ShanghaiTech University

CS101 Algorithms and Data Structures

Fall 2021

Homework 4

Due date: 23:59, October 24, 2021

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 the above may result in zero grade.
8. Problem 0 gives you a template on how to organize your answer, so please read it carefully.

Problem 0: Notes and Example

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. In Algorithm Design, you should describe each step of your algorithm clearly.
4. Unless required, writing pseudocode is optional. If you write pseudocode, please give some additional descriptions if the pseudocode is not obvious.
5. You are recommended to finish the algorithm design part of this homework with LaTeX.

0: 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.

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 ← ⌊(right − left)/2⌋ + 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 $\log_b a = 1 = d$, so $T(n) = O(\log n)$.

1: (2' + 2' + 2') Trees

Each question has **exactly one** correct answer. Please answer the following questions **according to the definition specified in the lecture slides**.

Note: Write down your answers in the table below.

Question 1	Question 2	Question 3
D	B	A

Question 1. *Which of the following statements is true?*

(A) Each node in a tree has exactly one parent pointing to it.

(B) Nodes with the same ancestor are siblings.

(C) The root node cannot be the descendant of any node.

(D) Nodes whose degree is zero are also called leaf nodes.

Question 2. *Given the following pseudo-code, what kind of traversal does it implement?*

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

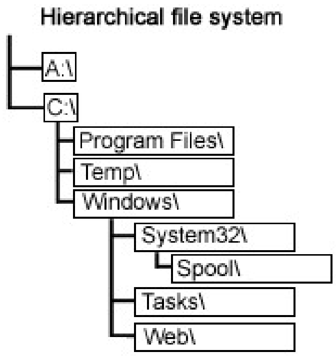
(A) Preorder depth-first traversal

(B) Postorder depth-first traversal

(C) Inorder depth-first traversal

(D) Breadth-first traversal

Question 3. *Which traversal strategy should we use if we want to print the hierachical structure ?*



(A) Preorder depth-first traversal

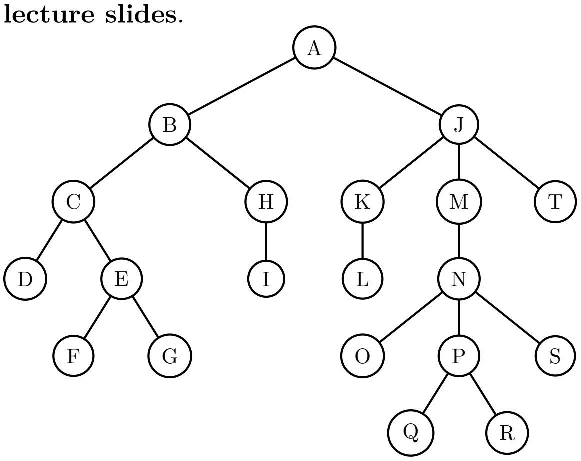
(B) Postorder depth-first traversal

(C) Inorder depth-first traversal

(D) Breadth-first traversal

2: (3+3+3pts) Tree Structure and Traversal

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



Question 4. *Please specify:*

1. *The **children** of the **root node** with their **degree** respectively.*
   B, J; deg(B) = 2, deg(J) = 3.

2. *All **leaf nodes** in the tree with their **depth** respectively.*
   F, G, Q, R; Depth of F and G are 4, Q and R are 5.

3. *The **height** of the tree.*
   5.

4. *The **ancestors** of O.*
   A, J, M, N, O.

5. *The **descendants** of C.*
   C, D, E, F, G.

6. *The **path** from A to S.*
   (A, J, M, N, S).

For the following two questions, traverse the **subtree** of the tree shown above with specified root.

Note: Form your answer in the following steps.

1. Decide on an appropriate **data structure** to implement the traversal.

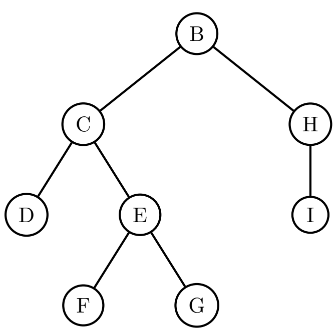
2. When you are pushing the children of a node into a **queue**, please push them alphabetically i.e. from left to right; when you are pushing the children of a node into a **stack**, please push them in a reverse order i.e. from right to left.

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.

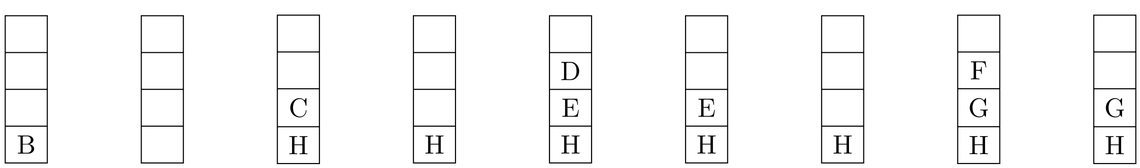
4. **Write down your traversal sequence** i.e. the order that you pop elements out of the data structure.

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

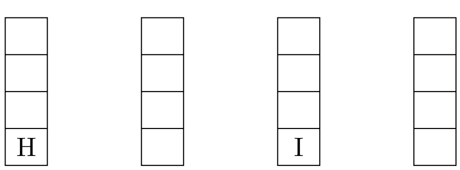
Question 5. *Run **Depth First Traversal** in the subtree with root B.*



We use stack to implement the traversal. Each step, we pop the top elment and then push all its children into the stack.



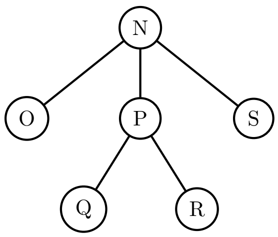
push B. pop B. push C, H. pop C. push D, E. pop D. pop E. push F, G. pop F.



pop G; pop H; push I. pop I.

Pop sequence: B, C, D, E, F, G, H, I.

Question 6. *Run* ***Breadth First Traversal*** *in the subtree with root N.*



We use queue to implement traversal.

N					

Push N.

pop N.

O	P	S			

Push O, P, S.

P	S				

Pop O.

S					

Pop P.

S	Q	R			

Push Q, R.

Q	R				

Pop S.

R					

Pop Q.

Pop R.

Pop sequence: N, O, P, S, Q, R.

3: (2+3pts) 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.
Note: ***Mark or circle*** *your final answer clearly.*

Question 7. $T(n) = 4T(n/2) + 42\sqrt{n}$.

$$a = 4, b = 2, \log_b a = \log_2 4 = 2.$$

Since $2 < \frac{1}{2}$, by Master Theorem, $T(n) = O(n^2)$.

Question 8. $T(n) = T(\sqrt{n}) + 1$. *You may assume that* $T(2) = T(1) = 1$.

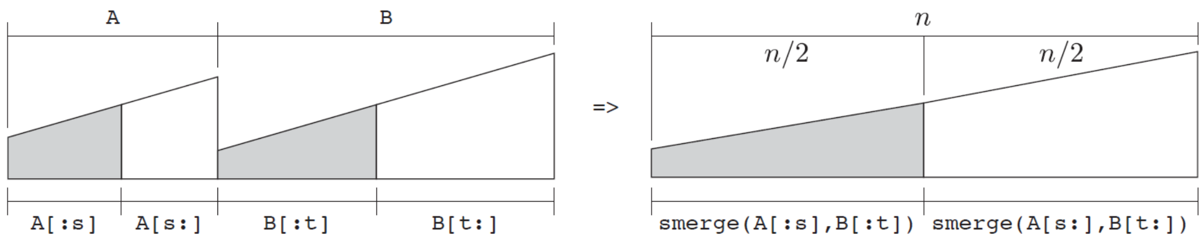
We assume that $T(n) = O(\log(\log n))$.
For any $m = \sqrt{n}$, we have $T(\sqrt{n}) \leq \log(\log \sqrt{n})$.
Therefore

$$\begin{aligned} T(n) &= T(\sqrt{n}) + 1 \\ &\leq \log(\log \sqrt{n}) + 1 \\ &= \log(\log \sqrt{n}) + \log 2 \\ &= \log(2 \log n^{\frac{1}{2}}) \\ &= \log(\log n) \end{aligned}$$

By using methematical induction, we know that $T(n) = O(\log(\log n))$

3: (7+6+6pts) Divide and Conquer Algorithm

Question 9. *In this problem, we will find an alternative approach to the merge step in Merge Sort named* ***Slice Merge*** *. Suppose A and B are sorted arrays with possibly different lengths, and let $n = len(A)+len(B)$. You may assume n is a power of two and all n elements have distinct value. The slice merge algorithm,* ***smerge(A,B)****, merges A and B into a single sorted arrays as follows:*



Step 1: *Find index s for subarray A and index t for subarray B ($s + t = \frac{n}{2}$) to form two prefix subarrays `A[:s]` and `B[:t]`, such that `A[:s]` $\cup$ `B[:t]` contains the smallest $\frac{n}{2}$ elements in all n elements of $A \cup B$.*

Step 2: *Recur for `X = smerge(A[:s], B[:t])` and `Y = smerge(A[s:], B[t:])` respectively to reorder and merge them. Return their concatenation $X + Y$, a sorted array containing all elements in $A \cup B$.*

For example, if $A = [1, 3, 4, 6, 8]$ and $B = [2, 5, 7]$, we should find $s = 3$ and $t = 1$ and then recursively compute:

`smerge([1, 3, 4], [2]) + smerge([6, 8], [5, 7]) = [1, 2, 3, 4] + [5, 6, 7, 8]`

1. *Describe an algorithm for Step 1 to find indices s and t in $O(n)$ time using $O(1)$ additional space. Write down your main idea briefly (or pseudocode if you would like to) and analyse the runtime complexity of your algorithm below. You may assume array starts at index 1.* ***(2pts)***

Algorithm 1 Step 1 of Slice Merge

function FINDSANDT(A, B)
  $half \leftarrow \lfloor (A.lenth + B.length)/2 \rfloor$
  $s \leftarrow 0$
  $t \leftarrow 0$
  for $j \leftarrow 1$ **to** $half$ **do**
    if $A[s+1] < B[t+1]$ **then**
      $s \leftarrow s + 1$
    else
      $t \leftarrow t + 1$
    end if
  end for
  return s, t
end function

Since $A.length + B.length = n$, the runtime complexity should be $O(2 \cdot \frac{1}{2}n) = O(n)$

2. *Write down a recurrence for the runtime complexity of* `smerge(A,B)` *when $A \cup B$ contains a total of n items. Solve it using the Master Theorem and show your calculation below.* ***(2pts)***
Note: Write your answer for time complexity in asymptotic order form i.e. $T(n) = O(g(n))$.

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

$$a = 2, b = 2, \log_b a = 1 = d$$

By Master Theorem, $T(n) = O(n \log n)$

3. *Recall the merge step* `merge(A,B)` *to combine two subarrays of length $n/2$ in the Merge Sort algorithm covered in our lecture slides. Compare the runtime complexity of* `smerge(A,B)` *with* `merge(A,B)`*.* ***(1pts)***

The runtime complexity of $merge(A, B)$ is $O(n)$, while the runtime complexity of $smerge(A, B)$ is $O(n \log n)$. Therefore $T_{smerge}(n) > T_{merge}(n)$.

4. *Replace* `merge(A,B)` *by* `smerge(A,B)` *in the merge stage of Merge Sort to develop a new sorting method namely* ***S-Merge Sort****. Write down a recurrence for the runtime complexity of S-Merge Sort. Solve it and show your calculation below.* ***(2pts)***
Note: Write your answer for time complexity in asymptotic order form i.e. $T(n) = O(g(n))$.

$$T(n) = 2T(n/2) + n \log n$$
$$a = 2, b = 2,$$
$$c^{crit} = \log_a b = 1 = d.$$

Since $f(n) = n \log n = n^{c^{crit}} \log n$, by Master Theorem,

$$T(n) = O\left(n \log^2 n\right)$$

Question 10. *There are n students in SIST and each student i has 2 scores A_i and P_i, score in Algorithms and Data Structures course and score in Probabilty and Statistics course respectively. Students i, j could form a mutual-help pair in CS101 class if and only if $A_i < A_j$ and $P_i > P_j$. How many possible mutual-help pairs (i, j) could be formed in CS101 class?*
Design an efficient algorithm to figure out this problem. For comparison, our algorithm runs in $O(n \log n)$ time. (Hint: how to count inversions?)

Note: Your answer should be consistent with the template we provide in Problem 0 Example.

First, sort the students according to A in ascending order using mergesort, which has been showed in the class. After that, sort the students again according to P in descending order. Meanwhile, count the inversions of P using merge sort. The inversions of P should be the answer. The complete algorithm is described as follows.

Algorithm 2 Question 10 CountInversionsByMergesort

```
function CountInversionsByMergesort(StudentArr, left, right, mid)
    count ← 0
    length ← right − left + 1
    tempArr ← an array of the type of the elements in StudentArr with length of length
    leftSubLength ← mid − left
    rightSubLength ← right − mid + 1
    leftptr ← 0
    rightptr ← 0
    tempptr ← 0
    while leftptr < leftSubLength and rightptr < rightSubLength do
        if StudentArr[left + leftptr].P > StudentArr[mid + rightptr].P then
            tempArr[tempptr] ← StudentArr[left + leftptr]
            leftptr ← leftptr + 1
            count ← count + rightSubLength − rightptr − 1
        else
            tempArr[tempptr] ← StudentArr[mid + rightptr]
            rightptr ← rightptr + 1
        end if
        tempptr ← tempptr + 1
    end while
    while leftptr < leftSubLength do
        tempArr[tempptr] ← StudentArr[left + leftptr]
        leftptr ← leftptr + 1
        tempptr ← tempptr + 1
    end while
    while rightptr < rightSubLength do
        tempArr[tempptr] ← StudentArr[mid + rightptr]
        rightptr ← rightptr + 1
        tempptr ← tempptr + 1
    end while
```

```
    for i ← 1 to length do
        StudentArr[left + i − 1] ← tempArr[i]
    end for
    return count
end function
```

Algorithm 3 Question 10 CountPairs

```
function CountPairs(StudentArr, left, right)
    if left >= right then
        return 0
    end if
    mid ← ⌊(right − left)/2⌋ + left
    countleft ← CountPairs(StudentArr, left, mid)
    countright ← CountPairs(StudentArr, mid + 1, right)
    count ← countleft + countright + CountInversionsByMergesort(StudentArr, left, right, mid)
    return count
end function
```

Algorithm 4 Quetion 10 Complete Algorithm

```
function Main(StudentArr)
    sort StudentArr according to A of each element in ascending order use Mergesort.
    answer ← CountPairs(StudentArr, 1, StudentArr.length)
    return answer
end function
```

$$T(n)_{CountPairs} = 2T(n/2)_{CountPairs} + n$$

By master Theorem,

$$\begin{gathered} a = 2 \\ b = 2 \\ \log_b a = 1 = d \\ T(n)_{CountPairs} = O(n \log n) \end{gathered}$$

Therefore,

$$\begin{aligned} T(n)_{Main} &= T(n)_{CountPairs} + T(n)_{Mergesort} \\ &= O(2n \log n) = O(n \log n) \end{aligned}$$

Question 11. *Suppose you are a teaching assistant for CS101, Fall 2077. The TA group has a collection of n suspected code solutions from n students for the programming assignment, suspecting them of academic plagiarism. It is easy to judge whether two code solutions are equivalent with the help of "plagiarism detection machine", which takes two code solutions* `(A,B)` *as input and outputs* `isEquivalent(A, B)` $\in$ {`True`, `False`} *i.e. whether they are equivalent to each other.*
TAs are curious about whether there exists a ***majority*** *i.e. an* ***equivalent class of size*** $> \frac{n}{2}$ *among all subsets of the code solution collection. That means, in such a subset containing more than $\frac{n}{2}$ code solutions, any two of them are equivalent to each other.*
Assume that the only operation you can do with these solutions is to pick two of them and plug them into the plagiarism detection machine. Please show TAs' problem can be sloved using $O(n \log n)$ invocations of the plagiarism detection machine.

Note: Your answer should be consistent with the template we provide in Problem 0 Example.

Find majority recursively, if the array has majority, record the element. Each step, seperate the array into two subarray, each contains $\frac{n}{2}$ elements.
Check whether two subarrays contains majority recursively. If both subarray has majority then the majority of the current array should be the majority of the subarray. If one of subarray has majority or they have different majority, then count the majority in another subarray to see if this element is the majority of the current array. This can be finished in $O(n)$. If neither of the array has majority, then the current array won't have majority.
If there's only one elment in the array, the majority should be that element.
The complete algorithm is described as follows.

Algorithm 5 Q11 CountElement

```
function CountElement(StudentSolution, element, left, right)
    count ← 0
    for i ← left to right do
        if isEquivalent(element, StudentSolution[i]) then
            count ← count + 1
        end if
    end for
end function
```

Algorithm 6 Q11 CountMajority

```
function CountMajority(StudentSolution, left, right)
    if left == right then
        hasMajority ← true
        major ← StudentSolution[left]
        return hasMajority, major
    end if
```

```
    mid ← ⌊(right − left)/2⌋ + left
    length ← right − left + 1
    hasMajorityLeft, majorLeft ← CountMajority(StudentSolution, left, mid)
    hasMajorityRight, majorRight ← CountMajority(StudentSolution, mid + 1, right)
    if hasMajorityLeft and hasMajorityRight and isEquivalent(majorLeft, majorRight) then
        hasMajority ← true
        major ← majorLeft
    else if hasMajorityLeft and CountElement(StudentSolution, majorLeft, left, right) > ⌊length/2⌋ then
        hasMajority ← true
        major ← majorLeft
    else if hasMajorityRight and CountElement(StudentSolution, majorRight, left, right) > ⌊length/2⌋ then
        hasMajority ← true
        major ← majorRight
    else
        hasMajority ← false
        major ← null
    end if
    return hasMajority, major
end function
```

Algorithm 7 Q11 Complete Algorithm

```
function Main(StudentSolution)
    hasMajority, major ← CountMajority(StudentSolution, 1, StudentSolution.length)
    return hasMajority
end function
```

The time complexity of $CountElement$ is $O(n)$. The time complexity of $CountMajority$ can be written as $T(n) = 2T(n/2) + O(n)$. By Master Theorem,

$$a = 2$$
$$b = 2$$
$$\log_b a = 1 = d$$
$$T(n) = O(n \log n)$$