**5.1**

1

A.

// Input an array arr[0 to n-1] with indexes between start and end where start <= end

// Output the largest index in the subset of arr[start:end]

LargestIndex(Arr, start, end)

If (start == end) return end

else

dE <- LargestIndex(Arr, start, floor((end + start) / 2))

dS <- LargestIndex(Arr, floor((end + start) / 2), end)

if (A[dE] >= A[dS] return dE

else return dS

B.

This algorithm should return the index of the leftmost largest element.

C.

The recurrence for the number of index comparisons should be C(n) = C(ceil(n/2)) + C(floor(n/2)) + 1, which would make C(1) = 0. Solving this through backwards substitution with n = 2^k provides the following results:

C(2^k) = 2C(2^k-1) + 1

= 2[2C(2^k-2) + 1 + 1

= 2^2 2C(2^k-3) + 1 + 2 + 1

= 2^i C(2^k-i) + 2^i-1 + 2^i-2 + … + 1

= n - 1

This verifies that C(n) = n - 1. Next, we need to consider the even and odd cases for all n > 1 since C(1) = 0. Given an even n = 2i and odd n = 2i-1 where i > 0, we can calculate the left and right sides of the recurrence. For the right it would be C(n) = 2i-1 which is equivalent to the left hand side. This allows us to verify that C(n) = n - 1.

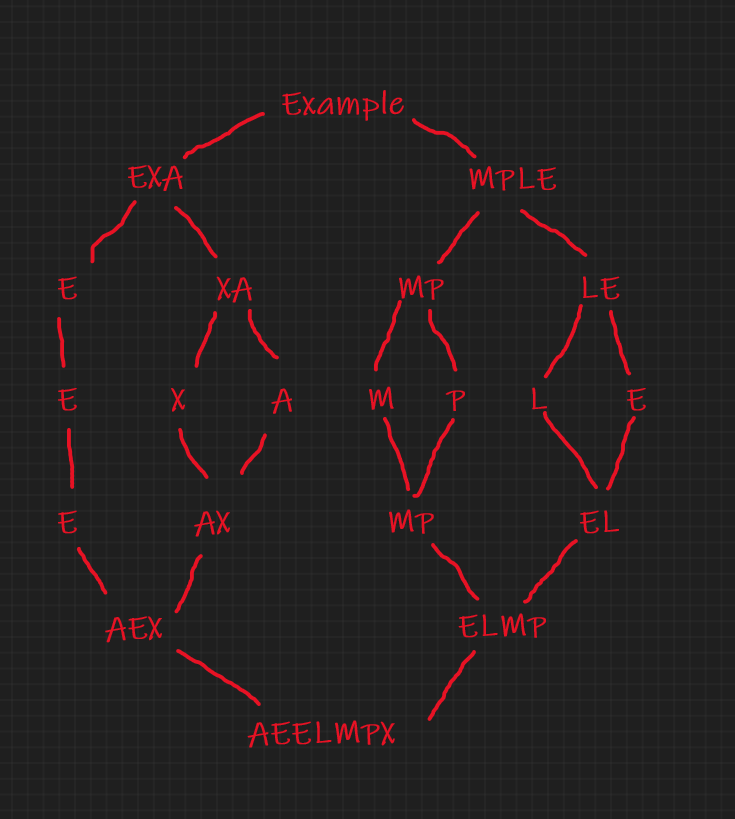
5

A. The growth order of T(n) = 4T(n/2) + n would be, through the application of the master theorem, theta(n^2) where a = 4, b = 2, and d = 1.

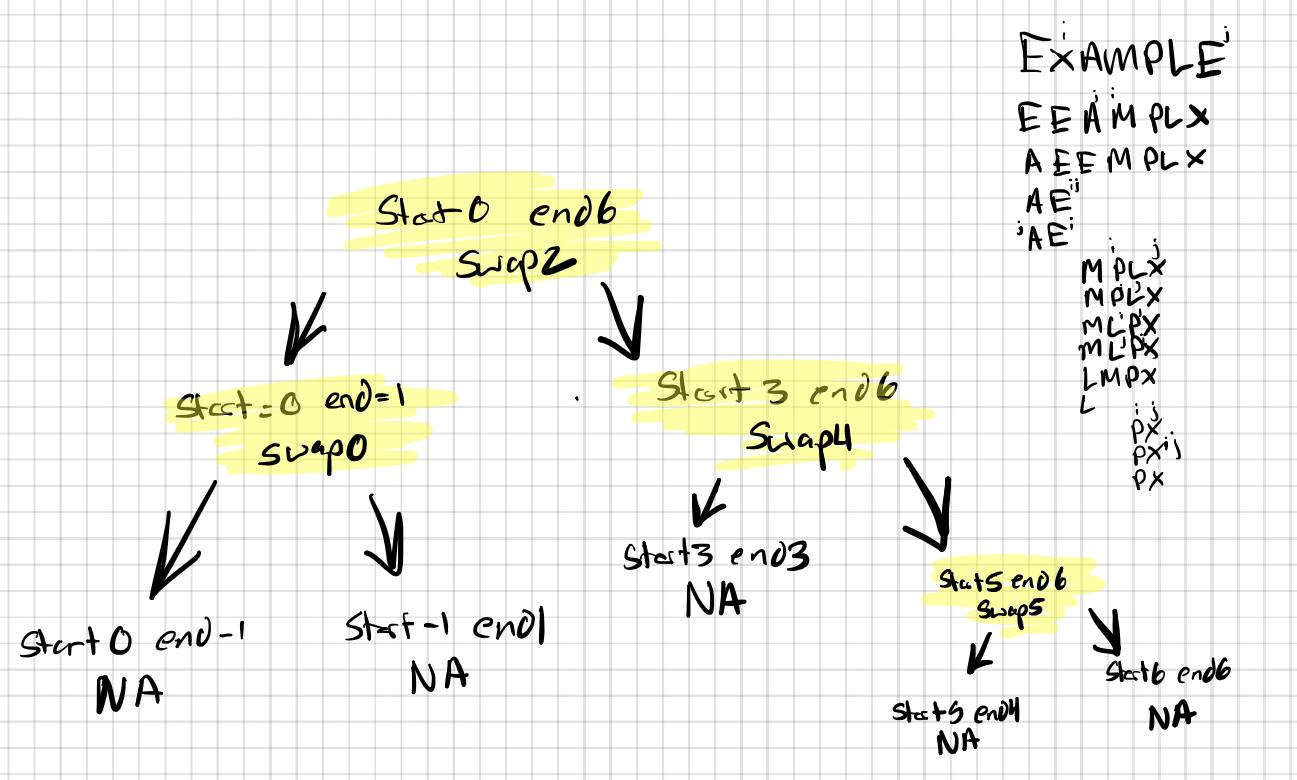
B. The growth order of T(n) = 4T(n/2) + n^2 would be, through the application of the master theorem, theta(n^2 log n) where a = 4, b = 2, and d = 2.

C. The growth order of T(n) = 4T(n/2) + n^3 would be, through the application of the master theorem, would be theta(n^3) where a = 4, b = 2, and d = 3.

6.



**5.2**

1. 

(i highlighted all the spots that had swaps)

2. For the partitioning procedure outlined in this section:

A. Prove that if the scanning indices stop while pointing to the same element, i.e., i = j the value they are pointing to must be equal to.

If p is the pivot’s value,

for i going left-to-right it stops at A[i] >= p, and j going right-to-left A[j] <= p so if i = j then A[i] = A[j] = p

B. Prove that when the scanning indices stop, j cannot point to an element more than one position to the left of the one pointed to by i.

If i is the value of the left to right scanner, since A[i-1] <= p, then the right to left scanner has to stop before passing i-1.

5.

A. Arrays that are composed of equal elements allow for the best case because all of the splits will occur in the middle of the subarrays, and then those subarrays.

B. they are the worst case because all spits will yield one empty subarray

11. I think it's like quicksort but I am not sure, we found an answer online that we would like to cite for half-credit. Link: [Solved: Nuts and bolts You are given a collection of n bolts of... | Chegg.com](https://www.chegg.com/homework-help/nuts-bolts-given-collection-n-bolts-different-widths-n-corre-chapter-5.2-problem-11e-solution-9780132316811-exc)

**5.3**

2. This algorithm is not correct because the base case always returns zero for any binary tree. To fix the algorithm, we can add a check if each left and right node is null, as during that case we’re on a leaf node and should return 1.

LeafCounter(T)

if T = NULL return 0

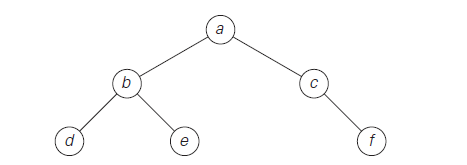
else if T->left = NULL and T->right = NULL return 1

Else return LeafCounter(T->left) + LeafCounter(T-right)

3.

If you traverse a binary tree with breadth-first search to find the level of each node, the largest level is the equivalent to the height of the tree.

5.



1. Preorder: a b d e c f
2. Inorder: d b e a c f
3. Postorder: d e b f c a

11. Since we can only break the bar in a straight line, we can use a binary tree structure to work through this puzzle. Our starting node in the tree will be the whole chocolate bar. We can break the bar in half and put half the bar as the left node and half the bar as the right node. With these two new nodes, we repeat this process until all of the nodes represent 1x1 pieces and do not have left or right nodes. The number of leaf nodes in the amount of single pieces we’re left with. This will take nm - 1 cuts to get nm pieces.

Given a chocolate bar n=2, m=8

2x8

2x4 2x4 1 cut

2x2 2x2 2x2 2x2 2 cuts

2x1 2x1 2x1 2x1 2x1 2x1 2x1 2x1 4 cuts

1x1 1x1 1x1 1x1 1x1 1x1 1x1 1x1 1x1 1x1 1x1 1x1 1x1 1x1 1x1 1x1 8 cuts

15 total cuts and 16 total pieces

**5.4**

1

The smallest number of digits that can be produced is 2n-1

The largest number of digits the can be produced is 2n

2.

**2101 \* 1130**

A1 = 21 A2 = 01 B1 = 11 B2= 30

c2 = 21 \* 11 = **R1**

c0 = 01 \* 30 = **R2**

c1 = **(22) \* (41)** - (c2 + c0) = (22) \* (41) - c2 - c0

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**21\*11**

c2 = 2 ∗ 1 = 2

c0 = 1 ∗ 1 = 1

c1 = (2 + 1) ∗ (1 + 1) − (2 + 1) = 3 ∗ 2 − 3 = 3

2 \* 102 + 3 \* 10 + 1 = **231 -> R1**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**01 \* 30**

c2 = 0 ∗ 3 = 0

c0 = 1 ∗ 0 = 0

C1 = (0 + 1) ∗ (3 + 0) − (0 + 0) = 1 ∗ 3 − 0 = 3

0 \* 102 + 3 \* 10 + 0 = **30 -> R2**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**22 \* 41**

c2 = 2 ∗ 4 = 8

c0 = 2 ∗ 1 = 2

c1 = (2 + 2)∗ (4 + 1) − (8 + 2) = 4 ∗ 5 − 10 = 10

8 · 102 + 10 · 10 + 2 = **902**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2101 \* 1130 = 231 \* 104 + 641 \* 102 + 30 = 237130**