

BLG335e
Pre-Midterm Special

Asymptotic Notation & Growth of Functions

2.2-1

Express the function $n^3/1000 - 100n^2 - 100n + 3$ in terms of Θ -notation.

1.2-2

Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n , insertion sort runs in $8n^2$ steps, while merge sort runs in $64n \lg n$ steps. For which values of n does insertion sort beat merge sort?

3.1-4

Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

3.2-1

Show that if $f(n)$ and $g(n)$ are monotonically increasing functions, then so are the functions $f(n) + g(n)$ and $f(g(n))$, and if $f(n)$ and $g(n)$ are in addition nonnegative, then $f(n) \cdot g(n)$ is monotonically increasing.

Exercise 1.2-2

We wish to determine for which values of n the inequality $8n^2 < 64n \log_2(n)$ holds. This happens when $n < 8 \log_2(n)$, or when $n \leq 43$. In other words, insertion sort runs faster when we're sorting at most 43 items. Otherwise merge sort is faster.

Exercise 3.1-4

$2^{n+1} \geq 2 \cdot 2^n$ for all $n \geq 0$, so $2^{n+1} = O(2^n)$. However, 2^{2n} is not $O(2^n)$. If it were, there would exist n_0 and c such that $n \geq n_0$ implies $2^n \cdot 2^n = 2^{2n} \leq c2^n$, so $2^n \leq c$ for $n \geq n_0$ which is clearly impossible since c is a constant.

Exercise 3.2-1

Let $n_1 < n_2$ be arbitrary. From f and g being monotonically increasing, we know $f(n_1) < f(n_2)$ and $g(n_1) < g(n_2)$. So

$$f(n_1) + g(n_1) < f(n_2) + g(n_1) < f(n_2) + g(n_2)$$

Since $g(n_1) < g(n_2)$, we have $f(g(n_1)) < f(g(n_2))$. Lastly, if both are nonnegative, then,

$$\begin{aligned} f(n_1)g(n_1) &= f(n_2)g(n_1) + (f(n_2) - f(n_1))g(n_1) \\ &= f(n_2)g(n_2) + f(n_2)(g(n_2) - g(n_1)) + (f(n_2) - f(n_1))g(n_1) \end{aligned}$$

Since $f(n_1) \geq 0$, $f(n_2) > 0$, so, the second term in this expression is greater than zero. The third term is nonnegative, so, the whole thing is $< f(n_2)g(n_2)$.

Divide&Conquer

4.4-3

Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 4T(n/2 + 2) + n$. Use the substitution method to verify your answer.

4.4-7

Draw the recursion tree for $T(n) = 4T(\lfloor n/2 \rfloor) + cn$, where c is a constant, and provide a tight asymptotic bound on its solution. Verify your bound by the substitution method.

4.4-8

Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(n - a) + T(a) + cn$, where $a \geq 1$ and $c > 0$ are constants.

Exercise 4.4-3

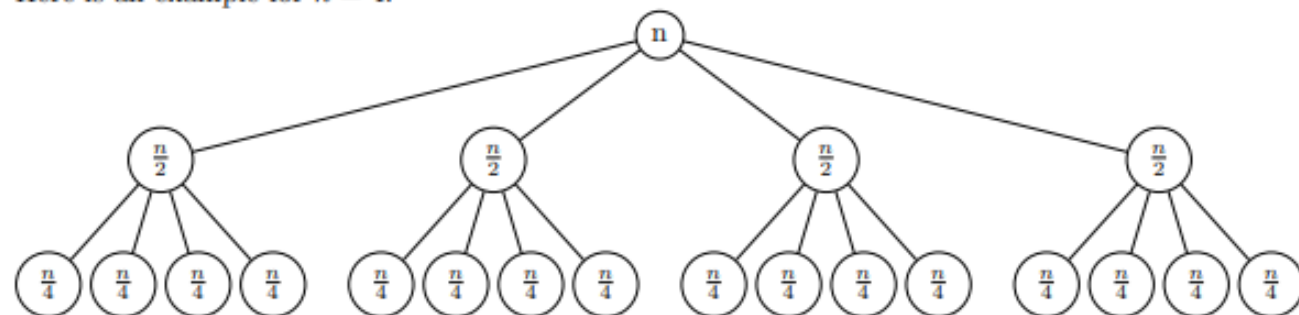
Again, we notice that the depth of the tree is around $\lg(n)$, and there are 4^i vertices on the i th level below the root, so, we have that our guess is n^2 . We show this fact by the substitution method. We show that $T(n) \leq cn^2 - 6n$

$$\begin{aligned} T(n) &= 4T(n/2 + 2) + n \\ &\leq 4c(n^2/4 + 2n + 4 - 3n - 12) + n \\ &= cn^2 - 4cn - 32c + n \end{aligned}$$

Which will be $\leq cn^2 - 6n$ so long as we have $-4c + 1 \leq -6$ and $c \geq 0$. These can both be satisfied so long as $c \geq \frac{7}{4}$.

Exercise 4.4-7

Here is an example for $n = 4$.



We can see by an easy substitution that the answer is $\Theta(n^2)$. Suppose that $T(n) \leq c'n^2$ then

$$\begin{aligned} T(n) &= 4T(\lfloor n/2 \rfloor) + cn \\ &\leq c'n^2 + cn \end{aligned}$$

which is $\leq c'n^2$ whenever we have that $c' + \frac{c}{n} \leq 1$, which, for large enough n is true so long as $c' < 1$. We can do a similar thing to show that it is also bounded below by n^2 .

Exercise 4.4-8

$$\begin{aligned} &T(a) + cn \\ &\quad \mid \\ &T(a) + c(n - a) \\ &\quad \mid \\ &T(a) + c(n - 2a) \\ &\quad \vdots \\ &T(1) \end{aligned}$$

Since each node of the recursion tree has only one child, the cost at each level is just the cost of the node. Moreover, there are $\lceil n/a \rceil$ levels in the tree. Summing the cost at each level we see that the total cost of the algorithm is

$$\sum_{i=0}^{\lceil n/a \rceil - 1} T(a) + c(n - ia) = \lceil n/a \rceil T(a) + c\lceil n/a \rceil n - ca \frac{\lceil n/a \rceil (\lceil n/a \rceil - 1)}{2}.$$

To compute the asymptotics we can assume n is divisible by a and ignore the ceiling functions. Then this becomes


$$\frac{c}{2a} n^2 + (T(a)/a + c/2)n = \Theta(n^2).$$

4-1 Recurrence examples

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


a. $T(n) = 2T(n/2) + n^4$.


b. $T(n) = T(7n/10) + n$.

c. $T(n) = 16T(n/4) + n^2$. 

d. $T(n) = 7T(n/3) + n^2$.

e. $T(n) = 7T(n/2) + n^2$.


f. $T(n) = 2T(n/4) + \sqrt{n}$. 


g. $T(n) = T(n-2) + n^2$. 


4-3 More recurrence examples

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

a. $T(n) = 4T(n/3) + n \lg n$.

b. $T(n) = 3T(n/3) + n/\lg n$. 

c. $T(n) = 4T(n/2) + n^2 \sqrt{n}$. 

d. $T(n) = 3T(n/3 - 2) + n/2$. 

e. $T(n) = 2T(n/2) + n/\lg n$.

f. $T(n) = T(n/2) + T(n/4) + T(n/8) + n$.

g. $T(n) = T(n-1) + 1/n$.

h. $T(n) = T(n-1) + \lg n$.

i. $T(n) = T(n-2) + 1/\lg n$.

j. $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

b. We first show by substitution that $T(n) \leq n \lg(n)$.

$$T(n) = 3T(n/3) + n/\lg(n) \leq cn \lg(n) - cn \lg(3) + n/\lg(n) = cn \lg(n) + n\left(\frac{1}{\lg(n)} - c \lg(3)\right) \leq cn \lg(n)$$

now, we show that $T(n) \geq cn^{1-\epsilon}$ for every $\epsilon > 0$.

$$T(n) = 3T(n/3) + n/\lg(n) \geq 3c/3^{1-\epsilon}n^{1-\epsilon} + n/\lg(n) = 3^\epsilon cn^{1-\epsilon} + n/\lg(n)$$

showing that this is $\leq cn^{1-\epsilon}$ is the same as showing

$$3^\epsilon + n^\epsilon/(c \lg(n)) \geq 1$$

Since $\lg(n) \in o(n^\epsilon)$ this inequality holds. So, we have that The function is soft Theta of n , see problem 3-5.

c. By Master Theorem, $T(n) \in \Theta(n^2 \lg(n))$

f. By Master Theorem, $T(n) \in \Theta(n^{1/2} \lg(n))$

g. Let $d = m \bmod 2$, we can easily see that the exact value of $T(n)$ is

$$\sum_{j=1}^{j=n/2} (2j+d)^2 = \sum_{j=1}^{n/2} 4j^2 + 4jd + d^2 = \frac{n(n+2)(n+1)}{6} + \frac{n(n+2)d}{2} + \frac{d^2n}{2}$$

This has a leading term of $n^3/6$, and so $T(n) \in \Theta(n^3)$

c. By Master Theorem, $T(n) \in \Theta(n^{2.5})$

d. it is $\Theta(n \lg(n))$. The subtraction occurring inside the argument to T won't change the asymptotics of the solution, that is, for large n the division is so much more of a change than the subtraction that it is the only part that matters. once we drop that subtraction, the solution comes by the master theorem.

PROBABILISTIC ANALYSIS

5.4-4 ★

How many people should be invited to a party in order to make it likely that there are *three* people with the same birthday?

5.4-6 ★

Suppose that n balls are tossed into n bins, where each toss is independent and the ball is equally likely to end up in any bin. What is the expected number of empty bins? What is the expected number of bins with exactly one ball?

Exercise 5.4-4

We can compute "likely" in two ways. The probability that at least three people share the same birthday is 1 minus the probability that none share the same birthday, minus the probability that any number of pairs of people share the same birthday. Let $n = 365$ denote the number of days in a year and k be the number of people at the party. As computed earlier in the section, we have

$$P(\text{all unique birthdays}) = 1 \cdot \left(\frac{n-1}{n}\right) \left(\frac{n-2}{n}\right) \cdots \left(\frac{n-k+1}{n}\right) \leq e^{-k(k-1)/2n}.$$

Next we compute the probability that exactly i pairs of people share a birthday. There are $\binom{k}{2} \binom{k-2}{2} \cdots \binom{k-2i+2}{2}$ ways to choose an ordered collection of i pairs from the k people, $\binom{n}{i}$ ways to select the set of birthdays the pairs will share, and the probability that any such ordered subset has these precise birthdays is $(1/n^2)^i$. Multiplying by the probability that the rest of the birthdays are different and unique we have

$$\begin{aligned} P(\text{exactly } i \text{ pairs of people same}) &= \frac{\binom{k}{2} \binom{k-2}{2} \cdots \binom{k-2i+2}{2} \binom{n}{i}}{n^{2i}} \left(\frac{n-i}{n}\right) \left(\frac{n-i-1}{n}\right) \cdots \left(\frac{n-k+i+1}{n}\right) \\ &\leq \frac{k! \binom{n}{i}}{2^i (k-2i)! n^{2i}} \left(1 - \frac{i}{n}\right) \left(1 - \frac{i+1}{n}\right) \cdots \left(1 - \frac{k-i-1}{n}\right) \\ &\leq \frac{k! \binom{n}{i}}{2^i (k-2i)! n^{2i}} e^{-(k-1)(k-2i-1)/2n} \end{aligned}$$

Thus,

$$P(\geq 3 \text{ people share a birthday}) \leq 1 - e^{-k(k-1)/2n} - \frac{k! \binom{n}{i}}{2^i (k-2i)! n^{2i}} e^{-(k-1)(k-2i-1)/2n}.$$

This is pretty messy, even with the simplifying inequality $1 + x \leq e^x$, so we'll do another analysis, this time on expectation. We'll determine the value of k required such that the expected number of triples (i, j, m) where person i , person j , and person m share a birthday is at least 1. Let X_{ijm} be the indicator variable that this triple of people share a birthday and X denote the total number of triples of birthday-sharers. Then we have

$$E[X] = \sum_{\text{distinct triples } (i,j,m)} E[X_{ijm}] = \binom{k}{3} \frac{1}{n^2}.$$

To make $E[X]$ exceed 1 we need to find the smallest k such that $k(k-1)(k-2) \geq 6(365)^2$, which happens when $k = 94$.

Problem 5.4-6

Let X_i be the indicator variable that bin i is empty after all balls are tossed and X be the random variable that gives the number of empty bins. Then we have

$$E[X] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n \left(\frac{n-1}{n}\right)^n = n \left(\frac{n-1}{n}\right)^n.$$

Now let X_i be the indicator variable that bin i contains exactly 1 ball after all balls are tossed and X be the random variable that gives the number of bins containing exactly 1 ball. Then we have

$$E[X] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n \binom{n}{1} \left(\frac{n-1}{n}\right)^{n-1} \frac{1}{n} = n \left(\frac{n-1}{n}\right)^{n-1}$$

because we need to choose which toss will go into bin i , then multiply by the probability that that toss goes into that bin and the remaining $n-1$ tosses avoid it.

Sorting

- Array : [2, 8, 4, 1, 6, 2, 1, 8]
- Sort
 - Insertion
 - Merge
 - Quick
 - Counting

7.2-2

What is the running time of QUICKSORT when all elements of array A have the same value?

7.2-3

Show that the running time of QUICKSORT is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

7.2-5

Suppose that the splits at every level of quicksort are in the proportion $1 - \alpha$ to α , where $0 < \alpha \leq 1/2$ is a constant. Show that the minimum depth of a leaf in the recursion tree is approximately $-\lg n / \lg \alpha$ and the maximum depth is approximately $-\lg n / \lg(1 - \alpha)$. (Don't worry about integer round-off.)

Exercise 7.2-2

The running time of QUICKSORT on an array in which every element has the same value is n^2 . This is because the partition will always occur at the last position of the array (Exercise 7.1-2) so the algorithm exhibits worst-case behavior.

Exercise 7.2-3

If the array is already sorted in decreasing order, then, the pivot element is less than all the other elements. The partition step takes $\Theta(n)$ time, and then leaves you with a subproblem of size $n - 1$ and a subproblem of size 0. This gives us the recurrence considered in 7.2-1. Which we showed has a solution that is $\Theta(n^2)$.

Exercise 7.2-5

The minimum depth corresponds to repeatedly taking the smaller subproblem, that is, the branch whose size is proportional to α . Then, this will fall to 1 in k steps where $1 \approx (\alpha)^k n$. So, $k \approx \log_\alpha(1/n) = -\frac{\lg(n)}{\lg(\alpha)}$. The longest depth corresponds to always taking the larger subproblem. we then have an identical expression, replacing α with $1 - \alpha$.

8.3-1

Using Figure 8.3 as a model, illustrate the operation of RADIX-SORT on the following list of English words: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

8.3-2

Which of the following sorting algorithms are stable: insertion sort, merge sort, heapsort, and quicksort? Give a simple scheme that makes any sorting algorithm stable. How much additional time and space does your scheme entail?

8.4-1

Using Figure 8.4 as a model, illustrate the operation of BUCKET-SORT on the array $A = \langle .79, .13, .16, .64, .39, .20, .89, .53, .71, .42 \rangle$.

8.4-2

Explain why the worst-case running time for bucket sort is $\Theta(n^2)$. What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time $O(n \lg n)$?

Exercise 8.3-1

Starting with the unsorted words on the left, and stable sorting by progressively more important positions.

<i>COW</i>	<i>SEA</i>	<i>TAB</i>	<i>BAR</i>
<i>DOG</i>	<i>TEA</i>	<i>BAR</i>	<i>BIG</i>
<i>SEA</i>	<i>MOB</i>	<i>EAR</i>	<i>BOX</i>
<i>RUG</i>	<i>TAB</i>	<i>TAR</i>	<i>COW</i>
<i>ROW</i>	<i>RUG</i>	<i>SEA</i>	<i>DIG</i>
<i>MOB</i>	<i>DOG</i>	<i>TEA</i>	<i>DOG</i>
<i>BOX</i>	<i>DIG</i>	<i>DIG</i>	<i>EAR</i>
<i>TAB</i>	<i>BIG</i>	<i>BIG</i>	<i>FOX</i>
<i>BAR</i>	<i>BAR</i>	<i>MOB</i>	<i>MOB</i>
<i>EAR</i>	<i>EAR</i>	<i>DOG</i>	<i>NOW</i>
<i>TAR</i>	<i>TAR</i>	<i>COW</i>	<i>ROW</i>
<i>DIG</i>	<i>COW</i>	<i>ROW</i>	<i>RUG</i>
<i>BIG</i>	<i>ROW</i>	<i>NOW</i>	<i>SEA</i>
<i>TEA</i>	<i>NOW</i>	<i>BOX</i>	<i>TAB</i>
<i>NOW</i>	<i>BOX</i>	<i>FOX</i>	<i>TAR</i>
<i>FOX</i>	<i>FOX</i>	<i>RUG</i>	<i>TEA</i>

Exercise 8.3-2

Insertion sort and merge sort are stable. Heapsort and quicksort are not. To make any sorting algorithm stable we can preprocess, replacing each element of an array with an ordered pair. The first entry will be the value of the element, and the second value will be the index of the element. For example, the array $[2, 1, 1, 3, 4, 4, 4]$ would become $[(2, 1), (1, 2), (1, 3), (3, 4), (4, 5), (4, 6), (4, 7)]$. We now interpret $(i, j) < (k, m)$ if $i < k$ or $i = k$ and $j < m$. Under this definition of less-than, the algorithm is guaranteed to be stable because each of our new elements is distinct and the index comparison ensures that if a repeat element appeared later in the original array, it must appear later in the sorted array. This doubles the space requirement, but the running time will be asymptotically unchanged.

Exercise 8.4-1

The sublists formed are $\langle .13, .16 \rangle$, $\langle .20 \rangle$, $\langle .39 \rangle$, $\langle .42 \rangle$, $\langle .53 \rangle$, $\langle .64 \rangle$, $\langle .71, .79 \rangle$, $\langle .89 \rangle$. Putting them together, we get $\langle .13, .16, .20, .39, .42, .53, .64, .71, .79, .89 \rangle$.

Exercise 8.4-2

In the worst case, we could have a bucket which contains all n values of the array. Since insertion sort has worst case running time $O(n^2)$, so does Bucket sort. We can avoid this by using merge sort to sort each bucket instead, which has worst case running time $O(n \lg n)$.