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.

```
function binarySearch($a$, $value$, $left$, $right$)
    if $right < left$ then
        return not found
    end if
    $mid \leftarrow \lfloor (right - left)/2 \rfloor + left$
    if $a[mid] = value$ then
        return $mid$
    end if
    if $value < a[mid]$ then
        return binarySearch($a$, $value$, $left$, $mid - 1$)
    else
        return binarySearch($a$, $value$, $mid + 1$, $right$)
    end if
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?

```
function ORDER(node)
    visit(node)
    if node has left child then
        order(node.left)
    end if
    if node has right child then
        order(node.right)
    end if
end function
```

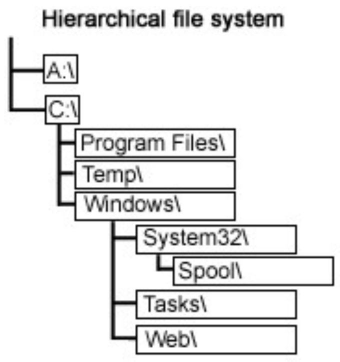
A. Preorder depth-first traversal

B. Postorder depth-first traversal

C. Inorder depth-first traversal

D. Breadth-first traversal

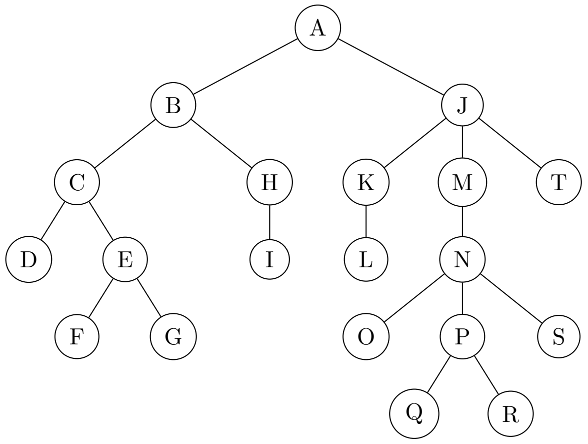
(c) (3') Which traversal strategy should we use if we want to print the hierarchical structure?



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:

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:

depth (D) = 2
depth (F) = 3
depth (G) = 3
depth (I) = 2
depth (L) = 2
depth (O) = 3
depth (Q) = 4
depth (R) = 4
depth (S) = 3
depth (T) = 1

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

Solution:

height = 5

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

Solution:

R, P, N, M, J, A

(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

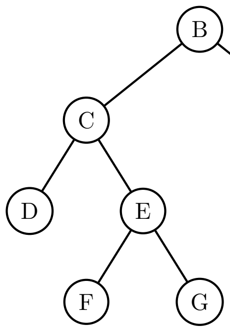
Answer the following questions for the tree shown below **according to the definition specified in the lecture slides**.

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:

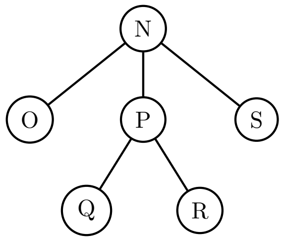
				B		H		I		
	→	C	→	D	→	D	→	D	→	
		F		F		F		F		
E		G		G		G		G		← stack bottom

D	→		→		→		→		→	
F		F								
G		G		G						

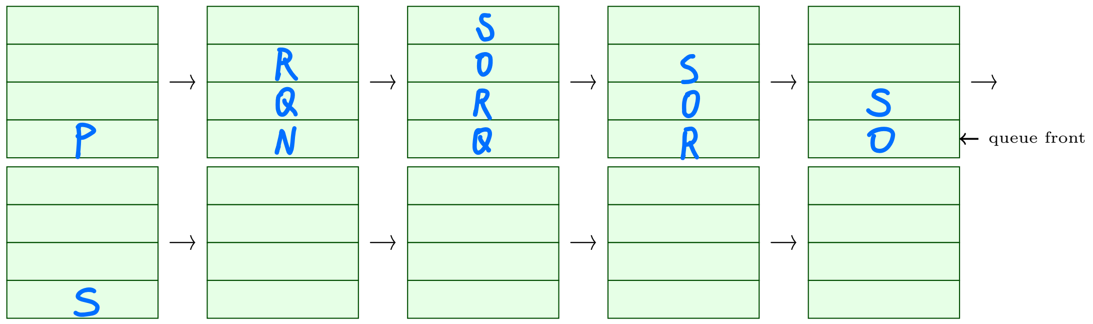
	→		→		→		→	

Traversal Sequence:

(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:

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

Solution:

$\log_2 4 = 2 > \frac{1}{2}$

according to Master Theorem

$$T(n) = \begin{cases} 1 & , n=0 \\ O(n^2) & , n>0 \end{cases}$$

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

Solution: assume $T(\frac{n}{2}) \geq T(\frac{n}{4})$

so $T(n) \leq 2T(\frac{n}{2}) + cn^2$

apply Master Theorem to the right side

$T(n) \leq \theta(n^2) \Rightarrow T(n) = O(n^2)$

$T(n) \geq cn^2 \Rightarrow T(n) \geq \theta(n^2) \Rightarrow T(n) = \Omega(n^2)$

since $T(n) = O(n^2)$ and $T(n) = \Omega(n^2)$

$T(n) = \theta(n^2)$

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

Solution: $T(n) = T(n^{\frac{1}{2}}) + 1 = T(n^{\frac{1}{4}}) + 2 = \cdots = T(n^{\frac{1}{2k}}) + k$

the recursion will stop at some point

let $n^{\frac{1}{2k}} = 2$

we can get $2^{2k} = n \Rightarrow k = \log(\log n)$

So $T(n) = \begin{cases} 1 & , n = 1, 2 \\ \log(\log n) & , n \geq 3 \end{cases}$

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) $(1 \le l \le r \le n)$,

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

The time complexity should be $\Theta(n \log n)$.

Solution:

Algorithm:

1.Divide the given array in two halves

2.Return the maximum of following:

①Maximum subarray sum in left half (using recursion)

②Maximum subarray sum in right half (using recursion)

③Maximum subarray sum such that the subarray crosses the middle point

(find the maximum sum starting from middle point and ending at some point on left of mid, then find the maximum sum starting from mid + 1 and ending with some point on right of mid + 1. Finally, combine the two and return the maximum among left, right and combination of both)

Time complexity analysis:

Finding maximum subarray sum is a recursive method and time complexity can be expressed as following recurrence relation: $T(n) = 2T(n/2) + \Theta(n)$

So the time complexity is $\Theta(n \log n)$

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 ($1 \le k \le n + m$), 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:

Algorithm:

1.Compare the middle elements of arrays arr1 and arr2(their indices are mid1 and mid2 respectively). Assume arr1[mid1]=k, then the elements after mid2 cannot be the required element. Set the last element of arr2 to be arr2[mid2].

2.In this way, define a new subproblem with half the size of one of the arrays.

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

Solution:

Algorithm:

1.Instead of dividing the array into segments of n / 2 and m / 2 and then recurse, we can divide them both by k / 2 and recurse.

2.Operate in the same way as in question (a)