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
```

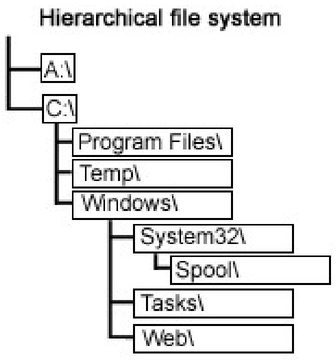
A. Preorder depth-first traversal

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. CorrectChoice depth-first traversal

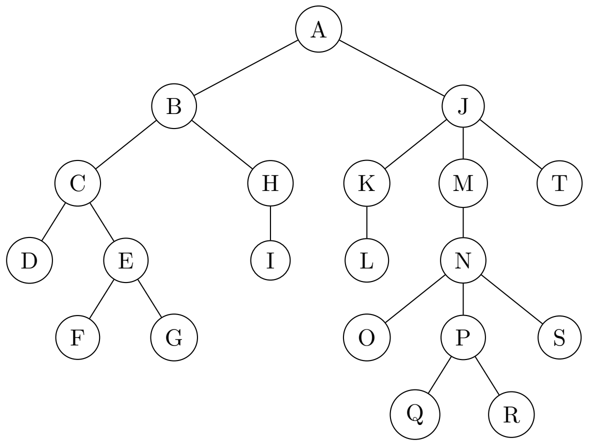
B. Postorder depth-first traversal

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: B, J
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, F, G, I, L, O, Q, R, S, T
height(T) = 1
height(D) = height(I) = height(L) = 2
height(F) = height(G) = height(O) = height(S) = 3
height(Q) = height(R) = 4

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

Solution: height = max{depth(v) | v ∈ V} = 5
so 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: $E \to C \to B \to A \to J \to M \to N \to S$

4. (8 points) Tree Structure and Traversal

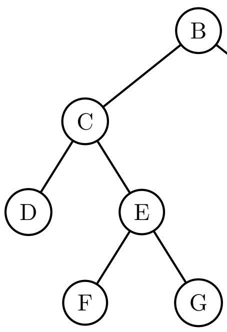
Answer the following questions for the tree shown below **according to the definition specified in the lecture slides**.

Note: Form your answer in the following steps.

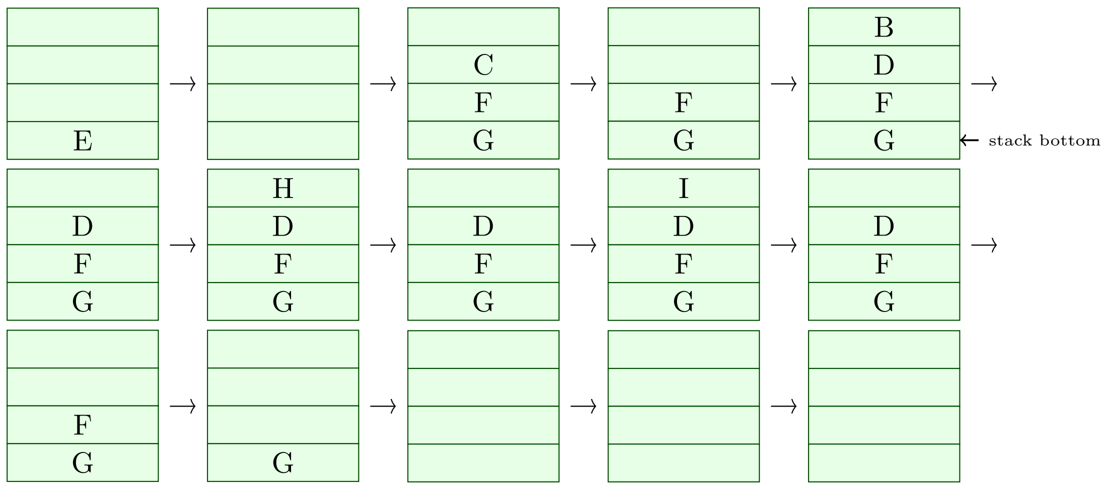
1. Decide on an appropriate **data structure** to implement the traversal.
2. When you are pushing the children of a node into a **queue**, please push them alphabetically; when you are pushing the children of a node into a **stack**, please push them in a reversely alphabetical order.
3. **Show all 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.

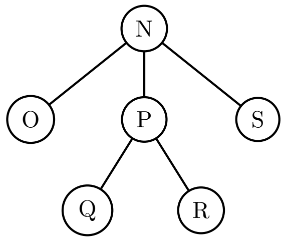


Solution:

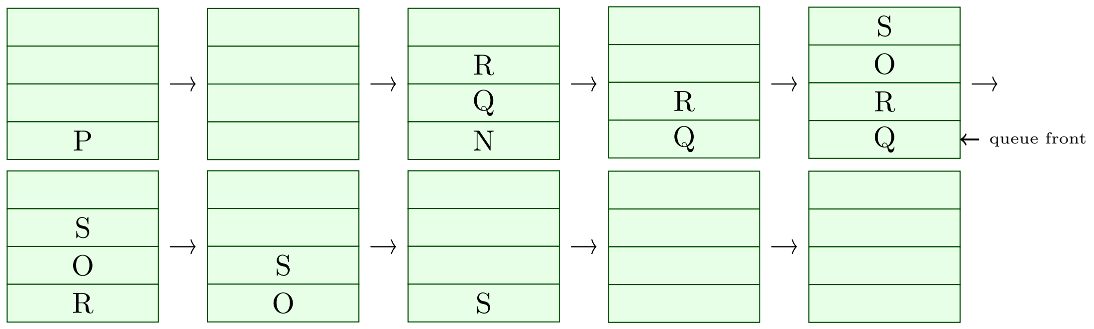


Traversal Sequence: E, C, B, H, I, D, F, G

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

Solution: by the Master Theorem,
$a = 4, b = 2, and\ 2^4\sqrt{n} = \Theta(\sqrt{n}), so\ d = \frac{1}{2}$
$log_b a = log_2 4 = 2 > d = \frac{1}{2}$
so $T(n) = O(n^{log_b a}) = O(n^2)$
$T(n) = O(n^2), g(n) = n^2$

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

Solution: set f_k be k-th the element in the fabonacci sequence
which $f_0 = 1, f_1 = 1, f_n = f_{n-2} + f_{n-1} (n \geq 2)$
and since $f_1 = 1 < 2^1, f_2 = 2 < 2^2, f_n = f_{n-2} + f_{n-1}, 2^n = 2^{n-1} + 2^{n-1}$
so $f_k \leq 2^k$
since $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + c \cdot n^2 = f_0 \cdot T(\frac{n}{2}) + f_0 \cdot T(\frac{n}{4}) + f_1 \cdot c \cdot n^2$
$T(\frac{n}{2}) = T(\frac{n}{4}) + T(\frac{n}{8}) + f_1 \cdot c \cdot (\frac{n}{2})^2$
so $T(n) = (f_0 + f_1) \cdot T(\frac{n}{4}) + f_1 \cdot T(\frac{n}{8}) + f_0 \cdot c \cdot n^2 + f_1 \cdot c \cdot (\frac{n}{2})^2 =$
$= f_2 \cdot T(\frac{n}{4}) + f_1 \cdot T(\frac{n}{8}) + f_0 \cdot c \cdot n^2 + f_1 \cdot c \cdot (\frac{n}{2})^2$
$\cdots$
$= f_i \cdot T(\frac{n}{2^i}) + f_{i-1} \cdot T(\frac{n}{2^{i+1}}) + \sum_{k=0}^{i-1} f_k \cdot c \cdot (\frac{n}{2^k})^2$
$= f_i \cdot \{T(\frac{n}{2^{i+1}}) + T(\frac{n}{2^{i+2}}) + c \cdot (\frac{n}{2^i})^2\} + f_{i-1} \cdot T(\frac{n}{2^{i+1}}) + \sum_{k=0}^{i-1} f_k \cdot c \cdot (\frac{n}{2^k})^2$
$= f_{i+1} \cdot T(\frac{n}{2^{i+1}}) + f_i \cdot T(\frac{n}{2^{i+2}}) + \sum_{k=0}^{i} f_k \cdot c \cdot (\frac{n}{2^k})^2$
$\cdots$
so $T(n) \leq 1 + 1 + \sum_{k=01}^{\lceil logn \rceil} f_k \cdot c \cdot (\frac{n}{2^k})^2$
and since we have proved above $f_k \leq 2^k$
so $T(n) \leq c \cdot \sum_{k=0}^{\lceil logn \rceil} 2^k * \frac{n^2}{(2^k)^2}$
$T(n) \leq c \cdot n^2 \cdot \sum_{k=0}^{\lceil logn \rceil} \frac{1}{2^k} \leq 2c \cdot n^2$
$T(n) \leq 2c \cdot n^2$
so above all, $T(n) = O(n^2), g(n) = n^2$

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

Solution: let $n^{\frac{1}{2^k}} \leq 2$
so $2^k \geq \log_2 n$
so $k \geq \lceil \log_2(\log_2 n) \rceil$
since T(2)=T(1)=1
so $T(n) = T(\sqrt{n}) + 1 = T(n^{\frac{1}{4}}) + 2 = \cdots = T(n^{\frac{1}{2^k}}) + k \leq 1 + k$
so $T(n) = O(k) = O(\log_2 \log_2 n)$
so above all, $T(n) = O(\log\log n), g(n) = \log\log n$

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

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

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

Solution:

Algorithm Design: after divide the sequence into two parts from the middle

the maximum contiguous subsequence may have three possible location distributions

1. the whole subsequence is in the left part
2. the whole subsequence is in the right part
3. the subsequence is in both left part and right part, and it get through the middle

and for the first two situations, for the left-subsequence and right-subsequence, they have the samilar situation above

and for the third situation, we just need to start at the middle point, expand to the left and right respectively, and find the maximum contiguous subsequence respectively, then sum them up.

so we can recursively divide and conquer the problem

Pseudocode : $\mathtt{left}$ and $\mathtt{right}$ are indecies of the leftmost and rightmost elements in given array a respectively.

```
function GET__MAX__CONTIGUOUS__SUBSEQUENCE($a$, $left$, $right$)
    if $right == left$ then
        return max(0,a[left])
    end if
    $mid \gets \lfloor (left + right)/2 \rfloor$
    $left\_subsequence \gets get\_max\_contiguous\_subsequence(a, left, mid)$
    $right\_subsequence \gets get\_max\_contiguous\_subsequence(a, mid+1, right)$
     define sum=0,lmax=0
     for(i from mid to left):
         sum+=a[i]
         lmax=max(lmax,sum)
     similarly, define sum=0,rmax=0
     for(i from (mid+1) to right):
         sum+=a[i]
         rmax=max(rmax,sum)
    $mid\_subsequence \gets (lmax + rmax)$
    $maxn \gets max\{left\_subsequence, right\_subsequence, mid\_subsequence\}$
    return maxn
end function
```

Time Complexity Analysis: During each recursion, the calculation of mid and comparison can be done in constant time, and getting the $lmax$ and $rmax$ takes the time of the sequence's length, which is exactly $\Theta(n)$. We will take both half part of the sequence, thus there are two times of each subproblem.

$$T(n) = 2 \cdot T(\frac{n}{2}) + \Theta(n)$$

Therefore, by the Master Theorem,

$a = 2, b = 2, \log_b a = 1 = d$, so $T(n) = \Theta(nlogn)$.

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: we do binary search for two arrays together

let mida,midb be the middle point of array a,b

and we take out two subarray a', b' from array a,b

start at their left bound, end at their middle,and mark the subarray's size be la,lb

without loss of generality, we regard a'[mida] < b'[midb].(if opposite, we just need to swap the two array and do it again)

and we need to discuss:

if $l1 + l2 \le k$,then all of the elements in a' must have the index $\le k$ in the merged array,so we take them and solve the subproblem.

if $l1 + l2 > k$,then all of the elements in b but not in b' must not have the index $\le k$ in the merged array,so we do not take them, and solve the subproblem

Algorithm Design: we do binary search for two arrays together

1. get the middle point mida, midb of array a,b
2. compare a[mida] and b[midb], to make sure that a have the less element, if not, just swap the two arrays and do it again
3. compare $l1 + l2$ and k, and solve the subproblem we discussed above

Pseudocode : the lefta, righta is the left,right bound of array a in this subproblem,

the leftb, rightb is the left,right bound of array b in this subproblem,

k means that we need to find the kth minimal element in the subprolem.

$mida, midb$ are the middle point of the arrays

l1,l2 are the length of the subarray that start at left bound and end at the middle

```
function KTH_MIN($a$, $b$, $lefta$, $righta$, $leftb$, $rightb$, $k$)
    if $righta < lefta$ then
        return b[leftb+k-1]
    end if
    if $rightb < leftb$ then
        return a[lefta+k-1]
    end if
    $mida \leftarrow (righta + lefta)/2$
    $midb \leftarrow (rightb + leftb)/2$
    if $a[mida] > b[midb]$ then
        return kth_min($b$, $a$, $leftb$, $rightb$, $lefta$, $righta$, $k$)
    end if
    $l1 \leftarrow mida - lefta + 1$
    $l2 \leftarrow midb - leftb + 1$
    if $l1 + l2 \leq k$ then
        return kth_min($a$, $b$, $mida + 1$, $righta$, $leftb$, $rightb$, $k - l1$)
    else
        return kth_min($a$, $b$, $lefta$, $righta$, $leftb$, $midb - 1$, $k$)
    end if
end function
```

Time Complexity Analysis: During each recursion, the calculation of $mida, midb, l1, l2$ 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 for each array. and we have two arrays with length of n and m, so the complexity is to sum them usepackage

which means that the complexity is $O(logn + logm)$.

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

Solution: to shortly explain, in each subprolem, we need to take the first few elements from the remaining elements of array a and b, mark the new subarray we take out as a' and b'

since a' and b' are sorted and have distincy elements,so we need to compare the last element of each new array,

the new subarray with the shorter last element must all be involed in the first k element of the merged array,

so we need to delete them before solving the subproblem,

as for the new subarray with the larger last element, we cannot make sure how many elements are in the merged array, so we just remain them, and figure out in the subproblem.

Algorithm Design: we divide and conquer by using the half of the k

1. we get the value of $len = \lfloor \frac{k}{2} \rfloor$
2. then we need to pick one element from each array, mark as a_pick,b_pick. which means that we are going to take the sequence from pos to $pick$.(pos will be explained in the pseudo code part)
3. to make it easier to write, we just let array a has less remaining elements.(if opposite, just swap the two arrays)
4. if the length of the remaining elements of the array a $< len$, we just take the end of the element as pick. But we need to make sure that the sum of elements we take from array a and b should be k.
5. then compare the element at a_pick in a and at b_pick in b.
6. delete the elements from the array that is the less one from the result above, and let $k\_new = k$ - (the number of elements we removed), and repeat the above operations until $k = 1$ or one of the array is empty.

Pseudocode: $posa$ and $posb$ are indecies of the leftmost that still remain in given array a, b respectively.

$len(\cdot)$ is the function the get the array's size. And we set the index of the elements in array start at 1

to easier get the element that we need to compare, just let the array a be the short one, so we just need to add a judge that is the remaining of array a is less than b, then swap them.(the 2nd to 4th row of the pseudo code) then the array a always has less remain elements.

```
function $\mathrm{kth\_min}(a, b, posa, posb, k)$
    if $(len(a) - posa + 1) > (len(b) - posb + 1)$ then
        return $\mathrm{kth\_min}(b, a, posb, posa, k)$
    end if
    if $posa > len(a)$ then
        return b[posb+k-1]
    end if
    if $posb > len(b)$ then
        return a[posa+k-1]
    end if
    if $k == 1$ then
        return max{a[posa],b[posb]}
    end if
    $length \leftarrow \lfloor k/2 \rfloor$
    $a\_remain \leftarrow (len(a) - posa + 1)$
    if $a\_remain <= length$ then
        new_posa=len(a)
    else
        new_posa=(posa+length-1)
    end if
    $new\_posb \leftarrow (posb + k - (posa - new\_posa + 1) - 1)$
    if $a[new\_posa] < b[new\_posb]$ then
        return $\mathrm{kth\_min}(a, b, new\_posa+1, posb, k-(new\_posa-posa+1))$
    else
        return $\mathrm{kth\_min}(a, b, posa, new\_posb+1, k-(new\_posb-posb+1))$
    end if
end function
```

Time Complexity Analysis: During each recursion, the calculation of $length, new\_posa, new\_posb$ and comparison can be done in constant time, which is $O(1)$. after one recursion, the k become $\frac{k}{2}$ unless the short array turn to the end, but in that case, the function will finish,so

$T(k) = T(\frac{k}{2}) + O(1)$ Therefore, by the Master Theorem $a = 1, b = 1, d = 0, \log_b a = 0 = d$, so $T(k) = O(\log k)$.