Syeda Kazmi-Shah

Algorithms

Chapter 4: (Page 107) 4-1 a, b, c, d, e, f, g

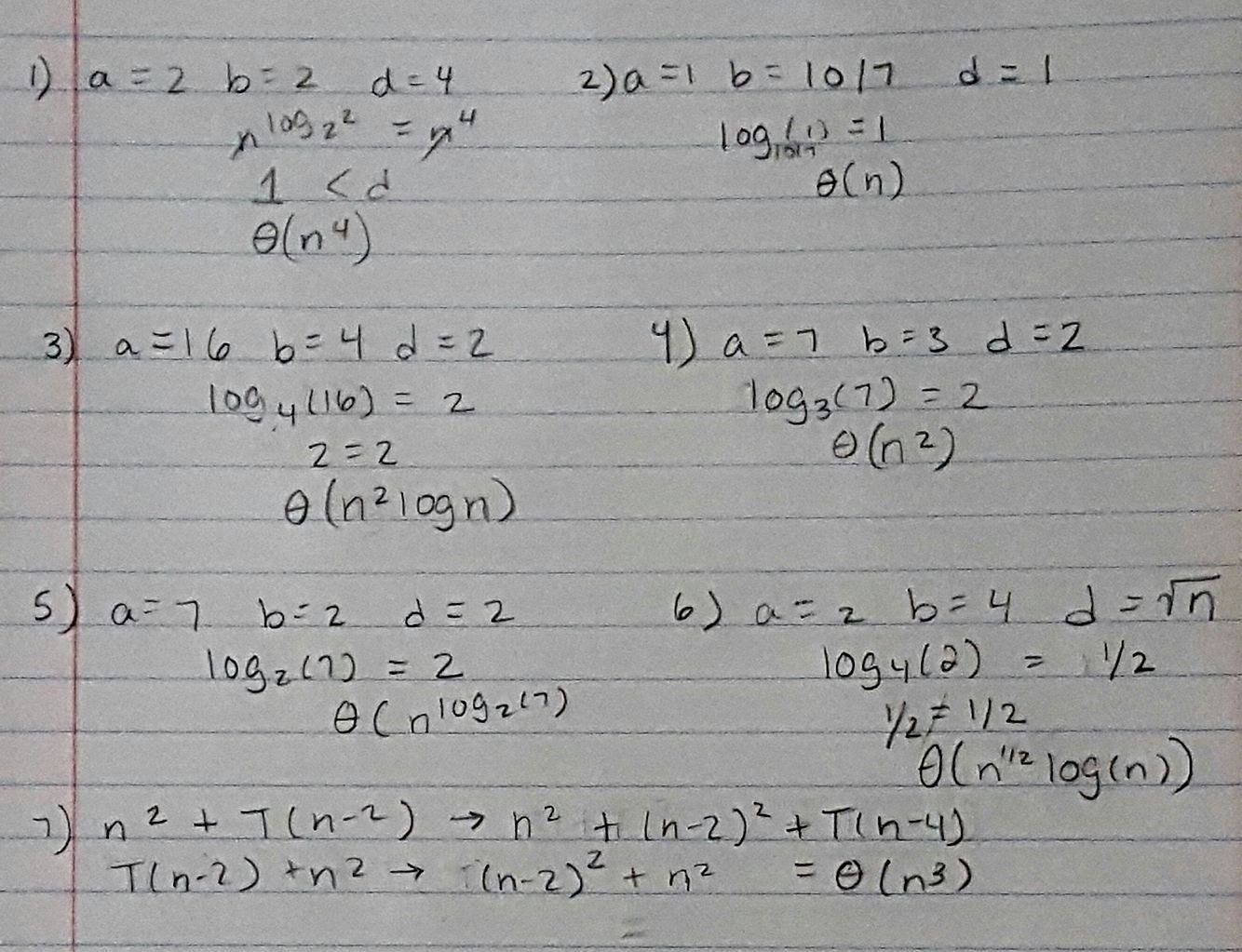
Chapter 7: (Page 178) 7.2-1, 7.2-3, 7.2-5

Chapter 8: (page 194) 8.1-4, (page 197) 8.2-4, (page 200) 8.3-3

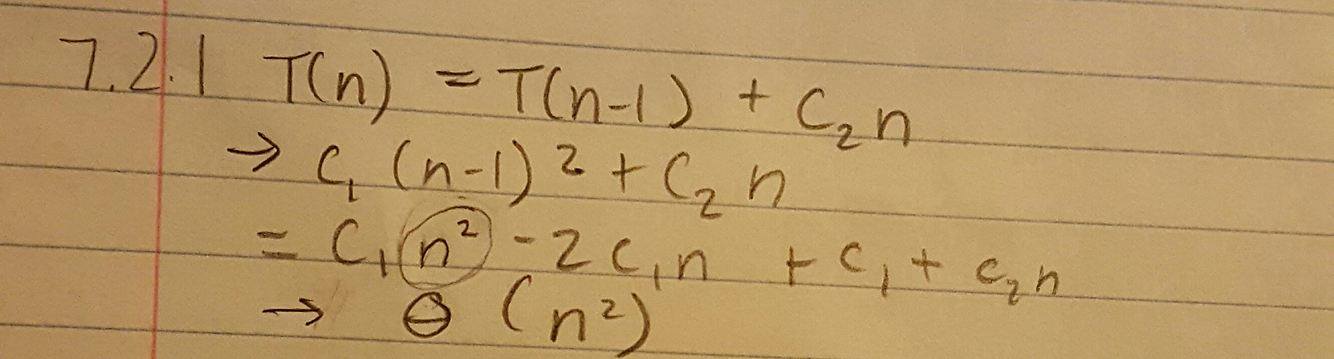
4-1. Recurrence example

Give asymptotic upper and lower bounds for Y(n) in each of the following recurrences. Assume that T(n) is constant for n<=2. Make your bounds as tight as possible, and justify your answers.

1. () (master method)
2. Θ(n) (master method,)
3. Θ) (master method)
4. () (master method)
5. (master method)
6. Θ )(master method)
7. T(n) = T(n-2) + = T(n-4)+ + = = T(n-2) + : T(n) = ()



7.2-1) Use the substitution method to probe that the recurrence T(n) = T(n-1) + (n) has the solution T(n) = , as claimed at the beginning of Section 7.2



7.2-3) Show that the running time of QUICKSORT is when the array A contains distinct elements and is sorted in decreasing order.

If for instance you have elements 1,3,5,4,2 and the pivots each time are chosen in order 1,2,3,4,5 Or 5,4,3,2,1. then the worst case running time is O(n\*n). If the pivot is the largest or smallest value in the array, then the partitioning step will result in one of the array (left or right) side will have n-1 elements and the other array will have zero elements.

Pseudocode for what Quicksort looks like:

procedure quickSort(left, right)

if right-left <= 0

return

else

pivot = A[right]

partition = partitionFunc(left, right, pivot)

quickSort(left,partition-1)

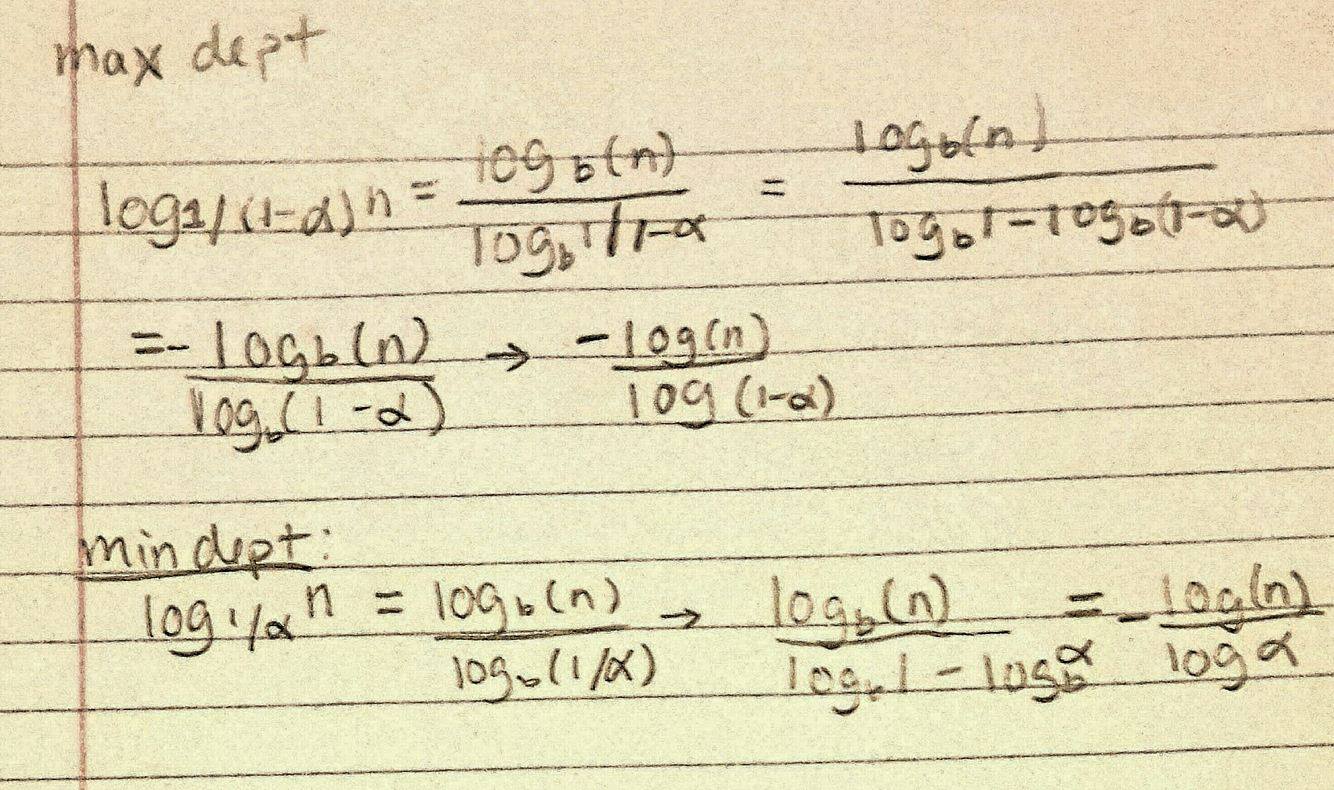
quickSort(partition+1,right)

end if

end procedure

7.2-5) Suppose that the splits at every level of quicksort are in the proportion 1- to α, where 0 < α ½ is a constant. Show that the minimum depth of a leaf in the recursion tree is approximately – lgn / lgα and the maximum depth is approximately -lgn / lg(1- α).

(don’t worry about integer round-off.)



8.1-4) Suppose that you are given a sequence of n elements to sort. The input sequence consists of n / k subsequences, each containing k elements. The elements in a given subsequence are all smaller than the elements in the succeeding subsequence and larger than the elements in the preceding subsequence. Thus, all that is needed to sort the whole sequence of length n is to sort the k elements in each of the n / k subsequences. Show an lower bound on the number of comparisons needed to solve this variant of the sorting problem. (Hint: It is not rigorous to simply combine the lower bounds for the individual subsequences.)

There are (k!)^n/k permutations of the whole sequence. So, the decision tree has  
(k!)^n/k leaves.

= n/k lg(k!) 🡪 from class lgn! Is nlogn  
  
= n/k · k lg k   
  
=

8.2-4) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range [a ..b] in O(2) time. Your algorithm should use preprocessing time.

For this, we just use the part of counting-sort that builds up the array A. Whenever we want to count the number of integers in [a..b], we take C[b] - C[a-1] (where C[-1] = 0). This gives us the number of integers in the given range. Counting Sort sorts the array such that using the difference of the final array will give you the amount of integers between the ranges a and b that you have queried.

8.3-3) Use induction to prove that radix sort works. Where does your proof need the assumption that the intermediate sort is stable?

Basis: If d = 1, sorting on that digit sorts the array correctly.  
Inductive step: Assume that radix-sort sorts d-1 digits correctly. Consider two elements a and b, with their dth digit a\_d and b\_d.  
🡪 a\_d > b\_d and b\_d > a\_d : radix-sort works correctly, because of most significant bit dominates regardless of the lower d-1 digits.  
🡪 a\_d = b\_d: radix-sort leaves a and b in the same order because it is stable sort. The order is righ since lower d-1 digits sorts correctly. This is the reason the intermediate sort must be stable.