

Exercises

I. Due Monday, 1/28

p. 13, 1.2-3 (2 pts) Give an exact answer.

p. 27, 2.2-3 (4 pts, 2 pts) Answer the first two questions: (a) How many elements need to be checked on the average (assuming the value that is being searched for is in the array, and is equally likely to be any one of the n elements in the array)? (b) How many elements need to be checked in the worst case? Give exact answers (in terms of n) in each case.

p. 50, 3.1-2. I have tried to help you answer this question by dividing it into several parts.

(a) (2 pts) You have to find a constant C and another constant N_1 so that $(n + a)^b \leq Cn^b$ for every $n \geq N_1$. Let's let $N_1 = |a|$ (i.e., a , if $a > 0$, and $-a$, if $a < 0$). Then if $n \geq N_1$, $n + a \leq 2n$. So what is a constant C so that the desired inequality holds for $n \geq N_1$?

(b) (4 pts) You have to find a constant D and another constant N_2 so that $(n + a)^b \geq Dn^b$ for every $n \geq N_2$. Again keeping in mind that a might be negative, what is a choice for D and N_2 that will work?

p. 50, 3.1-4 (4 pts) You only need to say yes or no in each case—no reasons are necessary.

p. 58, 3-3(a) (15 pts) **Important: Please read these instructions carefully.** Omit all the functions involving $*$ (there are 4 of them), and number the remaining 26 from 1 through 26, going across the rows. For example, function number 4 is $(\lg n)!$, function number 15 is 1, function number 26 is $2^{2^{n+1}}$. Your answer should be the numbers 1 through 26 arranged in some order, so that if we considered the 26 corresponding functions in that order, the growth rates would be increasing. (For example, if the first three numbers in your answer were 26, 4, and 15, I would interpret this to mean that $2^{2^{n+1}} = O((\lg n)!)^2$ and $(\lg n)! = O(1)$). It is important that your answer be in precisely this form, because I will grade this problem by entering your sequence of numbers as input to a computer program. **Please make sure that your answer contains all 26 numbers exactly once.** Note: there is more than one correct answer, since there are a few pairs of functions in this list that are O of each other, and in such a case, the two functions could appear in either order.

II. Due date to be specified later.

p. 75, 4.3-1 (2 points for each part)

p. 75, 4.3-2 (4 points) Another way to ask the question is: what is the largest integer value for a so that $T'(n) = o(T(n))$? (Give an exact value)

p. 85, Problem 4-1(e,f) (2 pts each)

p. 85, Problem 4-2 (6 pts) Describe precisely an algorithm to solve the problem in time $O(n)$. (Note: as of now, I don't know how to do this. I'm hoping I can figure it out.)

III. Due Friday, February 29

p. 129, 6.1-1 (2 pts)

p. 129, 6.1-4 (3 pts) Give the possible positions in the array (i.e., between 1 and n) where the smallest element might be.

p. 132, 6.2-6 (2 pts) For any n , describe how the numbers in the array might be arranged, and what the value of i could be, so that the call $\text{Max-Heapify}(A, i)$ would have runtime $\Omega(\log n)$

p. 136, 6.4-3 (4 pts)

p. 142, Problem 6-1 (3 pts each part) Use the following version of Max-Heap-Insert.

```
MaxHeapInsert(A, x)
  heapSize[A] = heapSize[A] + 1
  A[ heapSize[A] ] = x
  i = A[ heapSize[A] ]
  while i > 1 and A[ parent(i) ] < A[i]
  { swap( A[i], A[parent(i)] )
    i = parent(i)
  }
```

In part (b), you can answer the question by assuming that the values in the array are the numbers from 1 through n , and describing an initial order of the elements so that the BuildMaxHeap operation in the exercise requires time $\Omega(n \log n)$.

p. 148, 7.1-2 (2 pts) (only the first question in the exercise)

p. 153, 7.2-2 (3 pts)

p. 161, problem 7-3(b) (4 pts) Give a precise Θ -estimate.

IV. Due (tentatively) Wednesday, March 19

p. 167, 8.1-1 (2 pts)

p. 168, 8.1-3 (first and third questions) (4 pts each)

p. 173, 8.3-4 (5 pts) (There is an algorithm that can be described very concisely.)

p. 179, 8-3(b) (8 pts) The important thing is to describe your algorithm *simply*, and make it *clear* why its runtime is $O(n)$. How much credit you get will depend on how successfully you can do both things.

V. Due Friday, April 18

p. 192, 9.3-1 (first question only). You don't need to give a proof but say briefly what made you answer the way you did. (4 pts)

p. 192, 9.3-5 (4 pts) The key word here is "simple." Give pseudocode, which will not require many lines.

p. 259, 12.2-1 (4 pts)

p. 259, 12.2-4 (4 pts)

p. 530, 22.1-1 (4 pts) Note: the problem doesn't say, for every vertex, how long does it take to calculate the out-degree of that vertex? It says, how long does it take to do a calculation, the result of which are the out-degrees of all the vertices.

p. 531, 22.1-7 (4 pts)