

PSO 1

Recurrences, divide and conquer

Exercise 1.1. Use recursion trees to solve the following recurrences.

1. $T(n) = 3T(n/3) + 6n$ and $T(1) = 2$.

2. $T(n) = 2T(n/3) + 4n$ and $T(1) = 7$.

3. $T(n) = 4T(n/3) + n$ and $T(1) = 11$.

Exercise 1.3. Problems 1–5 describe a search problem on a nonempty array $A[1..n]$ of comparable elements.⁷ For each of these problems, design a recursive algorithm that solves the search problem as fast as possible.⁸ Briefly analyze the running time of each algorithm.⁹

1. Let $A[1..n]$ be in a *rotated* sorted order. That is, there exists an index $i \in [n]$, called the *rotation index*, such that the concatenation of $A[i..n]$ and $A[1..i - 1]$ is sorted in increasing order. (One can think of $A[1..n]$ as being sorted initially, and then rotated cyclically to the right by $i - 1$ slots). The goal is to compute the unknown rotation index i . For simplicity, you may assume all the elements are distinct.

Exercise 1.7. Let $A[1..n] \in \mathbb{R}^n$ be an array of n numbers. An *inversion* is a pair of numbers out of increasing order; more precisely, a pair of indices $i, j \in [n]$ such that $i < j$ and $A[i] > A[j]$. Design and analyze an algorithm (as fast as possible) for counting the number of inversions in A .

- . Give a linear time algorithm taking as input a tree and returning a centroid. Your proof of correctness doubles as a proof that every tree has a centroid.

Exercise 2.2. For each of the recursive specifications below, design a recursive algorithm implementing the specification. (No proof or analysis is needed.)⁵

1. **subset-sum**(x_1, \dots, x_n, T): Given n integers $x_1, \dots, x_n \in \mathbb{Z}$, and an additional integer T , returns **true** if there is a subset of indices $S \subseteq [n]$ such that $\sum_{i \in S} x_i = T$, and **false** otherwise.

1. $T(n) = 3T(n/3) + 6n$ and $T(1) = 2$.

Recall the steps of a recursion tree

1. Draw out the tree
2. Find the cost at the i th level and the number of levels
3. Derive the sum and closed form

1. $T(n) = 3T(n/3) + 6n$ and $T(1) = 2$.

Each node tells us the *non-recursive* work done.
Start with the root as $T(n)$

$T(n): 6n$



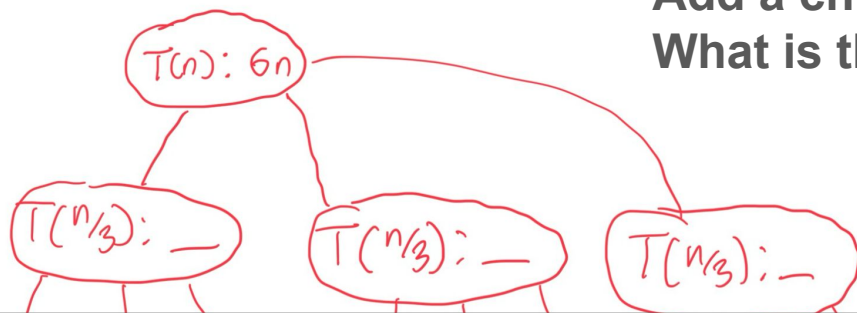
1. Draw out the tree
2. Find the cost at the i th level and the number of levels
3. Derive the sum and closed form

1. $T(n) = 3T(n/3) + 6n$ and $T(1) = 2$.

Each node tells us the *non-recursive* work done.

Add a child for each recursive call

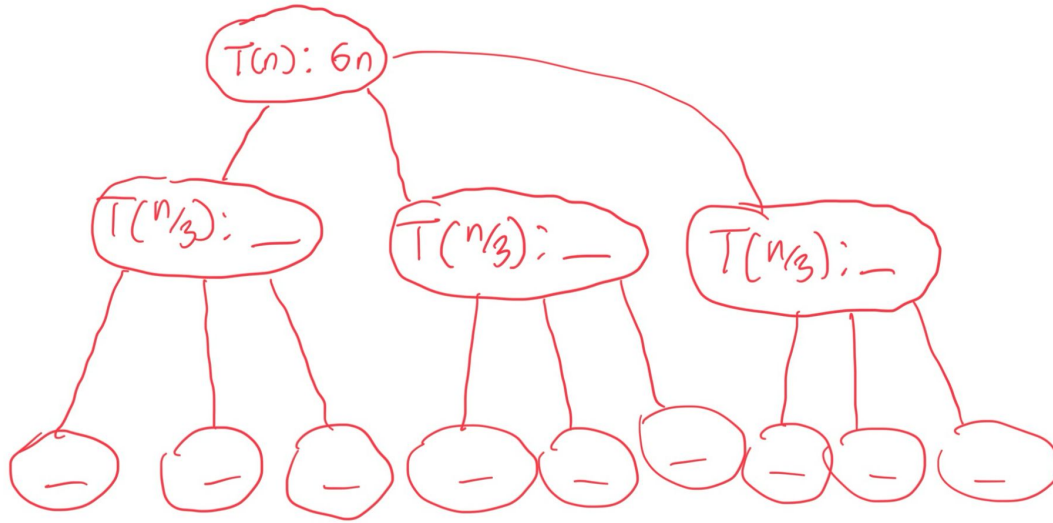
What is the non-rec. work done in each child?



1. Draw out the tree
2. Find the cost at the i th level and the number of levels
3. Derive the sum and closed form

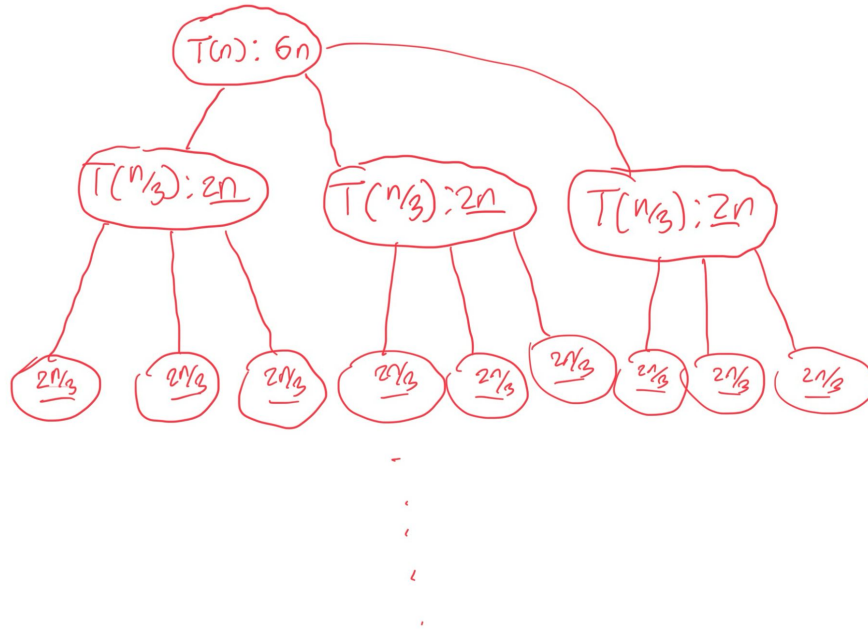
1. $T(n) = 3T(n/3) + 6n$ and $T(1) = 2$.

Each node tells us the *non-recursive* work done.



1. Draw out the tree
2. Find the cost at the i th level and the number of levels
3. Derive the sum and closed form

1. $T(n) = 3T(n/3) + 6n$ and $T(1) = 2$.



Cost at first level:

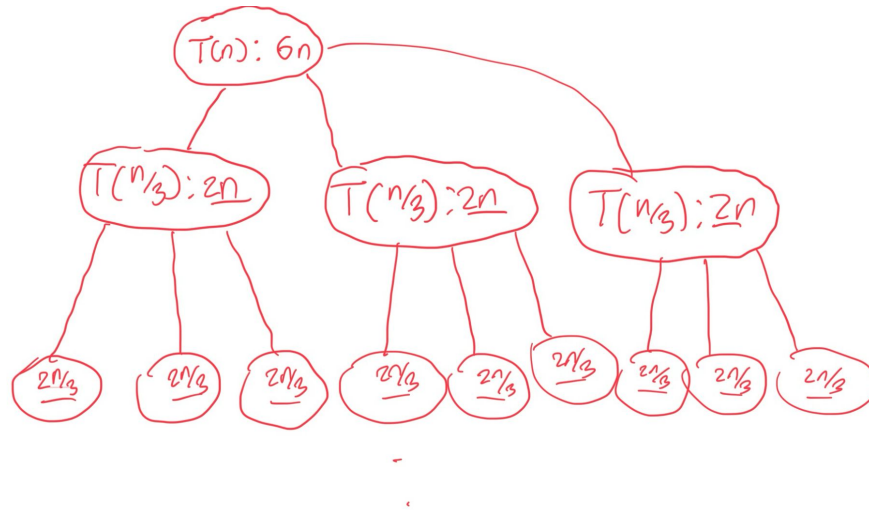
Cost at second level:

Cost at i th level:

1. Draw out the tree
2. Find the cost at the i th level and the number of levels
3. Derive the sum and closed form

levels:

1. $T(n) = 3T(n/3) + 6n$ and $T(1) = 2$.



C_i cost @ i th level : _____

N number of levels : _____

Solve sum:

$$\sum_{i=1}^N C_i = \underline{\hspace{2cm}}$$

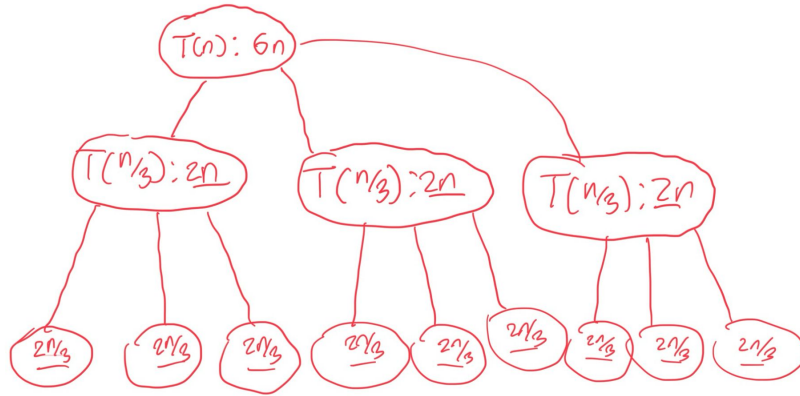
1. Draw out the tree
2. Find the cost at the i th level and the number of levels
3. Derive the sum and closed form

(big-O is fine)

Aside: this is the “merge sort” recurrence

For any constant c and C ,

$$T(n) = cT(n/c) + Cn \in O(n \log n)$$



Each level will sum to Cn !
(Here C was 6)

2. $T(n) = 2T(n/3) + 4n$ and $T(1) = 7$.

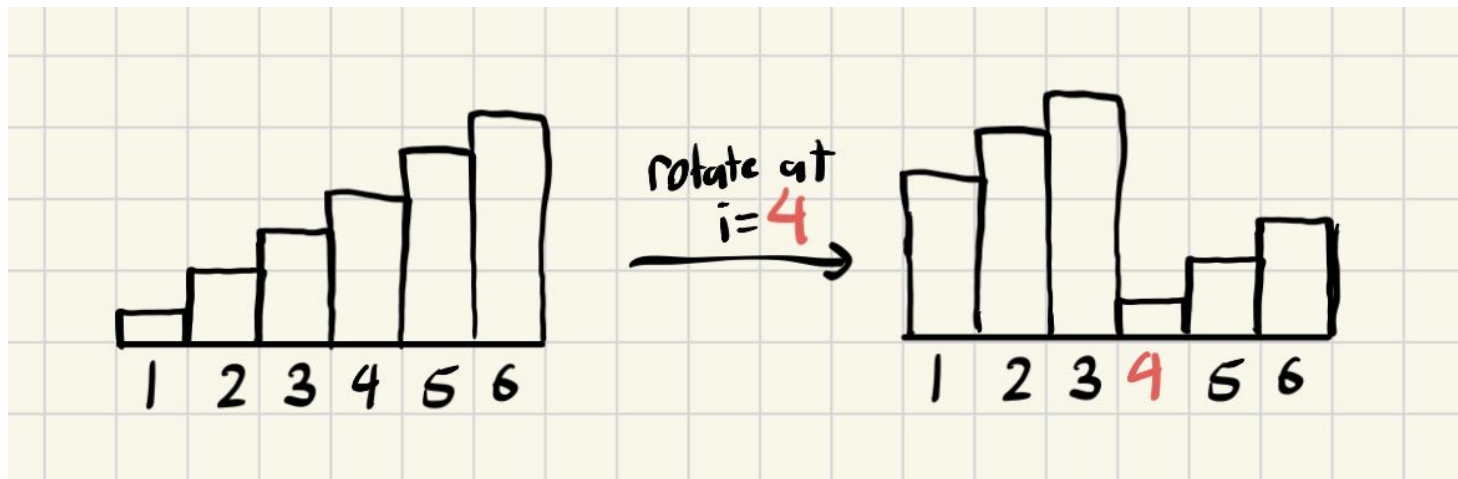
1. Draw out the tree
2. Find the cost at the i th level and the number of levels
3. Derive the sum and closed form

3. $T(n) = 4T(n/3) + n$ and $T(1) = 11$.

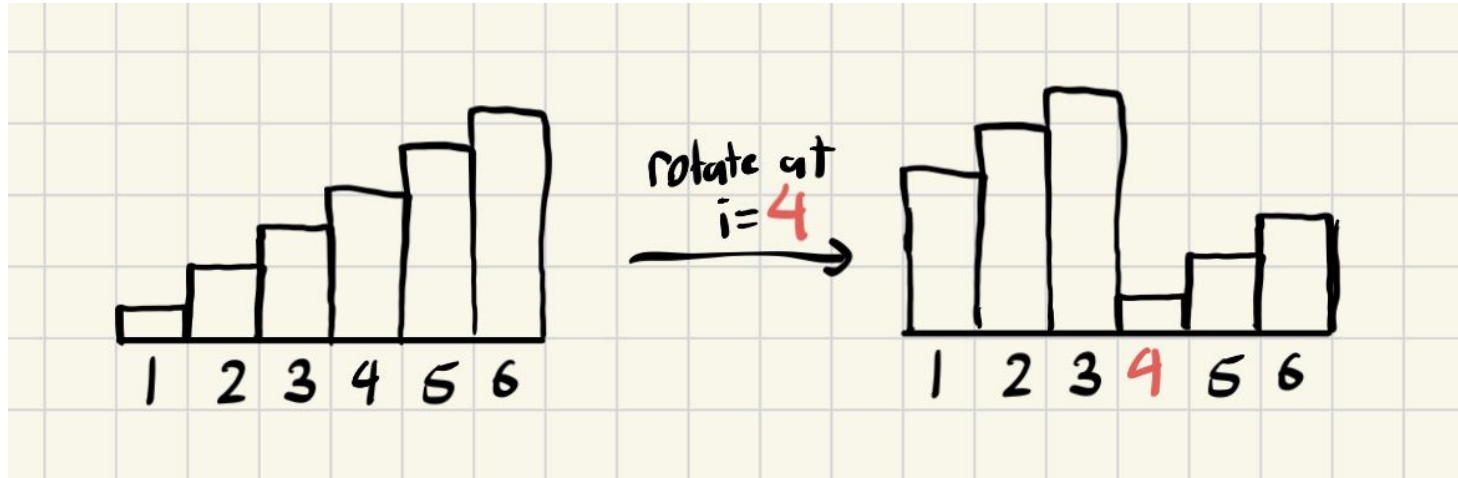
Exercise for you :)

1. Draw out the tree
2. Find the cost at the i th level and the number of levels
3. Derive the sum and closed form

1. Let $A[1..n]$ be in a *rotated* sorted order. That is, there exists an index $i \in [n]$, called the *rotation index*, such that the concatenation of $A[i..n]$ and $A[1..i-1]$ is sorted in increasing order. (One can think of $A[1..n]$ as being sorted initially, and then rotated cyclically to the right by $i-1$ slots). The goal is to compute the unknown rotation index i . For simplicity, you may assume all the elements are distinct.

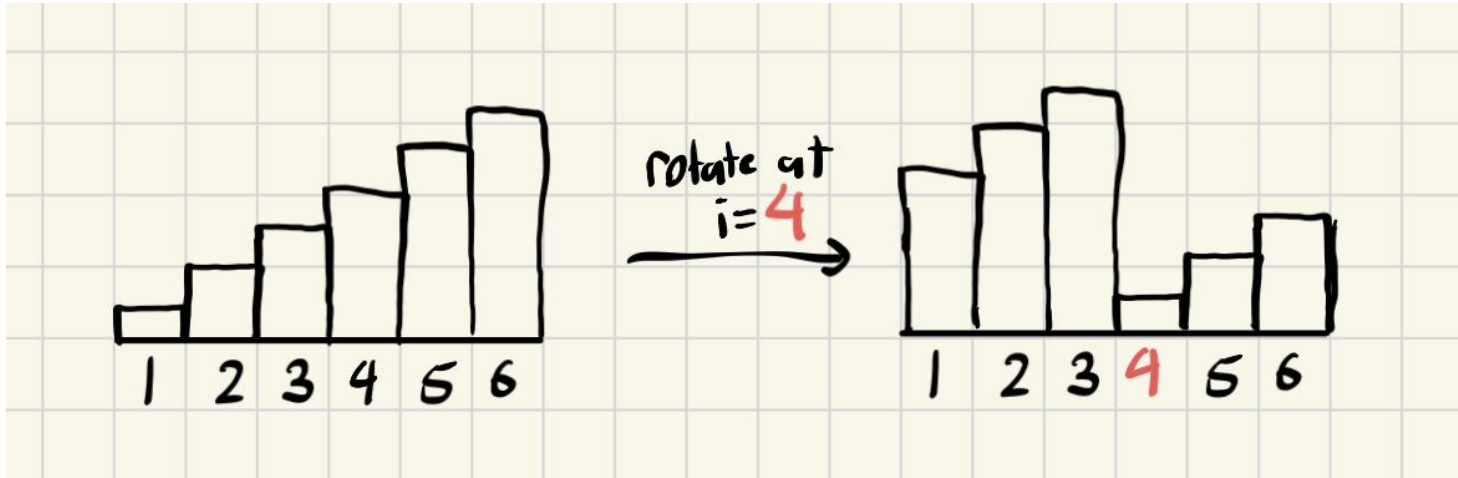


Goal: find index i given rotated array



Any ideas (maybe from lecture)? _____

Goal: find index i given rotated array



Any ideas (maybe from lecture)? **Binary search type question**

More generally, a recursive algorithm!

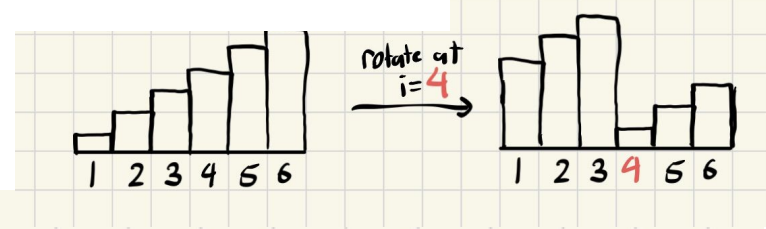
Steps to writing a recursive algorithm

1. Write the recursive spec
 - a. What is the input?
 - b. What is the output?
2. Use the recursive algorithm on smaller parts
3. Combine them appropriately

Example 1 : spec of merge sort

`merge-sort($A[1..n]$)`: Given an array $A[1..n]$ of comparable elements, returns an array consisting of the elements of A sorted in increasing order.

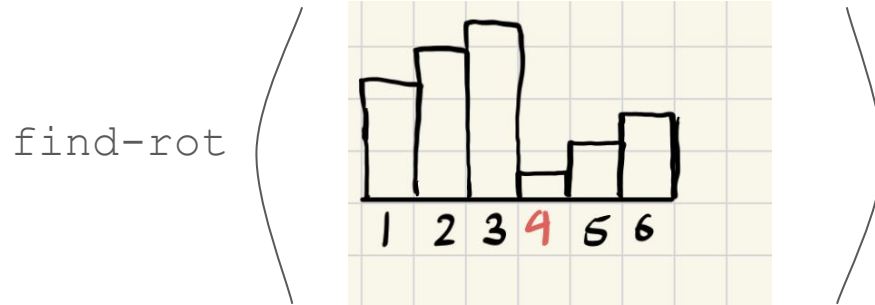
1. Let $A[1..n]$ be in a *rotated* sorted order. That is, there exists an index $i \in [n]$, called the *rotation index*, such that the concatenation of $A[i..n]$ and $A[1..i-1]$ is sorted in increasing order. (One can think of $A[1..n]$ as being sorted initially, and then rotated cyclically to the right by $i-1$ slots). The goal is to compute the unknown rotation index i . For simplicity, you may assume all the elements are distinct.



find-rot($A[1, \dots, n]$): Given _____ array A ,
(pre condition)
return _____,
(Post condition)

`find-rot(A[0...n-1]):` Given rotated-sorted
(pre condition) array A ,
return rotation index i
(Post condition).

1. Write the recursive spec
2. **Use the recursive algorithm on smaller parts**
3. Combine them appropriately



We want to binary search for index 4

$$\text{Middle index} = 6 / 2 = 3$$

How do we choose between recursing into left half or right half?
I.e., which half has the rotated index?

$\text{find_rot}(A[1, \dots, n])$: Given rotated-sorted
(pre condition) array A ,
return rotation index i
(Post condition).

1. Write the recursive spec
2. **Use the recursive algorithm on smaller parts**
3. Combine them appropriately

