(15)

1 Loop Invariant (15 Points)

The following algorithm is the combine step for merge sort. That is, let A be a sorted array of $n/2$ integers, and let B be a sorted array of $n/2$ integers. The algorithm outputs C which is an array of n elements consisting of the elements in $A \cup B$ in sorted order. For simplicity, assume n is even and that $A[n/2+1] = B[n/2+1] = \infty$.

Algorithm 1 MergeSortCombine(integer array A[1..n/2], integer array B[1..n/2])

```
a = b = 1
for all i = 1 to n do
    if A[a] < B[b] then
        m = A[a]
        a = a + 1
    else
        m = B[b]
        b = b + 1
    C[i] = m
return C
```

| 1 | 3 | 5 |

| 2 | 4 | 6 |

a = 1

1 → 1

2 -

| 1 | |

1. State a loop invariant for the for loop that is true in each iteration of the loop, and in the terminating iteration implies that the algorithm is correct. Briefly argue why the invariant indeed implies the correctness of the algorithm.

loop invarient :- after each iteration C[1...i] contains (a+b-2) sorted elements. ✓

Initialization :- before beginning of any loop there is 0 element in the C array and trivially it is sorted.

Maintainance: for each iteration, element A[a] and B[b] will be compared and smallest will be put in C. So, there must not be a case where there is one element that is placed in C but the other smaller element in A or ... so, loop actually produce C array.

Termination:- loop will terminate when ... as there is $\frac{n}{2}$ elements in ... B, so all the 'n' elements already. ✓

2. Prove that your loop invariant is true by induction.

Base case:- when loop is not started, C contains $\underset{(a)}{1}+\underset{(b)}{1}-2=0$ elements.

the first iteration, C will contains ... -2 = 1 or $\underset{(a)}{1}+\underset{(b)}{2}-2=1$ elements.

base case holds.

Inductive step:- Let us suppose after ... step there is (n-1) sorted elements. [assumption that for iteration < n our loop invarient is true]

that means, either a = ... or a = n+1 & b = n. [a+b-...

for the last iteration ... put the value in C order and, then in ... dependent m from ... value that means we have ... in C in sorted order

15

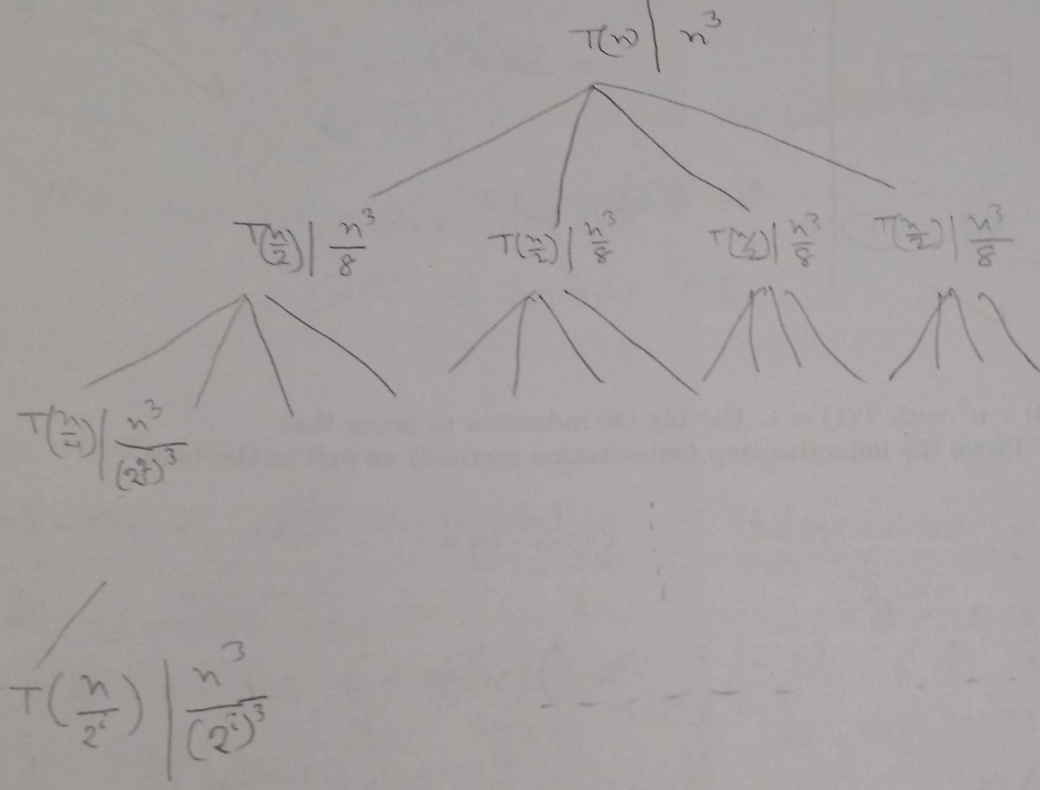
2 Recursion Tree (15 Points)

Ambelar Llahai One 1

Let $T(n) = 4T(n/2) + n^3$ with $T(1) = 1$. Use a recursion tree to generate a guess of what $T(n)$ solves to.

Total cost for the level

n^3



$4 \cdot \frac{n^3}{8} = \frac{1}{2} n^3$

$16 \cdot \frac{n^3}{64} = \frac{1}{4} n^3$

$4^i \cdot \frac{n^3}{(2^i)^3} = \frac{2^{2i} n^3}{2^{3i}}$

Total level $= \lg_2 n$

Total cost $= n^3 \left[\underbrace{1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^i}}_{} \right]$

even if we go to the infinity this sum converse to '2'.

$= \Theta(n^3)$

14

3 Asymptotic Growth (12 Points)

8/8

1. Show that $3n^3 + 4n - 6 \in \Theta(n^3)$.

$3n^3 + 4n - 6 \leq 3n^3 + 4n$

$\leq 3n^3 + 4n^3$ $\boxed{n \geq 1}$

$= 7n^3$

$= O(n^3)$ $\boxed{c = 7}$ --- (i)

$3n^3 + 4n - 6 \geq 3n^3$ when $4n - 6 \geq 0$

$\Rightarrow n \geq \frac{6}{4}$

$= \Omega(n^3)$ $c = 3$ --- (ii)

from (i) & (ii)

$3n^3 + 4n - 6 \in \Theta(n^3)$

6/7

2. Let $T(n) = 9T(n/3) + n^2$ with $T(1) = 1$. Use big-Oh induction to prove that $T(n) \in O(n^2 \log n)$. Prove the inductive step (substitution method) as well as the base case.

from big-O notation we know that

$T(n) \in O(n^2 \lg n) \Rightarrow T(n) \leq c \cdot n^2 \lg n$

for positive constant c.

Base case:

$n = 1$, $T(n) = 1$

now, $c \cdot n^2 \lg n = c \cdot 0 \; \overset{>}{<} \; 1$ needs to go other direction.

So, base case holds. X

Inductive step:

we assume that for $k < n$ our assumption true.

$\boxed{\therefore \; T(n) \leq c k^2 \lg k}$ --- (i)

now, $T(n) = 9\,T\left(\frac{n}{3}\right) + n^2$

$\leq 9 \cdot c \cdot \left(\frac{n}{3}\right)^2 \lg \frac{n}{3} + n^2$

$= \cancel{9} \cdot c \cdot \frac{n^2}{\cancel{9}} \lg \frac{n}{3} + n^2$

$= n^2 c \lg n - c n^2 \lg 3 + n^2$

$\leq c \cdot n^2 \lg n$ when, $n^2 - cn^2 \lg 3 \leq 0$

$= O(n^2 \lg n)$

(proved)

(15)

4 Master Method (15 Points)

Solve the following recurrences using the master theorem. Justify your answers shortly (i.e. specify ϵ and check the regularity condition if necessary).

1. $T(n) = 9T(n/3) + n \log n$

$a = 9$, $b = 3$, $n^{\lg_b a} = n^{\lg_3 9} = n^2$

$f(n) = n \lg n = O(n^{\lg_b a - \epsilon})$

what is ϵ?

case 1 apply, so, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$

2. $T(n) = T(n/2) + n$

$a = 1$, $b = 2$, $n^{\lg_2 1} = n^0 = 1$

$f(n) = n = \Omega(n^{0+1})$ for $\epsilon = 1$

Case III apply if we can satisfy the regularity condition which says that

$a f(\frac{n}{b}) \leq c \cdot f(n)$ for $c < 1$

now, $1 \cdot f(\frac{n}{2}) \leq c \cdot n$

$\Rightarrow \frac{n}{2} \leq cn$

$\Rightarrow c \geq \frac{1}{2}$

so, regularity holds for $c = 1/2$

$\therefore T(n) = \Theta(n)$

3. $T(n) = 8T(n/2) + n^3$

$a = 8$, $b = 2$, $n^{\lg_b a} = n^{\lg_2 8} = n^3$

✓

here, $f(n) = n^3 = \Theta(n^3)$

$T(n) = \Theta(n^3 \lg n)$

8

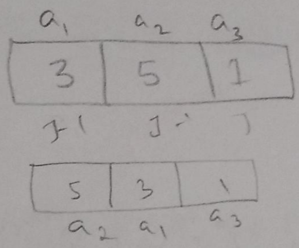
5 Decision Tree (15 Points)

Consider the following sorting algorithm known as *bubble sort*.

Algorithm 2 BubbleSort(integer array A[1..n])

for all $i = 1$ to n **do**
  for all $j = 2$ to $n - i + 1$ **do**
    if $A[j] < A[j-1]$ **then**
      Swap $A[j]$ and $A[j-1]$.

Draw the decision tree for BubbleSort on an input of size $n = 3$.



$i = 1,$

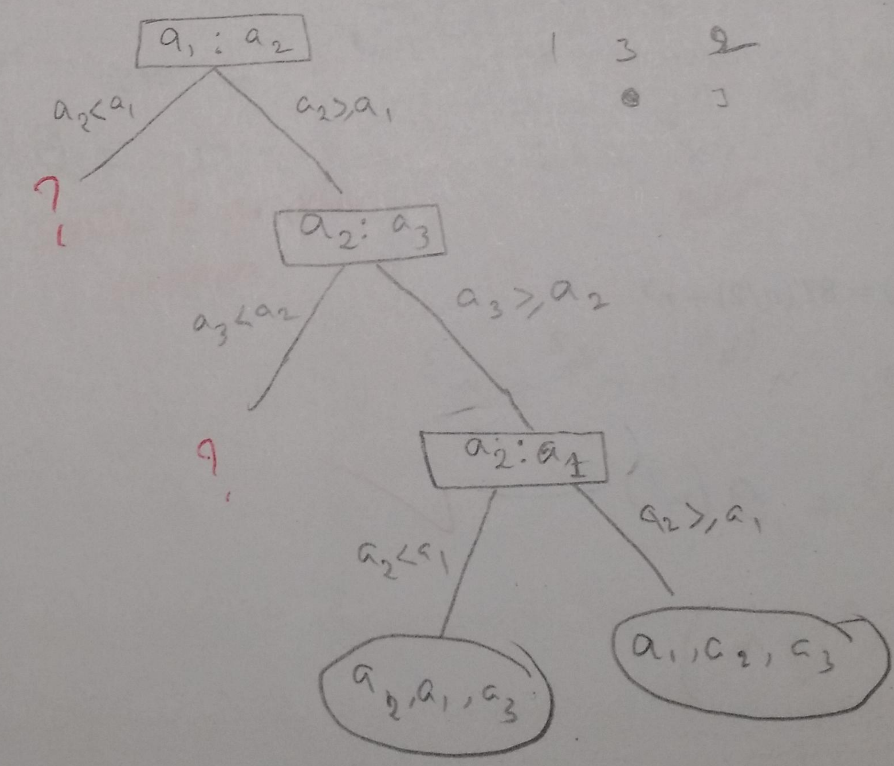
$J = 2 \ldots (n-1+1) = \boxed{2, 3}$

2, 1

$i = 2,$

$J = 2 - (n-2+1) = \boxed{2}$

Let denote the 3-inputs as a_1, a_2, a_3. the decisition ~~t~~ tree will look like



1 3 2

1

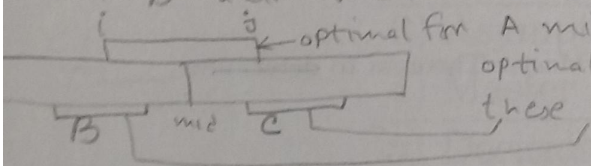
25

6 Divide and Conquer (25 Points)

Let A be an array of $n \geq 2$ integers (positive or negative). We are interested in computing indices l and r that minimizes the sum $\sum_{i=l}^{r} A[i]$. That is we want to compute the indices l and r such that the sum of integers between indicies l and r (inclusive) is minimized. Assume that A is indexed from 1 to n. For example, if $A = [1, 4, -2, -6, 7, -4, -5, 1, 3]$, then the optimal solution is $l = 3$ and $r = 7$ as the sum $\sum_{i=3}^{7} A[i] = -2 + -6 + 7 + -4 + -5 = -10$ and any other such sum will have a larger value. This problem can easily be solved in $O(n^2)$ time by checking all possibilities. This problem is about designing a divide and conquer algorithm for this problem with running time $o(n^2)$.

1. Suppose A is an array of size n where n is an even number, and let B denote the first $n/2$ elements of A and let C denote the last $n/2$ elements of A. Suppose we know an optimal solution for B and C. Give an $O(n)$ time algorithm that computes the optimal solution for A.

problem of finding optimal solⁿ for A from ...al solⁿ from B & C is that there might ...e case, the optimal solⁿ share some ... of B and some portion of C.



optimal for A might be optimal for these 3 case

...hen we try to find the optimal for A we need to somehow calculate ...r there is some other optimal solⁿ which share C & B.

we can use some $O(n)$ algorithm as given below -

```
Find-Min-Solⁿ-Shared-By-C&B (A, low, mid, high)
   left-sum ← +∞;
   sum ← 0;
   for i = 'mid' down to 'low'
       sum ← sum + B[i]
       if (left-sum > sum)
           left-sum ← sum;
           max-left = i
   right-sum ← +∞
   sum ← 0;
   for (j = mid+1 upto high)
       sum ← sum + C[j]
       if (right sum > sum)
           right-sum ← sum
           max-right ← j
   return (max-left, max-right, right sum + left sum)
```

and later on, we try to compare it to optimal of B & C and the return minimum of those

2. What is the base case for this problem? (note two more parts to this question are on the next page)

Base case is like when there is only 1 element then return that with its index as high and low also

3. Give the pseudocode for the algorithm. You can refer to your code above (i.e. you don't need to write it all over again, but do point out where it should go in the overall algorithm).

```
Find Min (A, low, high)
    if (low == high)
        return (low, high, A[low]);
    else  mid = ⌊(low + high)/2⌋
① (ll, lH, LS) = Find-Min(A, low, mid)
② (rl, rH, rS) = Find-min (A, mid+1, high)
③ (cl, cH, cS) = Find-Min-Soln-By-C&B(A, low, mid, high)
    if (1 < 2 & 1 < 3) return ①
    else if (2<1 & 2<3) return ②
    else return ③
```

4. What is the runtime relation for this algorithm? Use the Master Theorem to determine what it evaluates to.

recurrence will be, $T(n) = \begin{cases} 1 & , n = 1 \\ 2T(\frac{n}{2}) + n & , \text{otherwise} \end{cases}$

two sub problem   combine step

$a = 2,\ b = 2,\ n^{\lg_b a} = n^{\lg_2 2} = n$

$f(n) = n = \Theta(n^{\lg_2 2})$ $\boxed{\text{Case - II}}$

$\boxed{T(n) = (n \lg n)}$