Solution Divide and Conquer II

Fall 2022
Algorithms
Divide and Conquer HW 2

Learning objective is to analyze the time complexity of recursive divide and conquer algorithms.

Question	Full Marks	Marks Obtained
1	5	
2	Version 1 = 5 Version 2 = 5	
3	5 + 2 = 7	
Total	22	

Q1. Write recurrence relation for findMax method which is based on divide and conquer strategy. Derive its run-time using Masters Theorem. (2 + 3)

Ans: n will equal to (high – low). There are two subproblems and each subproblem is half the size.

Recurrence Relation is as follows:
$T(n) = 2T(n/2) + O(1)$

Combine or conquer cost is due to the leftMax, rightMax comparison in the end of the method findMax.

Give partial credit. Students may get the O(1) part wrong. You can give half credit in that case.

```
int findMax(int[] arr, int low, int high)
{
  if(low == high)
    return arr[low];

  int mid = low + (high-low)/2;

  int leftMax = findMax(arr, low, mid);

  int rightMax = findMax(arr, mid+1, high);

  if(leftMax > rightMax)
    return leftMax;
  else
```

Fall 2022
Algorithms
Divide and Conquer HW 2

Learning objective is to analyze the time complexity of recursive divide and conquer algorithms.

Question	Full Marks	Marks Obtained
1	5	
2	Version 1 = 5 Version 2 = 5	
3	5 + 2 = 7	
Total	22	

Q1. Write recurrence relation for findMax method which is based on divide and conquer strategy. Derive its run-time using Masters Theorem. (2 + 3)

Ans: n will equal to (high – low). There are two subproblems and each subproblem is half the size.

Recurrence Relation is as follows:
$T(n) = 2T(n/2) + O(1)$

Combine or conquer cost is due to the leftMax, rightMax comparison in the end of the method findMax.

Give partial credit. Students may get the O(1) part wrong. You can give half credit in that case.

```
int findMax(int[] arr, int low, int high)
{
  if(low == high)
    return arr[low];

  int mid = low + (high-low)/2;

  int leftMax = findMax(arr, low, mid);

  int rightMax = findMax(arr, mid+1, high);

  if(leftMax > rightMax)
    return leftMax;
  else
    return rightMax;
}
```

Applying Master theorem:
The recurrence has to be in standard form as follows:
$T(n) = aT(n/b) + f(n)$

For the problem, we have
$T(n) = 2T(n/2) + O(1)$

Calculate $n^{\log_b a}$
$a = 2$ and $b = 2$
$f(n)$ is $O(1)$

$n^{\log_b a} = n^{\log_2 2} = n$

$n^{\log_b a} > f(n)$ in this case, so we choose $n^{\log_b a}$ which is equal to n as time complexity.
So, anwer is O(n)

Give partial credit if student shows understanding of Masters Theorem, but gets the answer wrong.

Q2. Consider the following pseudocodes for calculating a^b, where a and b are positive integers.
Method FastMath1 and FastMath2
Input: positive integers a and b.
Output: $a^b = a*a* \ldots * a$ (multiplied b times)

Example: a = 3, b = 4, answer = 81

Version 1 **Time complexity is O(b)**	Version 2 **Time complexity is O(log b)**

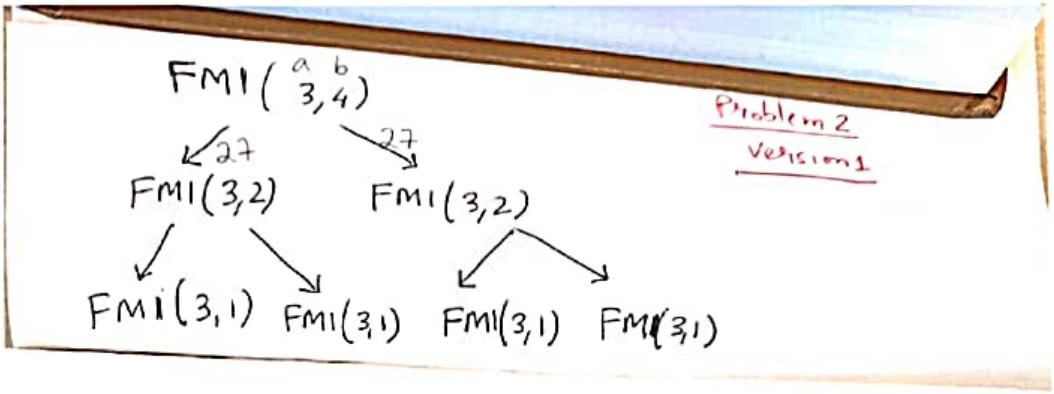
int FastMath1(int a, int b)
{
if b == 1 **then**
return a
else
int leftAns = FastMath1(a, b/2)
int rightAns = FastMath1(a, b/2)
 	
if b is odd **then**
return a* leftAns * rightAns
else
return leftAns * rightAns
}
	int FastMath2(int a, int b)
	{
	if b == 1 **then**
	return a
	else
	int c = a * a
	int ans = FastMath2(c, b/2)
	
	if b is odd **then**
	return a * ans
	else
	return ans
	}

Draw the recursion tree for both the methods and then analyze the time complexity of both versions using recurrence tree method and Masters Theorem.

Note: Give partial credit if student shows understanding of recursion tree and Masters Theorem, but gets the answer wrong. This is because a student may get merge/conquer cost wrong. This will make the final answer wrong. But, we want to give them partial credit for showing intermediate/incomplete work.

Students tree may look different than mine; depends on the amount of detail shown. If the answer is right, you can give them full credit.

See my answer below:



Problem 2
Version 1

FMI1
a=3, b=4
leftAns = 9
rightAns= 9
return 9x9 = 81

9
9

FMI 1
a=3, b=2
leftAns= 3
rightAns= 3
return 3x3 = 9

FMI 1
a=3, b=2
leftAns= 3
rightAns= 3
return 3x3 = 9

3
3
3
3

FMI 1
a=3, b=1
return 3

a=3, b=1
return 3

a=3, b=1
return 3

a=3, b=1
return 3

Time complexity of version 1 of FastMath1 method

Recurrence Tree Method

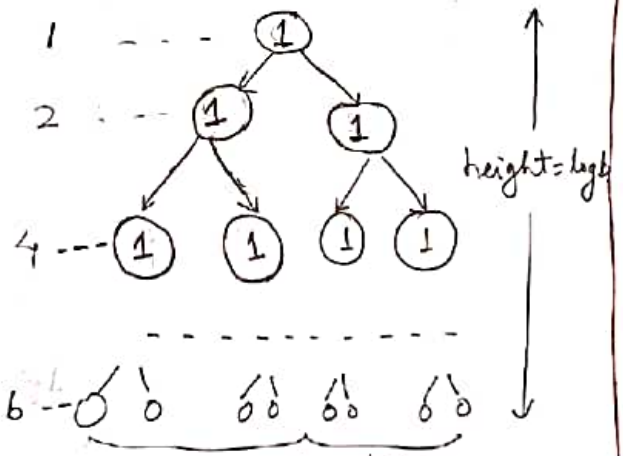
Version 1 (2 recursive calls)

$T(b) = 2\,T(\frac{b}{2}) + O(c)$

c is Constant

You can also write

$T(b) = 2T(\frac{b}{2}) + O(1)$



Sum of all levels

$= 1 + 2 + 4 + \cdots + \frac{b}{2} + b$

$\leq 2b$

$= O(b)$

Masters Theorem

Version 1 (2 recursive calls)

Standard Form

$T(n) = a\,T(\frac{n}{B}) + f(n)$ ⊛

$T(b) = 2\,T(\frac{b}{2}) + O(1)$

⊛ Here, instead of b, we use B to avoid confusion.

$b^{\log_B a} = b^{\log_2 2}$

$= b^1$

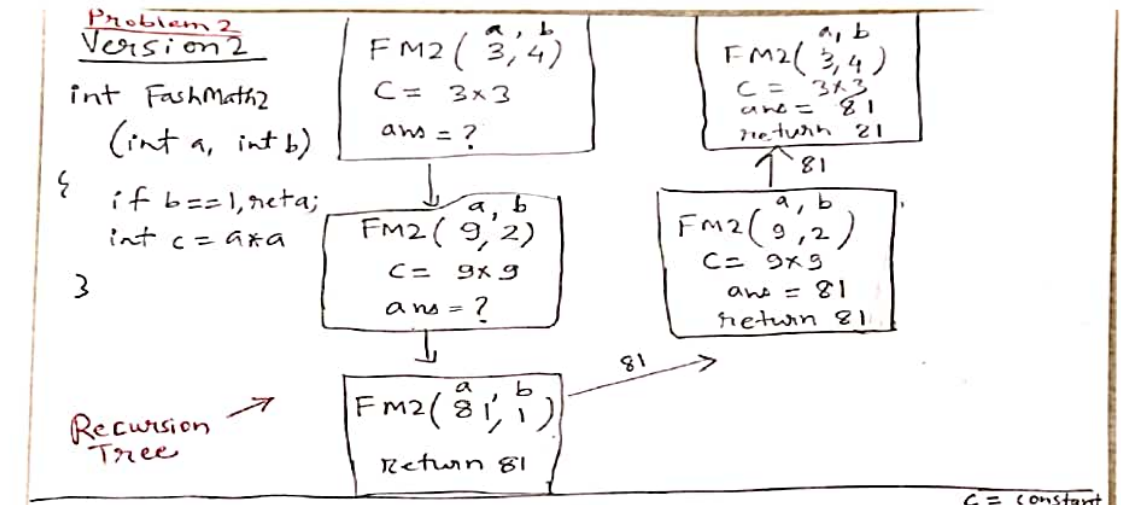
$= b$

In this case,

$b^{\log_B a} > f(n)$

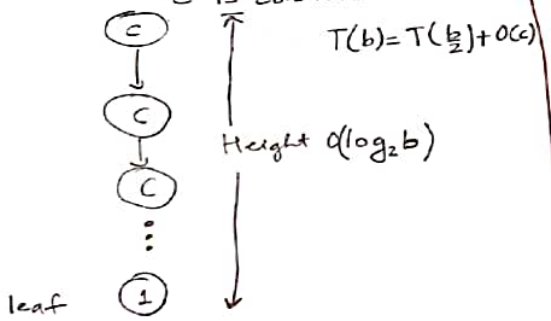
So, time Complexity $= O(b)$

Recurrence Tree and Time complexity of FastMath2 (version 2)



Recurrence Tree Method

c is constant



$T(b) = O(c * \log_2 b)$

Time complexity = $O(\log b)$

Masters Theorem

c = constant, b = B.

$T(B) = 1 \cdot T(\frac{B}{2}) + O(c)$

$a = 1 \quad f(n) = c$

$b = 2$

$T(n) = a \cdot T(\frac{n}{b}) + f(n)$

$n^{\log_2 1} \leq n^{\log_b a}$

$= n^0$

$= 1$

$f(n) = n^{\log_b a}$

So, time complexity = $1 \cdot \log b$

Case 2 Masters Theorem → $= O(\log b)$

Q3. Draw the recursion tree for the following recurrence relation where n is the input size of the problem:

$$T(n) = 2\,T(n/4) + n^2$$

Show the number of levels (height of tree), number of leaves and merge cost per level and the run time complexity for a program with this recurrence relation.

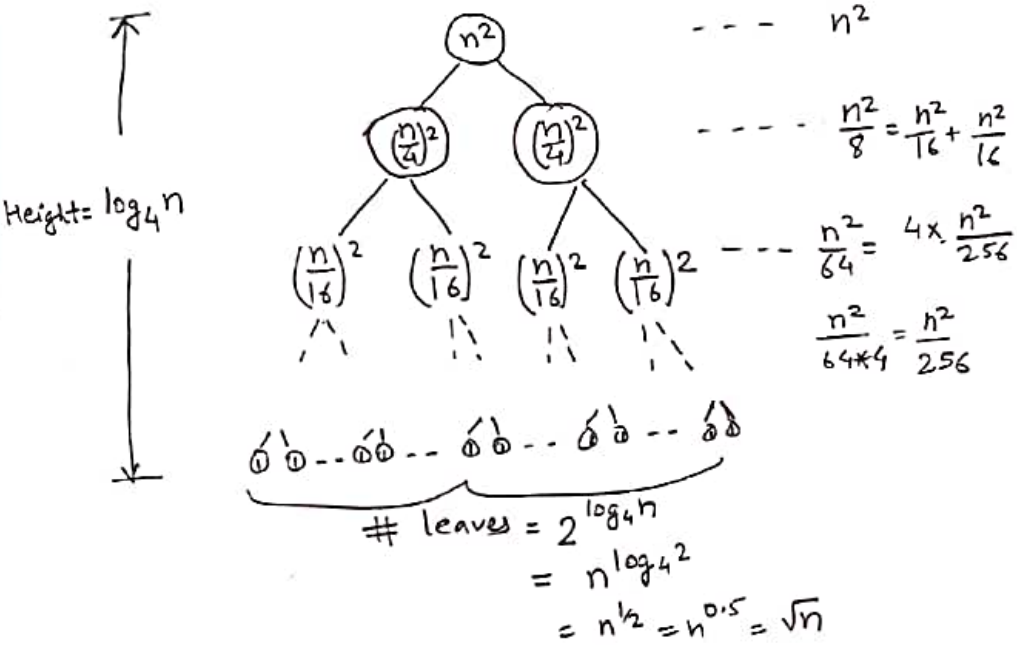
You have to draw a tree here first. The tree is shown on the next page.

The root has a cost of n^2 and this root has 2 children of cost $(n/4)^2$. Continuing recursively, the next level of the tree will have 4 children of cost $(n/16)^2$.

HW Problem 3

Draw recursion tree $T(n) = 2T(\frac{n}{4}) + n^2$

Two children of $\frac{1}{4}$ cost of the parent node

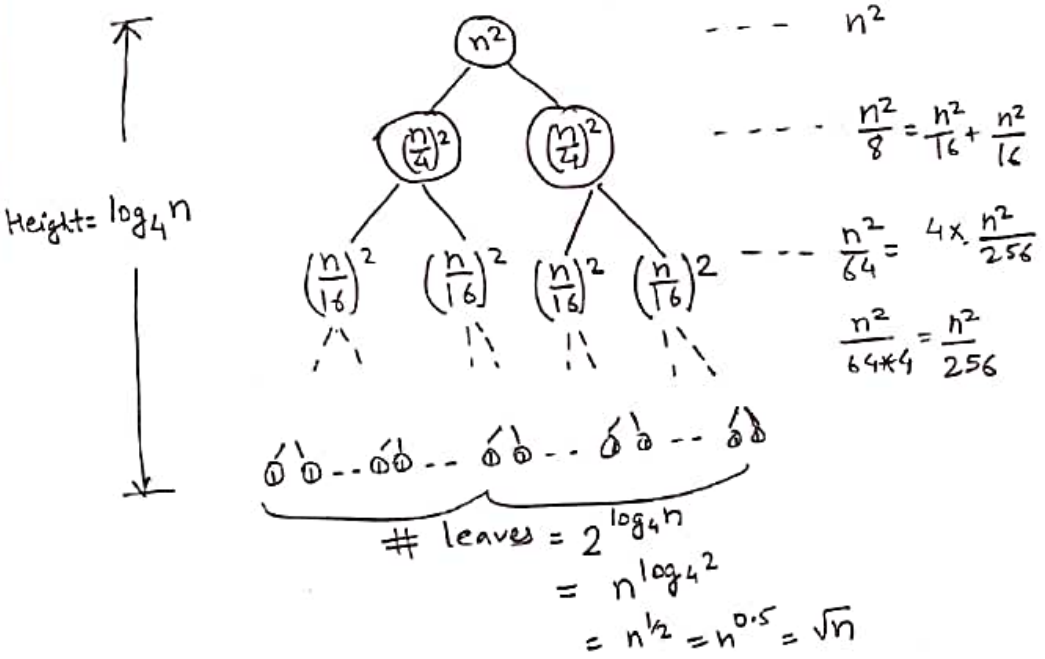


Merge cost at root (level 0) = n^2 1 node

HW Problem 3

Draw recursion tree $T(n) = 2T\left(\frac{n}{4}\right) + n^2$

Two children of $\frac{1}{4}$ cost of the parent node



leaves $= 2^{\log_4 n}$

$= n^{\log_4 2}$

$= n^{1/2} = n^{0.5} = \sqrt{n}$

Merge cost at root (level 0) = n^2 1 node

Merge cost at level 1 = $(n/4)^2 + (n/4)^2 = n^2/8$ 2 nodes
Merge cost at level 2 = $(n/16)^2 + (n/16)^2 + (n/16)^2 + (n/16)^2 = n^2/64$ => 4 nodes
Merge cost at level 3 = $8 * (n/64)^2 = n^2/512 = n^2/8^3$ => 8 nodes
...
Merge cost at leaf level = 1 + 1 + 1 + ... + 1 = square_root(n)

What is the height of this tree: $\log_4 (n)$

How many nodes are there at level 10 = 2^{10}
How many nodes are there at level log n = $2^{\log n}$

How many leaves will be there?
$2^{\log_4 (n)} = n^{\log_4 (2)} = n^{1/2}$ = square_root(n).

Total cost for all levels = $n^2 + n^2/8 + n^2/64 + ... + n^2/8^{\log n-1}$ + sqrt(n)
= $n^2(1 + 1/8 + (1/8)^2 + ... + (1/8)^{\log n-1})$ + sqrt(n)
We can apply "sum of geometric series" now on $(1 + 1/8 + (1/8)^2 + ... + (1/8)^{\log n-1})$.
x = 1/8

$$1 + 1/8 + (1/8)^2 + ... + (1/8)^{\log n-1} = \frac{1 - (1/8)^{\log n}}{1 - (1/8)}$$

To get an upper bound on the finite geometric series sum, we can use sum of infinite geometric series.
$1 + 1/8 + (1/8)^2 + ... + (1/8)^{\log n-1} < 1/(1 - 1/8)$
$= 8/7$
Total cost for all levels = $8/7\ n^2$ + sqrt(n)
= $O(n^2)$

Part 2) Find the time complexity of this problem using Masters Theorem.
a = 2
b = 4
$n^{\log_b a} = n^{\log_4 2} = n^{0.5}$

$f(n) = n^2$

$f(n) > n^{1/2}$ // asymptotically greater

So, time complexity is $O(n^2)$.

02-713 Homework #6: Divide and Conquer

Due: Mar. 29 by 9:30am

You may discuss these problems with your classmates, but you **must write up your solutions independently**, without using common notes or worksheets. You must indicate at the top of your homework who you worked with. Your write up should be clear and concise. You are trying to convince a skeptical reader that your answers are correct. Your homework should be submitted via Autolab (`https://autolab.cs.cmu.edu/02713-s13/`) as a typeset PDF.

1. Let $T = (V, E)$ be a tree with nodes V and edges E that is rooted at node r, and let $w(u)$ be the weight of node u. Recall that an independent set is a subset U of V such that no edge exists in between any two nodes in U. Give a polynomial time algorithm to compute the weight of largest weight Independent Set in T.

   Hint: Use divide & conquer: each node in the tree is the root of a subtree for which you you can solve a similar Independent Set problem.

   Hint$_2$: Consider the subproblems:

   - MaxIndepSetSize(u, `yes`): the size of the largest independent set in the subtree rooted at u, given that we select u.
   - MaxIndepSetSize(u, `no`): the size of the largest independent set in the subtree rooted at u, given that we **do not** select node u.

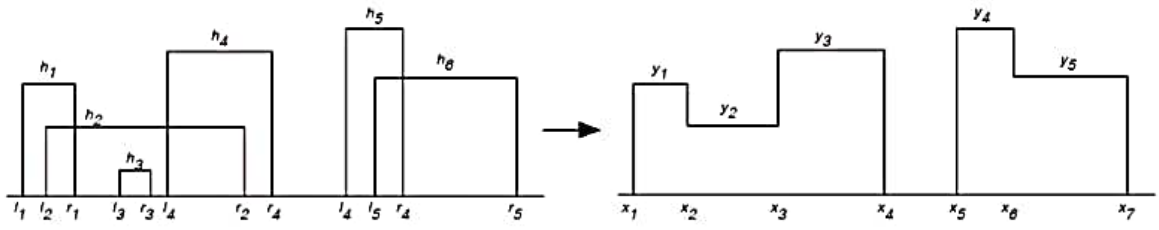
2. Let $x_1, \ldots, x_n$ be a set of integers. Give an $O(n)$ divide-and-conquer algorithm to find the largest possible sum of a subsequence of consecutive items in the list.

   Example: 10 -20 3 4 5 -1 -1 12 -3 1
   $\rightarrow$ 3 + 4 + 5 + -1 + -1 + 12 = 22

   Hint: Assume you can solve the problem for a list of $n-1$ or fewer items.

   Hint$_2$: As in counting inversions, try to do more than just solve the problem on the induction step: what other information do you need to go from a solution on a list of $n-1$ elements to a solution on a list of one additional element?

3. You want to draw a two-dimensional skyline like the following:



   You are given a list of the building x-coordinates and their heights:

$$(l_1, h_1, r_1), (l_2, h_2, r_2), \ldots, (l_n, h_n, r_n)$$

   This list will be sorted in increasing order of left-most x-coordinate. **Sketch** an $O(n \log n)$ algorithm to produce the skyline line to draw in the format:

$$x_1, y_1, x_2, y_2, x_3 \ldots$$

   meaning that at x_1 we draw a building at height y_1 until x_2 at which point we draw a line up or down to high y_2 and then continue horizontally until x_3 and so on.

   Note that for this problem, you need only sketch the main ideas, not provide as detailed a discussion as typically.