Q 1.

Two friends are discussing the growth of an algorithm. One of them makes a statement "The running time is at least O(n2)". The other person correctly points out that this is an invalid statement. Why is this an invalid statement? Explain in one sentence, (not more than two).

ANS 1.

Since by definition O(.) is an upper bound on performance, you can’t say at least O(.), you can say at most O(.). You can say at least about Ω(.).

Q 2.

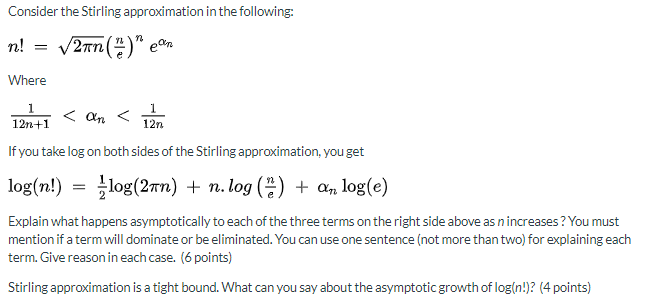
Do you have a preference for O(lg n) over O(lg n2)? The answer should consist of a justification for a Yes or a No. Use one sentence to justify a clearly yes or no answer. There is no point for correct answer without justification.

ANS 2.

No, you can’t have preference among logs of powers of *n* because power multiplies the log and is just like

*c log n.*

Q 3.

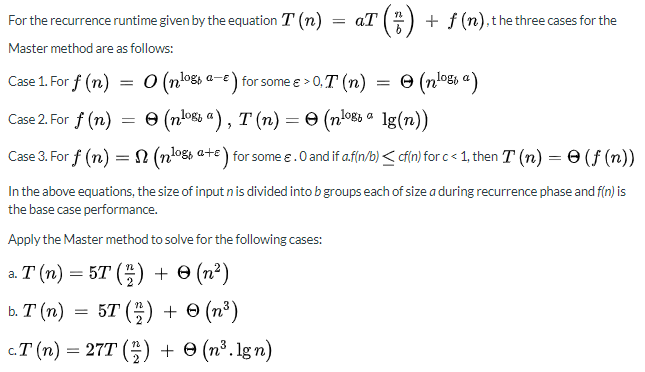


ANS 3.

First term grows as log(n) and will diminish as compared with the second term that grows as n log n. The second term dominates asymptotically. The third term diminishes quickly because of n being in the denominator.

We can say that n! = Θ(n lg n)

Q 4.



ANS 4.

1. For this part *a = 5, b = 2, nloga/b = n2+ε,* so it is case 1 and *T(n) = Θ(n2.?)*
2. For this part *a = 5, b = 2, nloga/b = n2+ε,* so it could be case 3.

Let’s check the second condition for case 3.

5n3/8 ≤ c n3 for 5/8 < c < 1, so case 3 applies

And therefore *T(n) = Θ(n3)*

1. For this part *a = 27, b = 2, nloga/b = n4.75,* so it could be case 3.

Let’s check the second condition for case 3.

Let’s check if the following relation can be satisfied for c ≤ 1

27/8n3 lg n/2 ≤ c n3 lg n

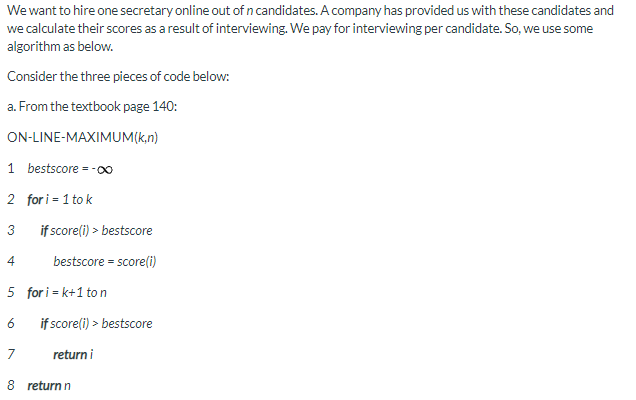
= 27/8 (lg n/2)/lg n ≤ c Since lg n is positive

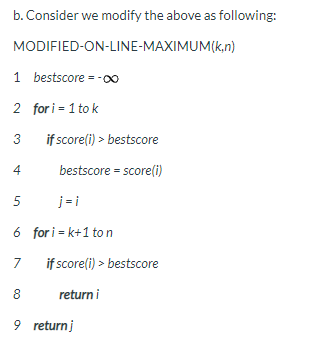
= 27/8 ( 1 + 1/lg n) ≤ c

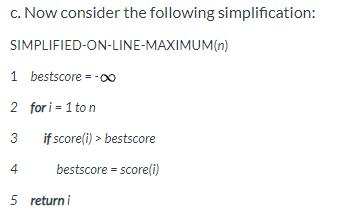
The above satisfies the condition that c < 1 for sufficiently large n, hence case 3 applies.

And therefore *T(n) = Θ(n3 lg n)*

Q 5.







Answer the following questions for the three types of codes above:

A. What will be returned by calling ON-LINE-MAXIMUM(3,5) for candidates whose scores are 67, 38, 75, 72, 45. (5 points)

B. What will be returned by calling MODIFIED-ON-LINE-MAXIMUM(3,5) for candidates whose scores are 67, 38, 75, 72, 45. (5 points)

C. What will be returned by calling SIMPLIFIED-ON-LINE-MAXIMUM(5) for candidates whose scores are 67, 38, 75, 72, 45. (5 points)

D. If we randomize the input, just like we did in assignment 3, we will pick up any one as the first candidate with probability 1/5, the second one with probability 1/4, the third one with 1/3 ans so on. What is the expected number of candidates to interview before you hire one who meets the criterion that the candidate should have at least 70 points?  (10 points)

E. What is the main problem with (a) and (c)? It is not the same problem. (5 points)

ANS 5.

1. 5 (the index of 45)
2. 3 (the index of 75)
3. 3 (the index of 75)
4. The criteria will be met if any of the candidate with 72 or 75 is picked up for interview

Let n be the number of candidates to be interviewed for picking up the successful candidate. Then,

P(1) = First candidate success = P (candidates with 72 or 75 picked up) = 2/5

P(2) = Success on second interview = P(failure on first)P(success on second | failure on first)

P(2) = 3/5 x ½ = 3/10

P(3) = P(failure on first and second, success on 3rd) = 3/5 x ½ x 2/3 = 1/5

P(4) = P(failure on first three, success on fourth) = 3/5 x ½ x 1/3 x 1 = 1/10

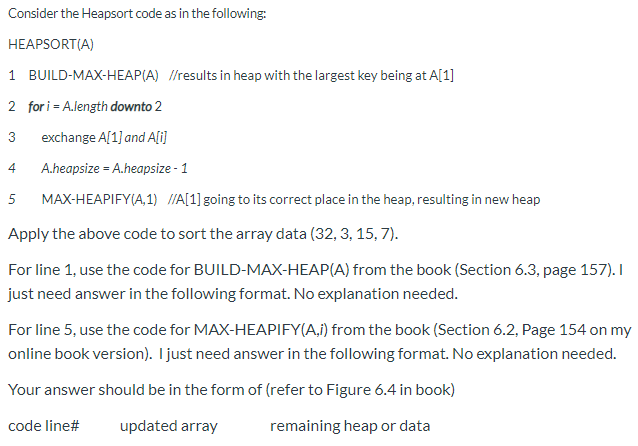
Average number of interviews = 1.P(1) + 2. P(2) + 3.P(3) + 4.P(4)

= 2/5 + 3/5 + 3/5 + 2/5 = 2

So, on the average, all you need is 2 interviews. This shows the power of randomized algorithms.

1. The main problem with (a) algorithm is that if no one is better than the criterion set by the first *k* candidates, we may select the worst candidate, which is the case for this example. The problem with the (c) algorithm is that we always pay for interviewing all the candidates. This is not practical in real life where there could be many many candidates to interview for selecting one.

Q 6.



ANS 6.

1 { } {32, 7, 15, 3}

2 { } {32, 7, 15, 3}

3 { } {3, 7, 15, 32}

4 {32} { 3, 7, 15}

5 {32} {15, 3, 7}

2 i = 3

3 {32} {7, 3, 15}

4 {32, 15} {7, 3}

5 {32, 15} {7, 3}

2 i = 2

3 {32, 15} {3, 7}

4 {32, 15, 7} {3}

Q 7.

For the quicksort partitioning algorithm the book says that the partitioning should be done around the last element of the array (called the pivot). This is not a good approach because if this element is the largest or the smallest, the partition will be one sided.

If the first element of the array is chosen as the pivot (instead of the last), will this make any difference to algorithm performance? (5 points for the correct answer and 5 for correct explanation).

ANS 7.

No, it will not make any difference to the algorithm performance.

Explanation: The probability of the first element being worst case is the same as the last element being worst case.

Q 8.

If an algorithm or a function grows as O(n), it also grows as Θ ( n ).

ANS 8.

False.

Q 9.

For worst case quicksort, partitioning takes Θ ( n ), which gives T ( n ) = T ( n − 1 ) + Θ ( n ) with all elements being on one side of partition. By expanding it down in a recurrence tree, we see that Θ ( n )has added about n times by the time we reach T(0). This gives a T(n) of Θ ( n 2 ).

ANS 9.

True

Q 10.

For best case quicksort, partitioning takes Θ ( n ), which gives T ( n ) = 2 T ( n /2 ) + Θ ( n ) with all elements being equally divided on both sides of the pivot. From common sense, we can tell that Θ ( n )has added about lg n times by the time we reach T(0). This gives a T(n) of Θ ( n l g n ).

ANS 10.

True.

Q 11.

Merge sort is known to be better than insertion sort for all values on n, the input size.

ANS 11.

False.

Q 12.

In the asymptotic sense, the recursive implementation of matrix multiplication performs better than iterative implementation.

ANS 12.

False.

Q13.

For this question, I have to tell a true but short story. if you do not follow the story, it will not affect your ability to answer correctly.

During my student years, one day my friends and I picked up a deck of cards lying on my apartment floor, got it cut and started distributing cards to play bridge. I got all black cards, a mix of clubs and spades. When bidding started, it became obvious that my partner also had all black cards and our opponents had all red cards. Very quickly, we bid 7 spades and made it without a problem. That is the highest bid in this situation. Later, I told a friend about this. She said 'actually I did it. I was bored and started putting the cards with alternating red and black and when I was done I put them on the floor and left'. The mistake we made while playing was that we did not shuffle them.

This was an example of worst case (for our opponents) or best case (for my partner and me) sorting. The lesson we learned was that "always shuffle".

In this problem, worth 5 points, we want to prove mathematically that given an array about which we have no knowledge, "always shuffling" is better and can hardly result in a worst case situation. Prove using probability that always shuffle of large arrays has asymptotically zero probability of getting a worst case scenario. Of course, by shuffle we mean permute.

You get additional bonus 5 points if you also explain why always shuffle is better despite shuffling costing you an additional Θ ( n ).

ANS 13.

If the array of size n is shuffled, the probability of a worst-case is 1/n! which is zero asymptotically.