CSC 3110
May Wandyez
Gq5426
Homework 4

CSC 3110

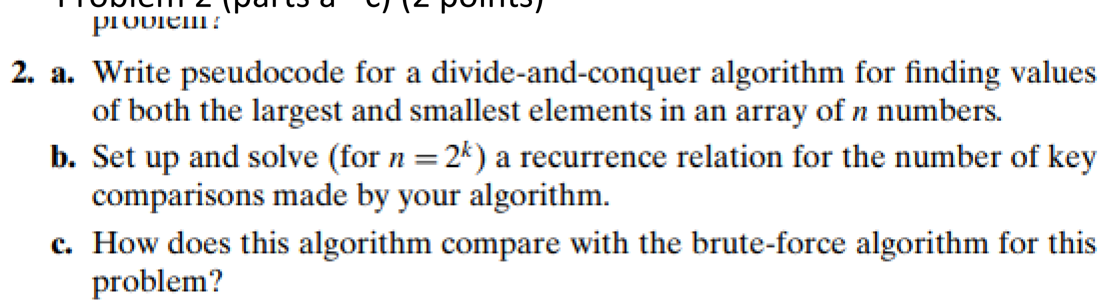
Algorithm Design and Analysis

(30 points)

Due: 03/18/2024 11:59 PM

Note: Submit answers in PDF document format. Please read the submission format for appropriate file naming conventions.

1) Exercises 5.1:
   a. Problem 2 (parts a - c) (2 points)



problem?

2. a. Write pseudocode for a divide-and-conquer algorithm for finding values of both the largest and smallest elements in an array of n numbers.

b. Set up and solve (for $n = 2^k$) a recurrence relation for the number of key comparisons made by your algorithm.

c. How does this algorithm compare with the brute-force algorithm for this problem?

a.) Dividererer(ArrayIn,Left,Right,Min,Max,WhatWeDoing):
/* this function will divide an array by splitting it down the middle into two smaller subarrays and return the min and max val found respectively until the max and min are found. Since the comparisons are made at the bottom, the max and min function will always be returned*/

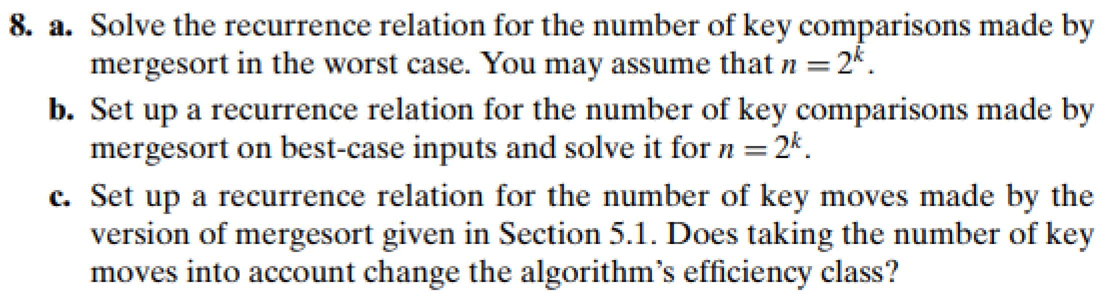
```
If r=l
        /this is the case where the sides have made the array size one
        If A[r] < minval:
                Minval = A[r]
        If A[r] > maxval:
                Maxval = A[r]
If r-l =1:
/case where the two sides are an even array
        If A[l] < minval:
                Minval = A[l]
        If A[l] > maxval:
                Maxval = A[l]

        If A[r] < minval:
                Minval = A[r]
        If A[r] > maxval:
                Maxval = A[r]
```

```
If r-l > 1:
    Dividererer(ArrayIn,Left, (Left+Right)/2, Min,Max )
    Dividererer(ArrayIn,(Left+Right)/2+1, Right, Min,Max,)
```

b.)The number of key comparisons for the recurrence relation is.
Key(n) = 2*Key(n/2)
Key(1) = 4
Key(0) = 2

c.) Brute force would require 2n worst case comparison s(two calculations made for every element in the array). The divide and conquer approach requires MORE calculations because it requires a division to occur for every 2 elements in the array until the array is down to one item, and comparison of that division to the current minimum and maximum, which is roughly 4n, divide and conquer is by far worse.

b. Problem 8 (parts a - c) (2 points)


8. a. Solve the recurrence relation for the number of key comparisons made by mergesort in the worst case. You may assume that $n = 2^k$.

b. Set up a recurrence relation for the number of key comparisons made by mergesort on best-case inputs and solve it for $n = 2^k$.

c. Set up a recurrence relation for the number of key moves made by the version of mergesort given in Section 5.1. Does taking the number of key moves into account change the algorithm's efficiency class?

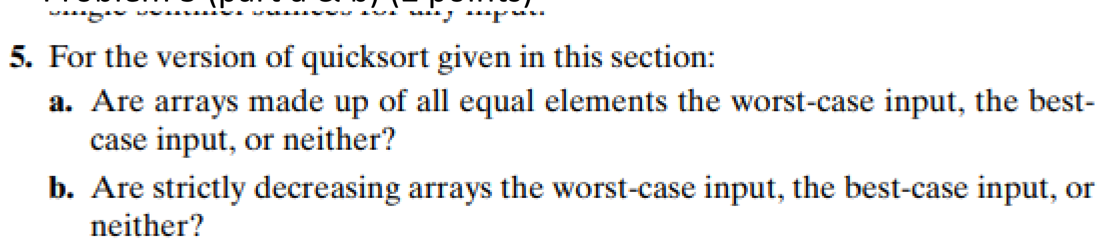
a.) Key(n) = 2Key(N/2) + n – 1 for n>1 and n = 2^k. Key(1) = 0
Looks like master theorem to me.
A = 2
B =2
K = 1
2 = 2
Key(n) = N*log(n)

b.) Best case inputs means the list is already sorted.
2Key(N/2) + n/2
Still is nlogn for the master theorem since the values for a b and k don't change. Might be less though. Googling it showed that it is exactly (½)n log n, but despite it making half as many comparisons it is still of type nlogn which is all I care about.
c.) There is the same number of key moves for the given algorithm, as the given recurrence relation is key(n) = 2(key(n/2)) + 2(n) for n>1, key(1)=0

CSC 3110
May Wandyez
Gq5426
Homework 4

2) Exercises 5.2:

a. Problem 5 (part a & b) (2 points)



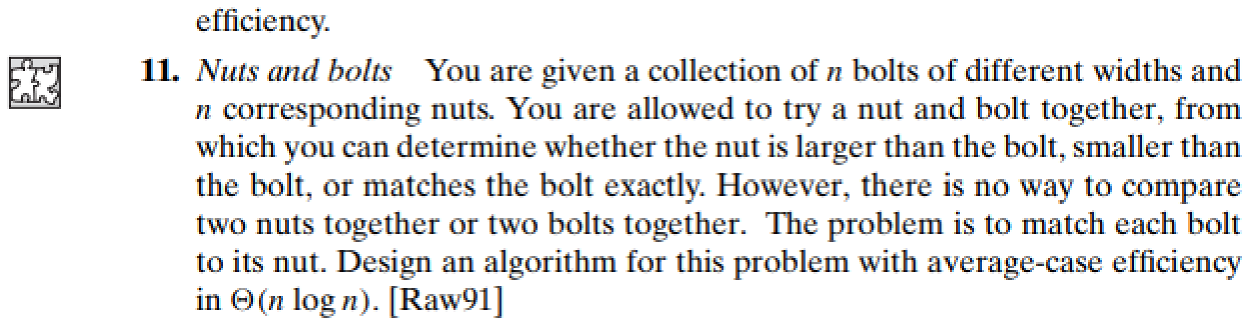
5. For the version of quicksort given in this section:

a. Are arrays made up of all equal elements the worst-case input, the best-case input, or neither?

b. Are strictly decreasing arrays the worst-case input, the best-case input, or neither?

a.) If all the elements are equal, then all the divisions occurring in quick sort will happen in the middle. This is the best case as it means there will be the least number of divisions.
b.) Strictly decreasing is the worst case scenario for a quicksort, as it means that the divisions will create an empty subarray (which by the way does NOTHING and wastes our time).

b. Problem 11 (2 points)



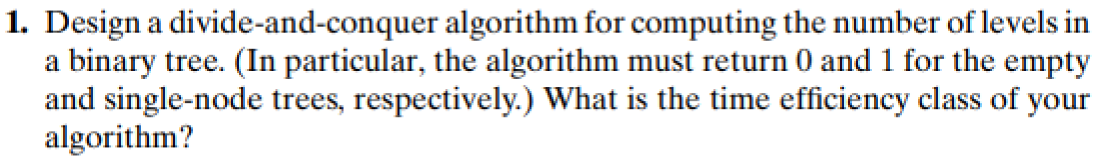
efficiency.

11. *Nuts and bolts* You are given a collection of n bolts of different widths and n corresponding nuts. You are allowed to try a nut and bolt together, from which you can determine whether the nut is larger than the bolt, smaller than the bolt, or matches the bolt exactly. However, there is no way to compare two nuts together or two bolts together. The problem is to match each bolt to its nut. Design an algorithm for this problem with average-case efficiency in $\Theta(n \log n)$. [Raw91]

Good news, we know if a nut is larger or smaller or matches the bolt, which means that when we begin making comparisons for a nut we can sort all other nuts into one of two sets. Also good news, if for the same nut we know that it is bigger than a bolt, we can sort the bolts as well based on the nuts. This basis of partitioning looks like quicksort since it effectively is creating pivots in the middle of each set of nuts and bolts. As we call, quick sort's efficiency is nlogn. Which means we did it. Yay!

3) Exercises 5.3:

a. Problem 1 (2 points)

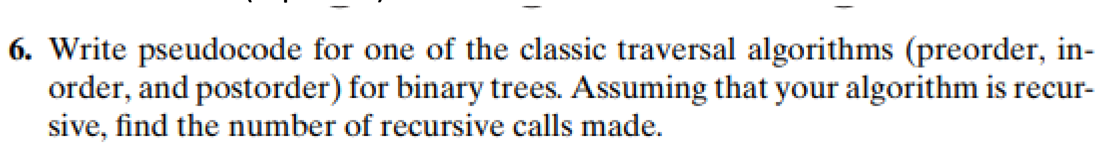


1. Design a divide-and-conquer algorithm for computing the number of levels in a binary tree. (In particular, the algorithm must return 0 and 1 for the empty and single-node trees, respectively.) What is the time efficiency class of your algorithm?

```
//this algorithm starts at the root of the tree fed in.
TotalLevels(RootIn):

If T = null:
return 0
Else: return(max(RootInLeft,RootInRight) +1)
```

This algorithm has to consider every single node in a tree, it cannot skip any, so as a result it has a time of O(n).

b. Problem 6 (2 points)

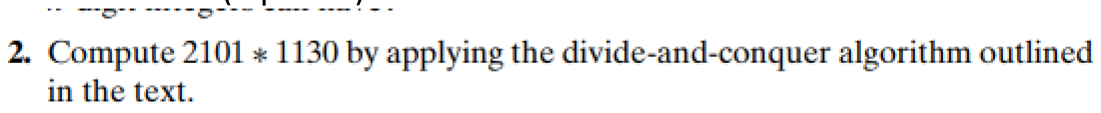

6. Write pseudocode for one of the classic traversal algorithms (preorder, inorder, and postorder) for binary trees. Assuming that your algorithm is recursive, find the number of recursive calls made.

PreOrderTraversal is Root,Left,Right. I will literally never remember this well enough to NOT write it down.

```
PreorderTraversal(Root):
    If Root != null:
        Print(Root)
        PreOrderTraversal(Left)
        PreOrderTraveral(Right)
```

4) Exercises 5.4:

a. Problem 2 (2 points)


2. Compute 2101 ∗ 1130 by applying the divide-and-conquer algorithm outlined in the text.

I get why we learn this, but this seems like something we don’t really need to know.

21*11
01*30
Break it down further
(2*1)*(1+1)– (2+1) = 3
2*10^2 + 3*10^2 +1 = 231

(0+1)*(3+0) – (0+0) = 3
0*10^2 + 3*10^1 + 0 = 30

(2+2)*(4+1)-(8+2) = 10
8*10^2 + 10*10^1 + 2 = 902
231*10^4 + (902-321-30) *10^2 = 2374130

CSC 3110
May Wandyez
Gq5426
Homework 4

b. Problem 7 (2 points)

7. Apply Strassen's algorithm to compute

$$\begin{bmatrix} 1 & 0 & 2 & 1 \\ 4 & 1 & 1 & 0 \\ 0 & 1 & 3 & 0 \\ 5 & 0 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 4 \\ 2 & 0 & 1 & 1 \\ 1 & 3 & 5 & 0 \end{bmatrix}$$

exiting the recursion when $n = 2$, i.e., computing the products of 2×2 matrices by the brute-force algorithm.

$$\left[\begin{array}{cc|cc} 1 & 0 & 2 & 1 \\ 4 & 1 & 1 & 0 \\ \hline 0 & 1 & 3 & 0 \\ 5 & 0 & 2 & 1 \end{array}\right] * \left[\begin{array}{cc|cc} 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 4 \\ \hline 2 & 0 & 1 & 1 \\ 1 & 3 & 5 & 0 \end{array}\right]$$

I get the gist, but I don't want to write down the clunky math on a computer. I've spent 3 hours on this homework so far. THREE. For one class. This is ridiculous professor, have you even tried completing the problems yourself before assigning them to understand their time commitment?By my estimates this singular problem will take 20 minutes.

$$\text{Result} = \begin{bmatrix} 5 & 4 & 7 & 3 \\ 4 & 5 & 1 & 9 \\ 8 & 1 & 3 & 7 \\ 5 & 8 & 7 & 7 \end{bmatrix}$$

5) Exercises 6.1

a. Problem 3 (part a & b) (2 points)

3. Consider the problem of finding the smallest and largest elements in an array of n numbers.

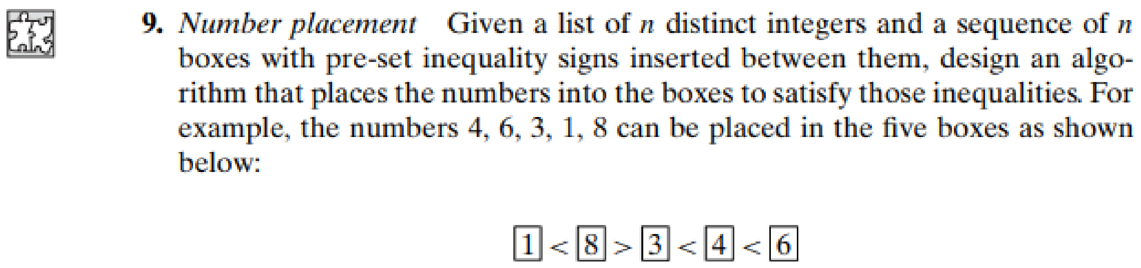
a. Design a presorting-based algorithm for solving this problem and determine its efficiency class.

b. Compare the efficiency of the three algorithms: (i) the brute-force algorithm, (ii) this presorting-based algorithm, and (iii) the divide-and-conquer algorithm (see Problem 2 in Exercises 5.1).

a.) The fastest sorting algorithm I know of is quick sort, so if we just apply quicksort to the array we can sort it in nlogn time. Now that the algorithm is sorted, only two checks need to be made, the end of the list and the beginning, this means that the total efficiency of this algorithm is nlogn +2

b. Brute force is n time to complete, while divide and conquer similarly takes n time to complete. Therefore they are faster in general. However, if you need to save time on calculations at some future moment, presorting is faster for actual use.

c. Problem 9 (2 points)



9. *Number placement* Given a list of n distinct integers and a sequence of n boxes with pre-set inequality signs inserted between them, design an algorithm that places the numbers into the boxes to satisfy those inequalities. For example, the numbers 4, 6, 3, 1, 8 can be placed in the five boxes as shown below:

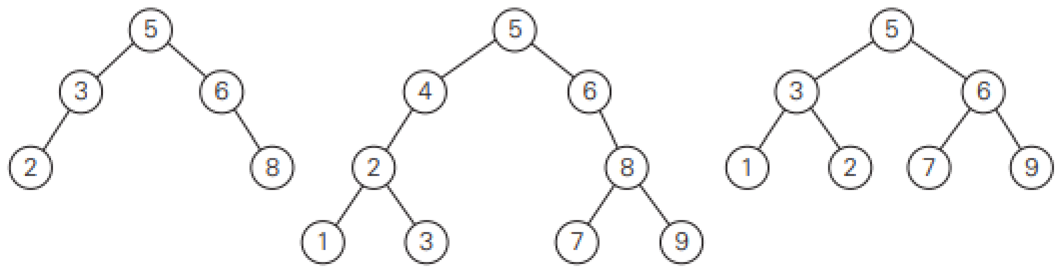
$$\boxed{1} < \boxed{8} > \boxed{3} < \boxed{4} < \boxed{6}$$

This problem looks tricky, however if you look closer it is just a sorting problem. The first step is to presort the list using quicksort. The next step is to find the lowest number in the lowest inequality position, because it is never higher than anything we can be sure this is the right place, delete this item from the list and the inequality position from consideration and continue this step until out of positions and elements.

6) Exercises 6.3

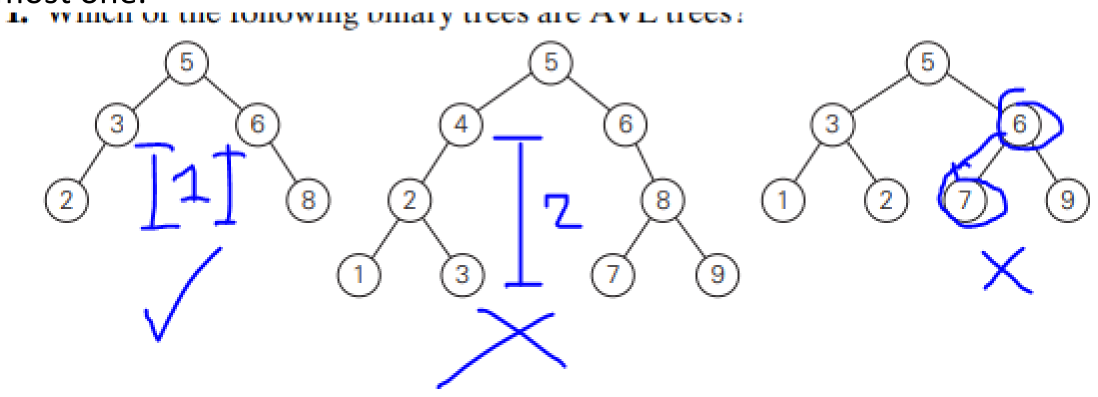
a. Problem 1 (2 points)

1. Which of the following binary trees are AVL trees?



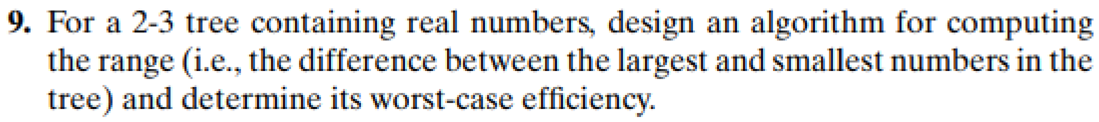
(a) (b) (c)

AVL trees mean the difference in height between the left and right subtrees is at most one.



Only A is a proper AVL tree. It should be noted that C meets the definition of an AVL tree provided, but is improperly organized and is not a binary tree. Kinda weird that the problem statement says it's a binary tree when it isn't.

b. Problem 9 (2 points)

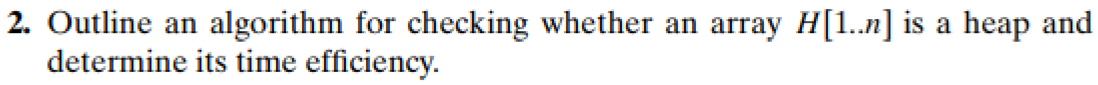


9. For a 2-3 tree containing real numbers, design an algorithm for computing the range (i.e., the difference between the largest and smallest numbers in the tree) and determine its worst-case efficiency.

In A 2-3 tree, the biggest element is the rightmost leaf, and the smallest is the leftmost leaf. This means all that we have to do is follow the edges of the tree to the left and right respectively and subtract the two. The height of a 2-3 tree is log(n), in ALL CASES. Which means that the time efficiency is always log(n), more specifically 2*log(n) +1, but we don't need to worry about that.

7) Exercises 6.4

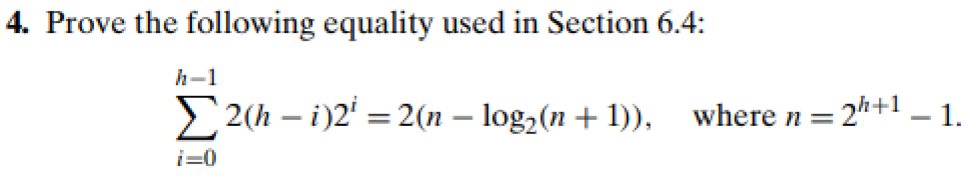
a. Problem 2 (2 points)



2. Outline an algorithm for checking whether an array $H[1..n]$ is a heap and determine its time efficiency.

A heap means that the parent is always larger than its children. This means we start at the parent, and check if its right and left child are greater than it. If they are, it is NOT a heap. This must repeat until n/2 is reached (the last point in the array where there are parents, and not just children). This algorithm must make two comparisons for every parent, so therefore its efficiency is n.

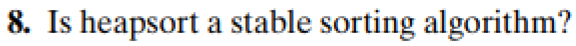
b. Problem 4 (2 points)



4. Prove the following equality used in Section 6.4:

$$\sum_{i=0}^{h-1} 2(h-i)2^i = 2(n - \log_2(n+1)), \quad \text{where } n = 2^{h+1} - 1.$$

c. Problem 8 (2 points)



8. Is heapsort a stable sorting algorithm?

An algorithm is stable if it maintains the relative positions of two otherwise equal elements. Heap sort does not do this, it is not stable.