answer the following questions for the tree shown below **according to the definition specified in the lecture slides**. Please specify:



(a) (2') The **children** of the **root node** with their **degree** respectively.

Solution: BJ, deg(B)=2, deg(J)=3

(b) (2') All **leaf nodes** in the forest with their **depth** if we remove A and the node with the lexicographically smallest character in a tree is taken as the root node.

Solution:
DFGILOQRST
dep(T)=1
dep(D)=dep(I)=dep(L)=2
dep(F)=dep(G)=dep(O)=dep(S)=3
dep(Q)=dep(R)=4

(c) (2') The **height** of the tree.

Solution: height = $\max\{\texttt{depth(nodes)}\} = 5$

(d) (2') The **ancestors** of R.

Solution: AJMNPR

(e) (2') The **descendants** of L.

Solution: L

(f) (2') The **path** from E to S.

Solution: E → C → B → A → J → M → N → S

4. (8 points) Tree Structure and Traversal

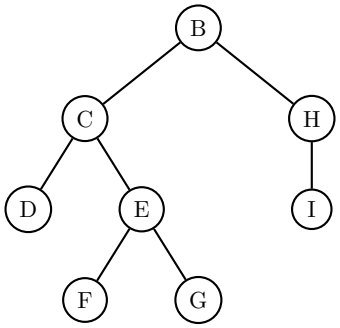
Answer the following questions for the tree shown below **according to the definition specified in the lecture slides**.

Note: Form your answer in the following steps.

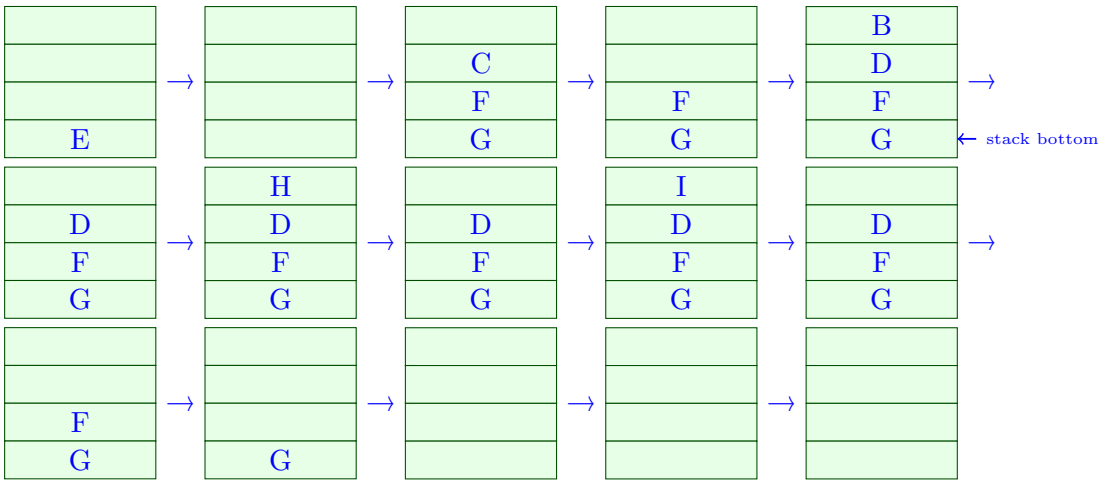
1. Decide on an appropriate **data structure** to implement the traversal.
2. When you are pushing the children of a node into a **queue**, please push them alphabetically; when you are pushing the children of a node into a **stack**, please push them in a reversely alphabetical order.
3. **Show all current elements in your data structure at each step** clearly. **Popping a node** or **pushing a sequence of children** can be considered as one single step.
4. **Write down your traversal sequence** i.e. the order that you pop elements out of the data structure.

Please refer to the examples displayed in the lecture slide for detailed implementation of traversal in a tree using the data structure.

(a) (4') Use stack to run **Preorder Depth First Traversal** in the tree with root E and you should fill stack step by step and then write down the traversal sequence.

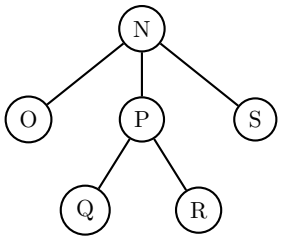


Solution:

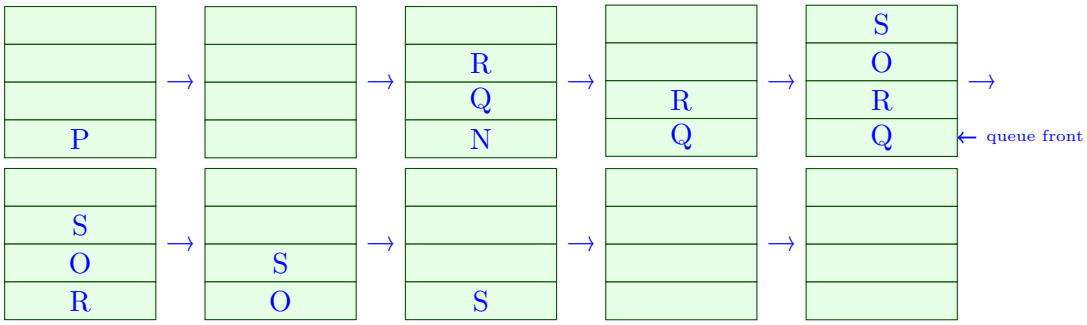


Traversal Sequence: E C B H I D F G

(b) (4') Use queue to run **Breadth First Traversal** in the tree with root P and you should fill queue step by step and then write down the traversal sequence.



Solution:



Traversal Sequence: P N Q R O S

5. (12 points) Recurrence Relations

For each question, find the asymptotic order of growth of $T(n)$ i.e. find a function g such that $T(n) = O(g(n))$. You may ignore any issue arising from whether a number is an integer. You can make use of the Master Theorem, Recursion Tree or other reasonable approaches to solve the following recurrence relations.

(a) (4') $T(n) = 4T(n/2) + 2^4\sqrt{n}$ and $T(0) = 1$.

Solution: $(a, b, d) = (4, 2, 1/2)$, so by the Master Theorem $\log_b a = 2 > d = 1/2$, hence $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

(b) (4') $T(n) = T(n/4) + T(n/2) + c \cdot n^2$ and $T(0) = 1$, c is a positive constant.

Solution:

$$
\begin{array}{ccccc}
 & & cn^2 & & \\
 & / & & \backslash & \\
 & c\left(\frac{n}{4}\right)^2 & & c\left(\frac{n}{2}\right)^2 & \\
 & / \quad \backslash & & / \quad \backslash & \\
c\left(\frac{n}{16}\right)^2 & c\left(\frac{n}{8}\right)^2 & c\left(\frac{n}{8}\right)^2 & c\left(\frac{n}{4}\right)^2 &
\end{array}
$$

$\cdots$

Then we get the following series

$$T(n) = c(n^2 + \frac{5n^2}{16} + \frac{25n^2}{256} + \cdots) = c(n^2 + \frac{5}{16} \cdot n^2 + \left(\frac{5}{16}\right)^2 \cdot n^2 + \cdots) = cn^2 \cdot \frac{1}{1-5/16}$$

Therefore, $T(n) = \Theta(n^2)$.

(c) (4') $T(n) = T(\sqrt{n}) + 1$ and $T(2) = T(1) = 1$.

Solution:
$2^k \to 2^{k/2} \to 2^{k/4} \to \ldots \to 2^4 \to 2^2 \to 2^1$

So there are $\log_2 k = \log_2(\log_2 n)$ levels in this recursion tree.

For each level, the work is $\Theta(1)$, so the total work $= \log_2(\log_2 n) \times 1 = O(\log(\log n))$.

Therefore, $T(n) = O(\log(\log n))$.

6. (8 points) Maximum Contiguous Subsequence Sum

Given an array $\langle a_1, \cdots, a_n \rangle$ of length n with both **positive** and **negative** elements, we will design a **divide and conquer** algorithm to find the maximum contiguous subsequence sum of a. We say m is the maximum contiguous subsequence sum of a such that for any integer pair (l, r) $(1 \leq l \leq r \leq n)$,

$$m \geq \sum_{i=l}^{r} a_i.$$

And the time complexity limit is $\Theta(n \log n)$.

Solution:

```
1:  function MCSS(a, l, r)
2:      if l ¿ r then
3:          return −∞
4:      end if
5:      mid ← (l + r)/2
6:      left_MCSS ← MCSS(a, l, mid)
7:      right_MCSS ← MCSS(a, mid + 1, r)
8:      left_max ← −∞, right_max ← −∞
9:      suffix_sum ← 0, prefix_sum ← 0
10:     for i = mid; i ≥ l; i − − do
11:         suffix_sum ← suffix_sum + a_i
12:         left_max ← max(left_max, suffix_sum)
13:     end for
14:     for i = mid + 1; i ≤ r; i + + do
15:         prefix_sum ← prefix_sum + a_i
16:         right_max ← max(right_max, prefix_sum)
17:     end for
18:     return max{left_MCSS, right_MCSS, left_max + right_max}
19: end function
```

In the process of recurrence, we divide the array into two equal size sub-arrays and call the function to calculate the MCSS of two sub-arrays which means we get the MCSS whose contiguous subsequences are located on the both sides of mid. In addition, we wonder the contiguous subsequence crosses the mid. In order to accomplish this, we plus the maximum suffix sum of $a_{l \cdots mid}$ and the maximum prefix sum of $a_{mid+1 \cdots r}$. In conclusion, the time complexity is $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$.

7. (12 points) New k-th Minimal Value

Given two **sorted** arrays $\langle a_1, \cdots, a_n \rangle$ of length n and $\langle b_1, \cdots, b_m \rangle$ of length m with $n + m$ **distinct** elements and an integer k ($1 \leq k \leq n + m$), we will design a **divide and conquer** algorithm to find k-th minimal element in the merged array $\langle a_1, \cdots, a_n, b_1, \cdots, b_m \rangle$ of length $n + m$. We say a_x is the k-th minimal value of a if there are exactly $k - 1$ elements in a that are less than a_x, i.e.

$$|\{i \mid a_i < a_x\}| = k - 1.$$

(a) (6') You should design a **divide and conquer** algorithm with time complexity $O(\log n + \log m)$.

Solution:

```
1: function K-TH(a, l_1, r_1, b, l_2, r_2, k)
2:     if l_1 ≥ r_1 then
3:         return b_{l_2+k-1}
4:     end if
5:     if l_2 ≥ r_2 then
6:         return a_{l_1+k-1}
7:     end if
8:     mid_1 ← (l_1 + r_1)/2, mid_2 ← (l_2 + r_2)/2
9:     left_size_1 ← mid_1 − l_1 + 1
10:    left_size_2 ← mid_2 − l_2 + 1
11:    if left_size_1 + left_size_2 ≥ k then
12:        if a_{mid_1} > b_{mid_2} then return K-TH(a, l_1, mid_1, b, l_2, r_2)
13:        else return K-TH(a, l_1, r_1, b, l_2, mid_2)
14:        end if
15:    else
16:        if a_{mid_1} > b_{mid_2} then return K-TH(a, l_1, r_1, b, mid_2 + 1, r_2)
17:        else return K-TH(a, mid_1 + 1, r_1, b, l_2, r_2)
18:        end if
19:    end if
20: end function
```

In the process of recurrence, we divide the two arrays into four equal size sub-arrays which are called a_l, a_r, b_l, b_r. Then we consider four situation (we use $|a_l|$ to denote the length of a_l)

- $|a_l| + |b_l| \geq k$ and the largest element in a_l is greater than the largest element in b_l, we can infer that the k-th minimal value cannot be in a_r otherwise there are at least k element in a and b which is smaller than it.
- $|a_l| + |b_l| \geq k$ and the largest element in a_l is not greater than the largest element in b_l, we can infer that the k-th minimal value cannot be in b_r otherwise there are at least k elements in a and b which is smaller than it.

- $|a_l| + |b_l| < k$ and the largest element in a_l is greater than the largest element in b_l, we can infer that the k-th minimal value cannot be in b_l otherwise there are at most $k-2$ elements in a and b which is smaller than it.
- $|a_l| + |b_l| < k$ and the largest element in a_l is not greater than the largest element in b_l, we can infer that the k-th minimal value cannot be in a_l otherwise there are at most $k-2$ elements in a and b which is smaller than it.

And every recursion, the length of a or length of b will be divided by two, so the time complexity is $O(\log m + \log n)$.

(b) (6') You should design **another divide and conquer** algorithm with better time complexity $O(\log k)$.

Solution:

```
1:  function K-TH(a, l1, r1, b, l2, r2, k)
2:      if l2 ≥ r2 then
3:          return a_{l1+k-1}
4:      end if
5:      if r1 − l1 > r2 − l2 then
6:          return K-TH(b, l2, r2, a, l1, r1, k)
7:      end if
8:      if r1 − l1 + 1 = 0 then
9:          return b_{l2+k-1}
10:     end if
11:     if k = 1 then
12:         return min(a_{l1}, b_{l2})
13:     end if
14:     i ← min(r1 − l1 + 1, k/2), j ← min(r2 − l2 + 1, k/2)
15:     if a_{l1+i-1} > b_{l2+j-1} then
16:         return K-TH(a, l1, r1, b, l2 + j, r2, k − j)
17:     else
18:         return K-TH(a, l1 + i, r1, b, l2, r2, k − i)
19:     end if
20: end function
```

In the process of recurrence, we consider the least $k/2$ elements in a and b which are called a_l, b_l. Then we consider two situation

- The largest element in a_l is greater than the largest element in b_l, we can infer that the k-th minimal value cannot be in b_l otherwise there are at most $k-2$ elements in a and b which is smaller than it.
- The largest element in a_l is not greater than the largest element in b_l, we can infer that the k-th minimal value cannot be in a_l otherwise there are at most $k-2$ elements in a and b which is smaller than it.

And every recursion, k will be divided by two, so the time complexity is $O(\log k)$.