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. X

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

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

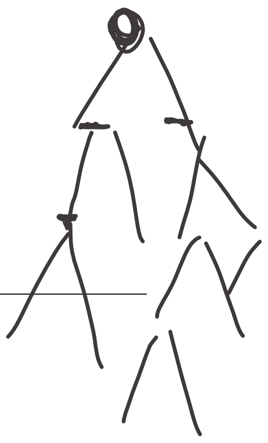
D. The root node cannot be the descendant of any node. X itself.

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
```



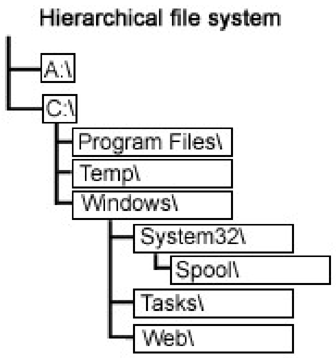
A. Preorder depth-first traversal

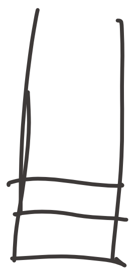
B. Postorder depth-first traversal

C. Inorder depth-first traversal

D. Breadth-first traversal X

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





A. Preorder depth-first traversal

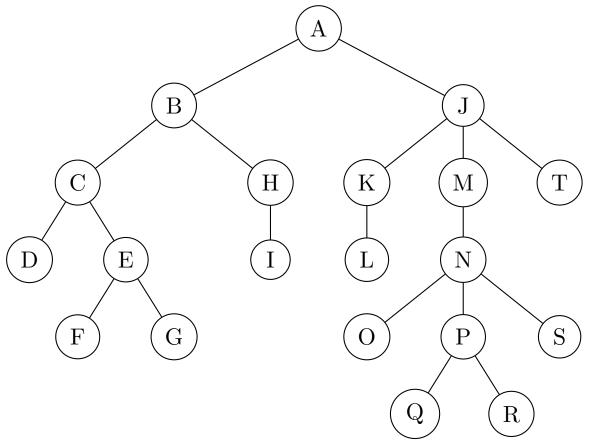
B. Postorder depth-first traversal

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: D. has depth 2 , F has depth 3, G has depth 3 I has depth 2 , L has depth 2, O has depth 3, Q has depth 4, R has depth 4, S has depth 3, T has depth 1

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

Solution: 5.

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

Solution: A . J , M . N . P . R

(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) has length 7.

4. (8 points) Tree Structure and Traversal

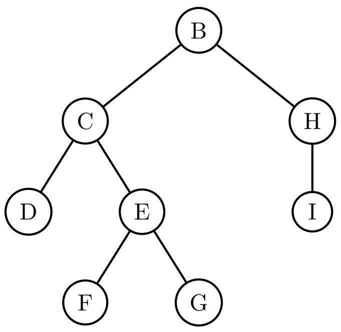
Answer the following questions for the tree shown below **according to the definition specified in the lecture slides**.

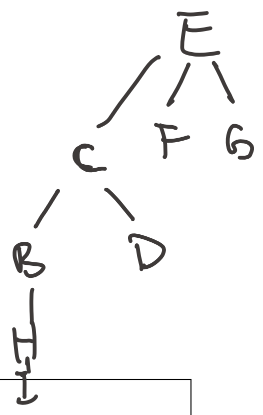
Note: Form your answer in the following steps.

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:

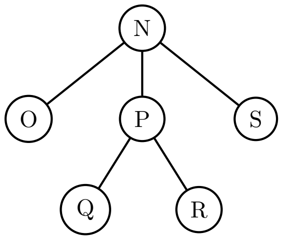
								B		
	→		→	C	→		→	D	→	
				F		F		F		
E				G		G		G		← stack bottom

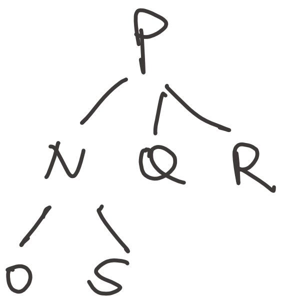
		H				I			
D	→	D	→	D	→	D	→	D	→
F		F		F		F		F	
G		G		G		G		G	

	→		→		→		→	
F								
G		G						

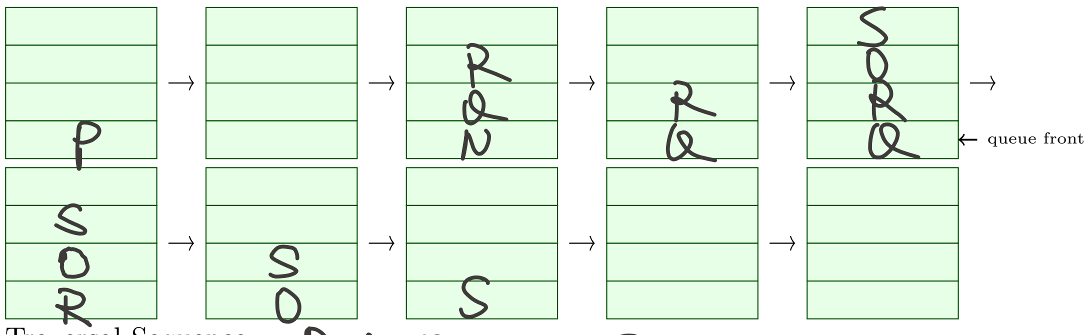
Traversal Sequence: E C B H I D F G

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

Solution:

$$T(n) = \begin{cases} 1 & n=0 \\ 4T(\frac{n}{2}) + 2^4\sqrt{n} & n \neq 0. \end{cases}$$

according to Master Theorem,

$d = \frac{1}{2} < \log_b a = \log_2 4 = 2.$

$\therefore T(n) = O(n^2)$

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

Solution:

$$T(n) = 2T(\tfrac{n}{4}) + T(\tfrac{n}{8}) + cn^2$$

$$\underbrace{2T(\tfrac{n}{4}) + cn^2}_{①} < T(n) < \underbrace{3T(\tfrac{n}{4}) + cn^2}_{②}$$

① $a=2.\ b=4.$ $\log_b a = \log_4 2 = \frac{1}{2} < d = 2$ $\therefore T(n) = O(n^2).$

② $a=3.\ b=4$ $\log_b a = \log_4 3 < d = 2$ $\therefore T(n) = O(n^2)$

$\therefore T(n) = O(n^2).$

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

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

$= (T(n^{\frac{1}{4}}) + 1) + 1$

$\sqrt[2^k]{n} = 2$ $\qquad = T(t) + k$

$= 1 + \sqrt[2^k]{n}$

$2^{2^k} = n.$

$2^k = \log_2 n.$ $\quad \therefore T(n) = O(\ln \ln n).$

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

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

相邻.

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

Solution:

Algorithm Design: We basically ignore half of the elements just after one comparison.
1. Divide a sequence into two roughly equal segments.
2. Get the maximum sum of subsequences in the left segment(recur)
3. Get the maximum sum of subsequences in the right segment(recur)
4. Get the maximum sum of subsequences in the middle (Sum of the largest subsequence with the right endpoint on the left and the largest subsequence with the left endpoint on the right)
5. Compare three sums (the maximum subsequence sum on the left and right respectively and the maximum value in the maximum subsequence sum spanning the two parts)

Time Complexity Analysis: During each recursion, left loop is from the midpoint to the start point, and the right is from the midpoint to the end point, which is O(N). We ignore half of the elements after each comparison, thus we need O(log N) recursions.
T(1) = 1
T(N) = 2T(N/2) + O(N)
Therefore, by the Master Theorem, a=2,b = 2,d = 2,T(N) = O(NlogN)

Pseudocode (Optional): left and right are indecies of the leftmost and rightmost elements in given array a respectively

```
function maxSubSum(a, left, right)
    if right = left then
        return max(0, a[right])
    end if
    mid ← (right − left)/2 + left
    maxleftSum = maxSubSum (a, left, mid)
    maxrightSum = maxSubSum (a, mid + 1, right)
    for I from mid to left:
        leftSumwithborder+=a[i]
        maxleftSumwithborder = max(maxleftSumwithborder, leftSumwithborder)
end loop
for I from mid to right:
        rightSumwithborder+=a[i]
          maxrightSumwithborder = max(maxrightSumwithborder, rightSumwithborder)
end loop;
maxmiddleSum = maxleftSumwithborder+ maxrightSumwithborder
    return max(maxleftSum, maxrightSum, maxmiddleSum)
end function
```

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 Design:
1. Divide A sequence into two segments. One segment has i elements and the other has (lengtha-i) elements. Divide B sequence into two segments. One segment has j elements and the other has (lengthb-j) elements. The initial values of i and j are allocated in proportion to the size of A and B.
2. If B [j-1]<A [i]<B [j], return A [i]; if A [i-1]<B [j]<A [i], return B [j]. Because kth smallest must be A [i] or B [j]
3. If not 2, compare A [i] and B [j]. If A [i]<B [j], we can exclude elements smaller than A [i] and elements larger than B [j]. We can find the k-i smallest element in A [i+1, m] and B [0, j]. If A [i]>B [j], we can exclude elements smaller than B [j] and elements larger than A [i]. We can find the element with the lowest k-j in A [0, i] and B [j+1, n]

Time Complexity Analysis:
Each time, the scale of A or B can be reduced by half, so recursion times can only be loglengtha+loglengthb. After half reduction, we continue to use the above algorithm as a new problem until A or B is reduced to a small enough size: If A is gone, so we only need to find the k-largest element in B, that is, B [k]. If B is gone, the same result is A [k]. During each recursion, we have to compare A [i] and B [j], that is O(1).
A [lengtha]+ B [lengthb] = A [i]+ B [lengthb] + O(1)
A [lengtha]+ B [lengthb] = A [lengtha]+ B [j] + O(1)
Therefore, T(N) = O(loglengtha+loglengthb)

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

Solution:

```
function find(a, n, b, m, k)
  if n > m then
     return find(b, n, a, m, k)
  end if
  if n=0 then
     return b[k-1]
  end if
  if k = 1 then
     return min(a[0],b[0])
end if
halfk ← min(k/2, n)
halfk_another ← k - halfk
if a[halfk -1] < b[halfk_another -1] then
   return find(a+ halfk, n - halfk, b, m, k-halfk)
else if(a[halfk -1] > b[halfk_another -1]) then
   return find(a, n, b+ halfk_another, m-halfk_another,k- halfk_another)
else
   return a[halfk -1]
end if
end function
```

Algorithm Design:
1. If A [] or B [] is empty, that is, there is only one array, then B [k-1] or A [k-1] is returned.
2. let the array with least elements to be A, the array with least elements to be B.
3. Compare A[halfk -1] and B[halfk_another -1]. If A[halfk-1] < B[halfk_another -1], we can exclude elements smaller than A [halfk-1]. We can find the k-halfk smallest element in A [halfk, m] and B [0, n]. If A[halfk-1] > B[halfk_another -1], we can exclude elements smaller than B [halfk_another-1]. We can find the element k- halfk_another in A [0, m] and B [halfk_another, n]

Time Complexity Analysis: During each recursion, the scale of k can be reduced by half. After half reduction, we continue to use the above algorithm as a new problem until A or B or k is reduced to a small enough size. During each recursion, we have to compare A [k/2] and B [k/2], which is O(1).
T(k) = O(k/2)+O(1)
Therefore, by the Master Theorem, a=1,b = 2,d = 1,T(N) = O(logk)