2.1-1
Using Figure 2.2 as a model, illustrate the operation of INSERTION-SORT on an array initially containing the sequence $\langle 31, 41, 59, 26, 41, 58 \rangle$.

31 41 59 26 41 58

31 41 59 26 41 58

31 41 59 26 41 58

26 31 41 59 41 58

26 31 41 41 59 58

26 31 41 41 58 59 #

2.1-2

Consider the procedure SUM-ARRAY on the facing page. It computes the sum of the n numbers in array $A[1:n]$. State a loop invariant for this procedure, and use its initialization, maintenance, and termination properties to show that the SUM-ARRAY procedure returns the sum of the numbers in $A[1:n]$.

```
SUM-ARRAY(A, n)
sum = 0
for i = 1 to n
    sum = sum + A[i]
return sum
```

Loop invariant: After the i-th iteration, "sum" will be the sum of A[1:i]

Initialization: Before the loop start, there is no element of "A" is added into "sum", so "sum" is "0". The invariant holds.

Maintenance: If the invariant holds before the i-th iteration begins ("sum" is the sum of "A[1:i-1]". Then in the i-th iteration, "A[i]" is added to "sum", expanding "sum" to "A[1:i]", The invariant holds.

Termination: The loop stop after the n-th iteration, so "sum" is the sum of "A[1:n]" finally, which satisfied the requirement.

2.1-4

Consider the ***searching problem***:

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$ stored in array $A[1:n]$ and a value x.

Output: An index i such that x equals $A[i]$ or the special value NIL if x does not appear in A.

Write pseudocode for ***linear search***, which scans through the array from beginning to end, looking for x. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

```
linear-search (A, x) {
    for i=1 to n {
        if A[i] == x {
            return i
        }
    }
    return nil
}
```

Loop invariant: after the i-th iteration, "x" is not in "A[1:i]"

Initialization: Before the first iteration begin, "A[1:0]" is a empty list, so "x" must not in it. The invariant holds.

Maintenance: At the i-th iteration, if "x" not equal to "A[i]", the invariant holds, Else the loop will terminate and return i

Termination: After the n-iteration, we know that "x" is not in "A[1:n]", that is "x" is not in "A". So the function return "nil"

2.2-3

Consider linear search again (see Exercise 2.1-4). How many elements of the input array need to be checked on the average, assuming that the element being searched for is equally likely to be any element in the array? How about in the worst case? Using Θ-notation, give the average-case and worst-case running times of linear search. Justify your answers.

Assume every element in "A" has the probability p to be "x"

The probability of finding "$x == A[k]$" at the k-th iteration is $p(1-p)^{k-1}$

$$E(step) = \underbrace{A.length \cdot (1-p)^{A.length}}_{\text{not found}} + \underbrace{\sum_{k=1}^{A.length} k \cdot p(1-p)^{k-1}}_{\text{found at k-th}} \rightarrow \text{worst case: } \Theta(A.length)$$

#

$$= A.length \cdot (1-p)^{A.length} + \sum_{s=1}^{A.length} \sum_{k=s}^{A.length} p(1-p)^{k-1}$$

$k=1, 2, 3$. reverse

$\Rightarrow A.length,\ A.length-1$ - - - -

$$= A.length\,(1-p)^{A.length} + \sum_{s=1}^{A.length} p \sum_{k=s}^{A.length} (1-p)^{k-1}$$

$$\sum_{k=s}^{A.length} (1-p)^{k-1} = (1-p)^{s-1}\left(1-(1-p)^{A.length-s+1}\right) / \left(1-(1-p)\right)$$

$a_1 + ra_1 + r^2a_3 + \dots + r^n a_1$

$= a_1(1-r^n)/(1-r)$

$$= \frac{(1-p)^{s-1} - (1-p)^{A.length}}{p}$$

$$= \cancel{A.length\,(1-p)^{A.length}} + \sum_{s=1}^{A.length} (1-p)^{s-1} - \underbrace{(1-p)^{A.length}}_{\cancel{A.length\,(1-p)^{A.length}}}$$

$$= \sum_{s=1}^{A.length} (1-p)^{s-1} = (1-p)^{1-1}\left(1-(1-p)^{A.length}\right) / \left(1-(1-p)\right)$$

$$= \frac{1-(1-p)^{A.length}}{p}$$

$$\Rightarrow E(steps) = \frac{1}{p} - \frac{1}{p}(1-p)^{A.length}$$

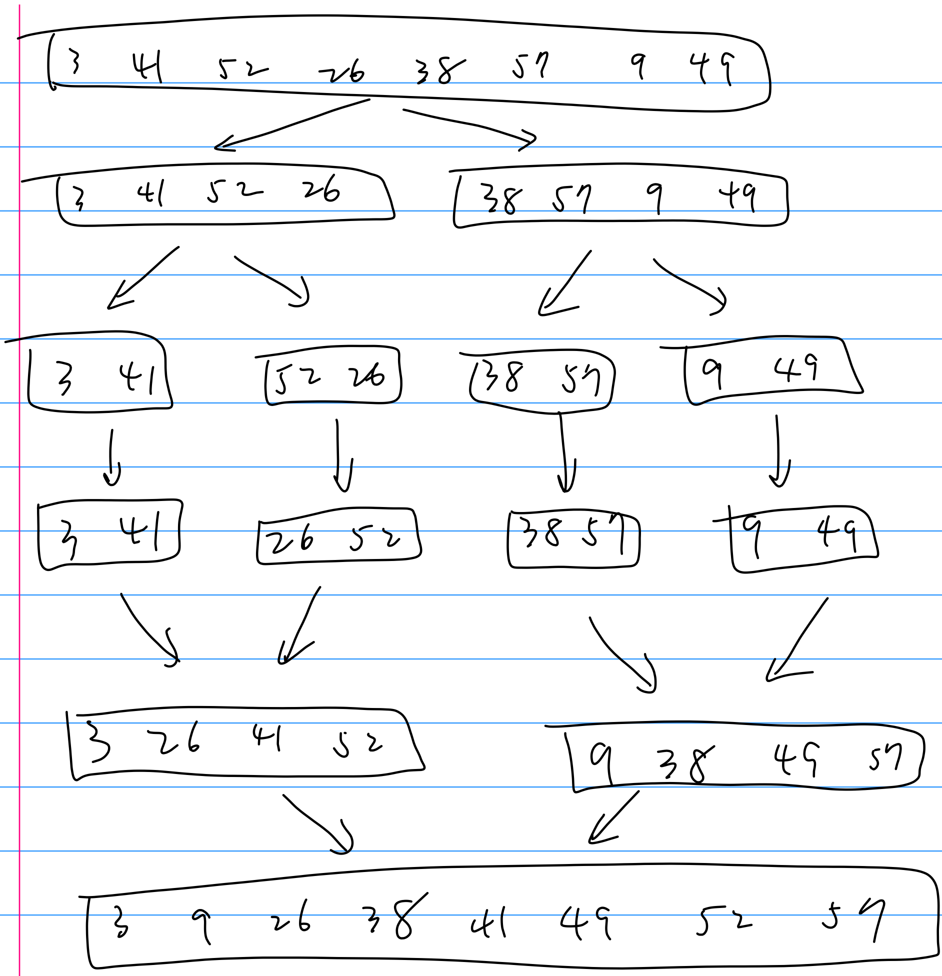
#

- $0 \le (1-p)^{A.length} \le 1 \Rightarrow E(steps) \le \frac{1}{p}$
- $\because A.length \ge 1$, $E(steps) \ge \frac{1}{p} - \frac{1-p}{p} = 1$

$\Rightarrow 1 \le E(steps) \le \frac{1}{p}$

2.3-1

Using Figure 2.4 as a model, illustrate the operation of merge sort on an array initially containing the sequence $\langle 3, 41, 52, 26, 38, 57, 9, 49 \rangle$.



2.3-4
Use mathematical induction to show that when $n \geq 2$ is an exact power of 2, the solution of the recurrence

$$T(n) = \begin{cases} 2 & \text{if } n = 2 , \\ 2T(n/2) + n & \text{if } n > 2 \end{cases}$$

is $T(n) = n \lg n$.

when $n=2$, $T(2) = 2\lg 2 = 2$ ✓

assume when $n=k$, $T(k) = k\lg k$

when $n = 2k$

$T(2k) = 2T(\frac{2k}{2}) + 2k$

$= 2 \cdot k\lg k + 2k$

$= 2k(\lg k + 1)$ $\lg 2$

$= 2k \lg 2k$

by induction, $T(n) = n\lg n$ when $n \geq 2$

and is an exact power of 2

#

2-2 *Correctness of bubblesort*

Bubblesort is a popular, but inefficient, sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order. The procedure BUBBLESORT sorts array $A[1:n]$.

```
BUBBLESORT(A, n)
1  for i = 1 to n - 1
2      for j = n downto i + 1
3          if A[j] < A[j - 1]
4              exchange A[j] with A[j - 1]
```

i

A_1 A_2 A_3 A_4 A_5 A_6

a. Let A' denote the array A after BUBBLESORT(A, n) is executed. To prove that BUBBLESORT is correct, you need to prove that it terminates and that

$$A'[1] \le A'[2] \le \cdots \le A'[n] . \tag{2.5}$$

In order to show that BUBBLESORT actually sorts, what else do you need to prove?

The next two parts prove inequality (2.5).

$A'[i]$ in A $\forall i \in [1, n]$

$\because$ what bubblesort does is just swap elements. $\therefore$ all of the elements in A will in A', vice versa.

b. State precisely a loop invariant for the **for** loop in lines 2–4, and prove that this loop invariant holds. Your proof should use the structure of the loop-invariant proof presented in this chapter.

Loop invariant: Before each iteration, the position of the smallest element in "$A[i:n]$" is at most "j"

Initialization: Before the loop begin "j" is at the bottom of "A" ($\because$ "$j = n$"). So the invariant holds ("j" is already at the rightest edge of "A")

Maintenance: In the iteration, the "if" statement ensure that "$A[j] \ge A[j-1]$" So if the smallest elements in "$A[i:n]$" is not "$A[j]$" before the "if", the variant is still holds. If it is "$A[j]$" before, the exchange will move it to "$A[j-1]$" to keep the invariant hold for the next iteration.

Termination: After the last iteration, the smallest element in "$A[i:n]$" is "$A[i]$"

c. Using the termination condition of the loop invariant proved in part (b), state a loop invariant for the **for** loop in lines 1–4 that allows you to prove inequality (2.5). Your proof should use the structure of the loop-invariant proof presented in this chapter.

Loop invariant: Before the i-th iteration, "A[1:i]" is sorted ascendingly

Initialization: Before the first iteration ("i=1"), "A[1:1]" has only one element. So the invariant holds naturally.

Maintenance: If "A[1:i-1]" is sorted by ascendingly, then after the inner for loop, the smallest element in "A[i:n]" is "A[i]". And the invariant of the inner loop ensure that "A[i]" is larger than every element in "A[1:i-1]". Then, the invariant of outer loop holds for "A[1:i]".

Termination: After the last iteration ("i=n-1"), "A[1:n-1]" is sorted ascendingly, and the invariant of inner loop ensure that "A[n]" is large than every element in "A[1:n-1]". So "A[1:n]" is sorted ascendingly.

d. What is the worst-case running time of BUBBLESORT? How does it compare with the running time of INSERTION-SORT?

Bubblesort need to iterate every element in "A", so the running time is $\Theta(n^2)$, which is the same of insertion-sort.

In the best case, bubble has the running time of $\Theta(n^2)$. Insertion, however, has the running time of $\Theta(n)$ (the input "A" is already sorted ascending)

3.2-1

Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the basic definition of Θ-notation, prove that $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$.

$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0, \text{ s.t. } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \ \forall n > n_0\}$

Assume $f(n) \ge g(n)$ for all n, if not, exchange them.

① $\because$ f, g are nonnegative

$\therefore$ $f(n) \le f(n) + g(n) \Rightarrow c_2 = 1$

② $f(n) + g(n) \le 2f(n)$

$\Rightarrow \frac{1}{2}(f(n) + g(n)) \le f(n) \Rightarrow c_1 = \frac{1}{2}$

①+② $\Rightarrow$ $0 \le c_1(f(n) + g(n)) \le \max\{f(n), g(n)\} = f(n) \le c_2(f(n) + g(n))$

$\Rightarrow \max\{f(n), g(n)\} = \Theta(f(n) + g(n))$

#

3.2-2

Explain why the statement, “The running time of algorithm A is at least $O(n^2)$,” is meaningless.

$O(n^2)$ means "at most n^2", which is conflict with "at least".

3.2-3

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

- $2^{n+1} = 2 \cdot 2^n = O(2^n)$ $\because$ the coefficient 2 is a constant

- $2^{2n} = (2^n)^2 \neq O(2^n)$

  $\quad\quad \shortparallel$

  $\quad\quad O(4^n)$

  $\rightarrow$ you can't find a $C > 0$ s.t. $(2^n)^2 \leq C \cdot 2^n$

  for all large n

3.2-5

Prove that the running time of an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$.

$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0 \text{ s.t. } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \quad \forall n > n_0\}$

$O(g(n)) = \{f(n) : \exists c_2, n_0 > 0 \text{ s.t. } 0 \le f(n) \le c_2 g(n) \quad \forall n > n_0\}$

$\Omega(g(n)) = \{f(n) : \exists c_1, n_0 > 0 \text{ s.t. } 0 \le c_1 g(n) \le f(n) \quad \forall n > n_0\}$

"$\Rightarrow$" trivial

"$\Leftarrow$" trivial

4.3-1

Use the substitution method to show that each of the following recurrences defined on the reals has the asymptotic solution specified:

a. $T(n) = T(n-1) + n$ has solution $T(n) = O(n^2)$.

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&= T(n-2) + (n-1) + n \\
&= T(n-3) + (n-2) + (n-1) + n \\
&\vdots \\
&= \sum_{k=0}^{n} k = \frac{n(n+1)}{2} = O(n^2) \ \#
\end{aligned}
$$

b. $T(n) = T(n/2) + \Theta(1)$ has solution $T(n) = O(\lg n)$.

$$
T(n) = T(\tfrac{n}{2}) + \Theta(1)
$$

$$
\lg n \left\{
\begin{aligned}
&= T(\tfrac{n}{4}) + \Theta(1) + \Theta(1) \\
&= T(\tfrac{n}{8}) + \Theta(1) + \Theta(1) + \Theta(1) \\
&\ \vdots \\
&= \lg n \cdot \Theta(1) = O(\lg n)
\end{aligned}
\right.
$$

d. $T(n) = 2T(n/2 + 17) + n$ has solution $T(n) = O(n \lg n)$.

$$
T(n) = 2T(\tfrac{n}{2} + 17) + n
$$

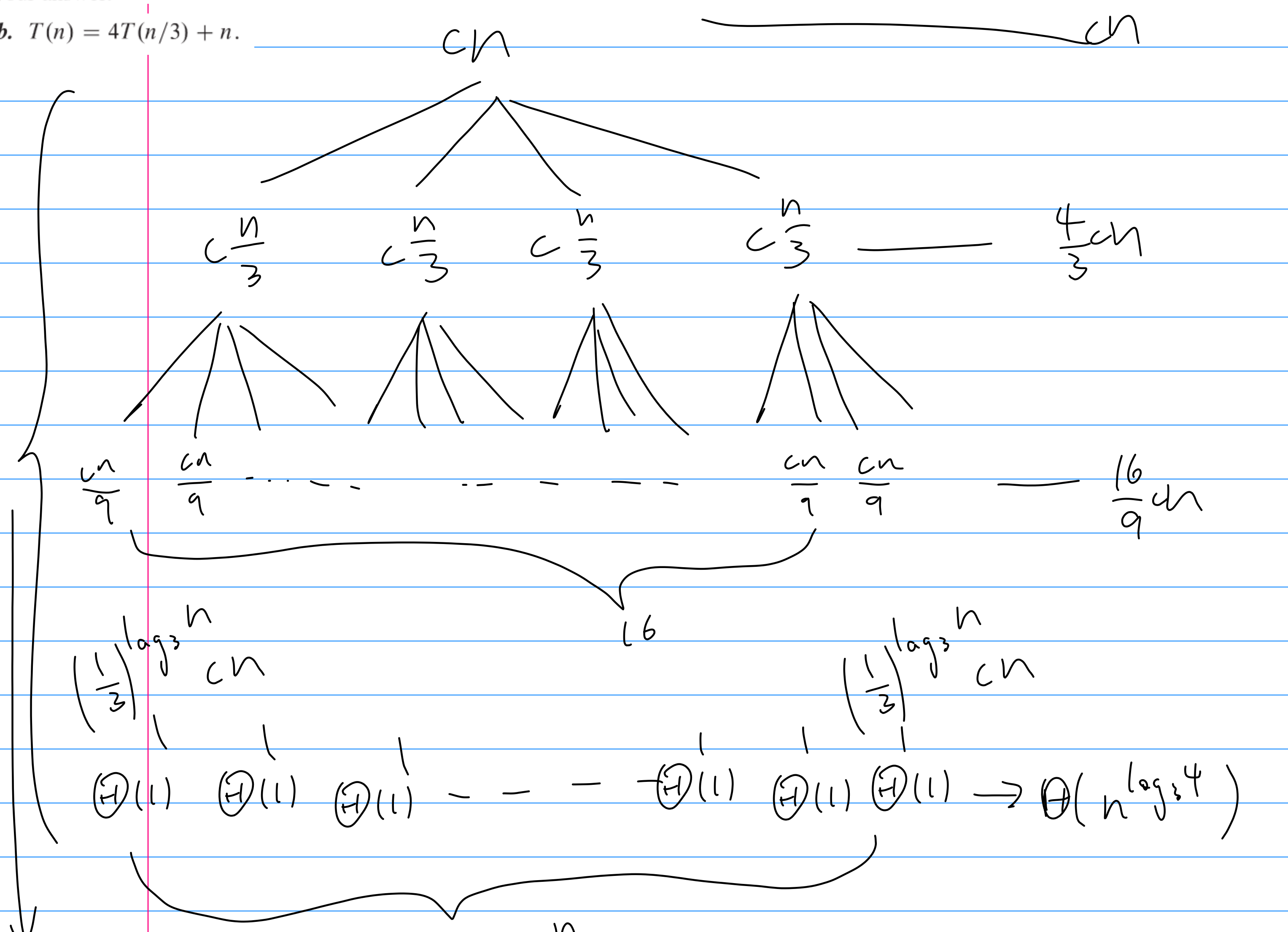
$$
\lg n \left\{
\begin{aligned}
&= 2\left(2T\left(\frac{\frac{n}{2}+17}{2} + 17\right) + \frac{n}{2} + 17\right) + n = 4T\left(\frac{n}{4} + \frac{3}{2}\times 17\right) + 2\times 17 + 2n \\
&= 4\left[2T\left(\frac{1}{2}\left(\frac{n}{4} + \frac{3}{2}\times 17\right) + 17\right) + \frac{n}{4} + \frac{3}{2}\times 17\right] + 2\cdot 17 + 2n \\
&= 8T\left(\frac{n}{8} + \frac{7}{4}\times 17\right) + 3n + 8\times 17 \\
&\ \vdots
\end{aligned}
\right.
$$

$$
= O(n \cdot \lg n)
$$

4.4-1

For each of the following recurrences, sketch its recursion tree, and guess a good asymptotic upper bound on its solution. Then use the substitution method to verify your answer.

b. $T(n) = 4T(n/3) + n$.



$\log_3 n$

$$4^{\log_3 n} = n^{\log_3 4}$$

Use master method to gauss $T(n) = O(n^{\log_3 4})$

* substitution: $T(n) = 4T(\frac{n}{3}) + n$

Assume $T(m) \le c\, m^{\log_3 4} \quad \forall m < n$

$T(n) = 4T(\frac{n}{3}) + n$

$\rightarrow T(n) \le 4c(\frac{n}{3})^{\log_3 4} + n = c\,n^{\log_3 4} \dfrac{4}{\underbrace{\boxed{3^{\log_3 4}}}_{=4}} + n = \underbrace{c\,n^{\log_3 4} + n}$

when n large

$n^{\log_3 4} \approx n^{1.26} >> n$

$\Rightarrow T(n) \le c\,n^{\log_3 4} \Rightarrow T(n) = O(n^{\log_3 4})$ #

4.4-4

Use a recursion tree to justify a good guess for the solution to the recurrence $T(n) = T(\alpha n) + T((1-\alpha)n) + \Theta(n)$, where α is a constant in the range $0 < \alpha < 1$.

0 $\quad \Theta(n)$

1 $\quad \Theta(\alpha n) \quad \Theta((1-\alpha)n)$

2 $\quad \Theta(\alpha^2 n) \quad \Theta(\alpha(1-\alpha)n) \quad \Theta(\alpha(1-\alpha)n) \quad \Theta((1-\alpha)^2 n)$

3 $\quad T(\alpha^3 n) \quad T(\alpha^2(1-\alpha)n) \quad T(\alpha^2(1-\alpha)n) \quad T(\alpha(1-\alpha)^2 n) \quad T(\alpha^2(1-\alpha)n) \quad T(\alpha(1-\alpha)^2 n) \quad T(\alpha(1-\alpha)^2 n) \quad T((1-\alpha)^3 n)$

$\vdots$

$$n: \sum_{r=0}^{n} \binom{n}{r} \Theta\left(\alpha^r (1-\alpha)^{n-r} n\right) = \Theta\left(\underbrace{\sum_{r=0}^{n} \binom{n}{r} \alpha^r (1-\alpha)^{n-r}}_{1} n\right) = \Theta(n)$$

$\Rightarrow$ each level is $\Theta(n)$, there are $\lg n$ level in total

$\Rightarrow T(n) = (n \lg n)$ #

4.5-1

Use the master method to give tight asymptotic bounds for the following recurrences.

a. $T(n) = 2T(n/4) + 1$.

$n^{\log_4 2} = n^{\frac{1}{2}} > 1 = f(n)$

$\Rightarrow T(n) = \Theta(n^{\frac{1}{2}})$ # case I

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

$n^{\log_4 2} = \sqrt{n} < \sqrt{n}\lg^2 n = f(n)$

$\Rightarrow T(n) = \Theta(\sqrt{n}\lg^2 n)$ # case III

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

$n^{\log_4 2} = n^{\frac{1}{2}} < n^2 = f(n)$

$\Rightarrow T(n) = \Theta(n^2)$ # case III

4.5-3

Use the master method to show that the solution to the binary-search recurrence $T(n) = T(n/2) + \Theta(1)$ is $T(n) = \Theta(\lg n)$. (See Exercise 2.3-6 for a description of binary search.)

$\hookrightarrow n^{\log_2 1} = n^0 = 1 = f(n) \Rightarrow$ case 2

$T(n) = \Theta(\underbrace{f(n)}_{1} \lg n) = \Theta(\lg n)$

4.5-4
Consider the function $f(n) = \lg n$. Argue that although $f(n/2) < f(n)$, the regularity condition $af(n/b) \le cf(n)$ with $a = 1$ and $b = 2$ does not hold for any constant $c < 1$. Argue further that for any $\epsilon > 0$, the condition in case 3 that $f(n) = \Omega(n^{\log_b a+\epsilon})$ does not hold.

- $f(\frac{n}{2}) < f(n)$, $f(n) = \lg n$

$f(\frac{n}{2}) = \lg \frac{n}{2} = \lg n - \lg 2 < \lg n = f(n)$ ✓

- $af(n/b) \le cf(n)$

$\xRightarrow[b=2]{a=1} 2f(\frac{n}{2}) \le cf(n) \Rightarrow \underbrace{f(\frac{n}{2})}_{\lg n - \lg 2} \le \underbrace{\frac{c}{2}f(n)}_{\frac{c}{2}\lg n}$

$\Rightarrow (1-\frac{c}{2})\lg n \le \lg 2$

if $c < 1 \Rightarrow \lg n \le \lg 2 \Rightarrow n \le 2$

$\Rightarrow$ the regularity condition doesn't hold. #

---

if case III holds, $f(n) = \Omega(n^{\log_b a - \varepsilon}) = \Omega(n^{\log_2 (1+\varepsilon)})$

for $\varepsilon > 0 \Rightarrow \log_2(1+\varepsilon) > 0 \Rightarrow f(n) \ge \Omega(n) \ge \lg n = f(n)$

$\Rightarrow f(n) \ne \Omega(n^{\log_b (a-\varepsilon)})$, case 3 doesn't hold

4-1 Recurrence examples

Give asymptotically tight upper and lower bounds for $T(n)$ in each of the following algorithmic recurrences. Justify your answers.

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

$n^{\log_2 2} = n < n^3 \Rightarrow$ case 3

check regularity condition : $af\left(\frac{n}{b}\right) \leq cf(n)$

$\Rightarrow 2\left(\frac{n}{2}\right)^3 \leq c\,n^3$

$\quad\; \| $

$2 \cdot \frac{n^3}{8} = \frac{1}{4} n^3$

$\Rightarrow c \geq \frac{1}{4}$ ✓

$\Rightarrow T(n) = \Theta(n^3)$ #

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

$n^{\log_4 16} = n^2 = n^2 \Rightarrow$ case 2

$\Rightarrow T(n) = \Theta(n^2 \lg n)$ #

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

$$
\begin{aligned}
T(n) &= T(n-2) + n^2 \\
&= T(n-4) + (n-2)^2 + n^2 \\
&= T(n-6) + (n-4)^2 + (n-2)^2 + n^2 \\
&\;\vdots \\
&= \sum_{k=0}^{n/2} (n-2k)^2 = \sum_{k=0}^{n/2} n^2 - 4nk + 4k^2 = \Theta(n^3)
\end{aligned}
$$

#

4-3 Solving recurrences with a change of variables

Sometimes, a little algebraic manipulation can make an unknown recurrence similar to one you have seen before. Let's solve the recurrence

$$T(n) = 2T\left(\sqrt{n}\right) + \Theta(\lg n) \tag{4.25}$$

by using the change-of-variables method.

a. Define $m = \lg n$ and $S(m) = T(2^m)$. Rewrite recurrence (4.25) in terms of m and $S(m)$.

$① \Rightarrow n = 2^m, \sqrt{n} = n^{\frac{1}{2}} = 2^{\frac{m}{2}}$

$\Rightarrow T(n) = T(2^m) = 2T(2^{\frac{m}{2}}) + \Theta(m)$

$② \Rightarrow S(m) = 2S\left(\frac{m}{2}\right) + \Theta(m)$ #

b. Solve your recurrence for $S(m)$.

$S(m) = 2S(\frac{m}{2}) + \Theta(m)$

$m^{\log_2 2} = m$

$\xrightarrow{\text{by master method}} S(m) = \Theta(m \lg m) \xlongequal{m=\lg n} \Theta(\lg n \lg(\lg n))$ #

Solve the following recurrences by changing variables:

e. $T(n) = 2T(\sqrt{n}) + \Theta(1)$.

$\xrightarrow{m=\lg n} T(2^m) = 2T(2^{\frac{m}{2}}) + \Theta(1)$

$\xrightarrow{S(m)=T(2^m)} S(m) = 2S(\frac{m}{2}) + \Theta(1)$

$m^{\log_2 2} = m > 1 \Rightarrow$ case 1

$\Rightarrow S(m) = \Theta(m^{\log_2 2}) = \Theta(m) = \Theta(\lg n)$ #

f. $T(n) = 3T(\sqrt[3]{n}) + \Theta(n)$.

$\xrightarrow{m=\log_3 n} T(3^m) = 3T(3^{\frac{m}{3}}) + \Theta(3^m)$

$\xrightarrow{T(3^m)=S(m)} S(m) = 3S(\frac{m}{3}) + \Theta(3^m)$

$m^{\log_3 3} = m < 3^m \Rightarrow$ case 3

$\Rightarrow S(m) = \Theta(3^m) = \Theta(3^{\log_3 n}) = \Theta(n)$ #