2.3-2

The test in line 1 of the MERGE-SORT procedure reads "**if** $p \geq r$" rather than "**if** $p \neq r$." If MERGE-SORT is called with $p > r$, then the subarray $A[p:r]$ is empty. Argue that as long as the initial call of MERGE-SORT$(A, 1, n)$ has $n \geq 1$, the test "**if** $p \neq r$" suffices to ensure that no recursive call has $p > r$.

If we call MERGE-SORT(A,1,n) with $n \geq 1$, then

```
MERGE-SORT(A, p, r)
1  if p ≥ r                          // zero or one element?
2      return
3  q = ⌊(p + r)/2⌋                   // midpoint of A[p : r]
4  MERGE-SORT(A, p, q)               // recursively sort A[p : q]
5  MERGE-SORT(A, q + 1, r)           // recursively sort A[q + 1 : r]
6  // Merge A[p : q] and A[q + 1 : r] into A[p : r].
7  MERGE(A, p, q, r)
```

$q = \lfloor (1+n)/2 \rfloor = r \geq 1 = p$, insuring $r \geq p$

so when $p = r$, A has 1 element, otherwise $p < r$.

2.3-3

State a loop invariant for the **while** loop of lines 12–18 of the MERGE procedure. Show how to use it, along with the **while** loops of lines 20–23 and 24–27, to prove that the MERGE procedure is correct.

variant:

```
MERGE(A, p, q, r)
 1  n_L = q − p + 1          // length of A[p : q]
 2  n_R = r − q              // length of A[q + 1 : r]
 3  let L[0 : n_L − 1] and R[0 : n_R − 1] be new arrays
 4  for i = 0 to n_L − 1     // copy A[p : q] into L[0 : n_L − 1]
 5      L[i] = A[p + i]
 6  for j = 0 to n_R − 1     // copy A[q + 1 : r] into R[0 : n_R − 1]
 7      R[j] = A[q + j + 1]
 8  i = 0                    // i indexes the smallest remaining element in L
 9  j = 0                    // j indexes the smallest remaining element in R
10  k = p                    // k indexes the location in A to fill
11  // As long as each of the arrays L and R contains an unmerged element,
    //     copy the smallest unmerged element back into A[p : r].
12  while i < n_L and j < n_R
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
18      k = k + 1
19  // Having gone through one of L and R entirely, copy the
    //     remainder of the other to the end of A[p : r].
20  while i < n_L
21      A[k] = L[i]
22      i = i + 1
23      k = k + 1
24  while j < n_R
25      A[k] = R[j]
26      j = j + 1
27      k = k + 1
```

2.3-5

You can also think of insertion sort as a recursive algorithm. In order to sort $A[1:n]$, recursively sort the subarray $A[1:n-1]$ and then insert $A[n]$ into the sorted subarray $A[1:n-1]$. Write pseudocode for this recursive version of insertion sort. Give a recurrence for its worst-case running time.

```
Insertion-sort(A, n)
    if n ≤ 1
        return
    Insertion-sort(A, n-1)
    key = A[n]
    j = n - 1

    while j ≥ 0 and A[j] > key
        A[j+1] = A[j]
        j = j - 1
    A[j+1] = key
```

$T(n) = T(n-1) + O(n)$

if it has to scan thrugh the entire array, it will have $O(n^2)$ time complexity.

2.3-8

Describe an algorithm that, given a set S of n integers and another integer x, determines whether S contains two elements that sum to exactly x. Your algorithm should take $\Theta(n \lg n)$ time in the worst case.

```
Algorithm ( A , x )
    sort (A)
    i = 1 , j = n
    while  i < j
        if A[i] + A[j] = x
            return true
        if A [i] + A[j]  < x
            i = i + 1
        else
            j = j - 1
    return false
```

Merge Sort : $O(n \lg n)$

while loop : $O(n)$

$+ = O(n \lg n)$

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]
```

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).

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.

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.

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

a. we need to show that A's is permutations of A.

b. for the inner loop, $A[j]$ being the smallest element in $A[j:n]$ is the loop invariant.

Initialization: when $j=n$, A has 1 element.

Maintenance: before every iteration, $A[j]$ is the smallest element of $A[j:n]$. Then $A[j-1]$ is the smallest element of $A[j-1:n]$. If $A[j] < A[j-1]$, they'd be swapped in line 4.

Termination: The for loop terminates at $j = i+1$. Then $j=i$, and $A[i]$ is the smallest element of $A[i:n]$

c.

loop invariant: the subarray $A[1:i-1]$ consists of $i-1$ smallest element of A in sorted order.

Initialization: when $i=1$, $A[1:i-1]$ is empty.

Maintenance: $A[1:i-1]$ is sorted and its largest element is $A[i-1]$. In part (b), $A[i]$ is the smallest in $A[i:n]$. So $A[1:i]$ contains i smallest elements of A in sorted order

Termination: The loop terminates at $i=n$, and the

subarray $A[1:i-1] = A[1:n-1]$ contains $n-1$ elements in sorted order, which makes $A[n]$ the largest.

d.

$\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$, its worst case running time is $O(n^2)$, same as insertion sort.

3.3-3

Use equation (3.14) or other means to show that $(n + o(n))^k = \Theta(n^k)$ for any real constant k. Conclude that $\lceil n \rceil^k = \Theta(n^k)$ and $\lfloor n \rfloor^k = \Theta(n^k)$.

$O(n) = \{ f(n) \mid \forall n>0, \exists n_0>0 \text{ s.t. } 0 < f(n) < cn, \forall n \geq n_0 \}$

$(n+O(n))^k = n^k (1+\frac{O(n)}{n})^k < n^k(1+c)^k \leq n^k \cdot e^c = O(n^k)$

$(n+O(n))^k \geq n^k \geq cn^k = \Omega(n^k)$, $\forall\ 0 \leq c \leq 1 \Rightarrow (n+O(n))^k = O(n^k)$ *

$n^k < (\lceil n \rceil)^k \leq (n+1)^k$, $(\lceil n \rceil)^k = (n+O(n))^k = O(n^k)$

$[(n-2)+1]^k < (\lfloor n \rfloor)^k \leq n^k$, $(\lfloor n \rfloor)^k = (n-2+O(n))^k = O(n^k)$

3.3-4

Prove the following:

a. Equation (3.21).

b. Equations (3.26)–(3.28).

c. $\lg(\Theta(n)) = \Theta(\lg n)$.

$$\log_b a = \frac{1}{\log_a b}, \qquad a^{\log_b c} = c^{\log_b a}, \qquad n! = o(n^n), \quad n! = \omega(2^n), \quad \lg(n!) = \Theta(n \lg n),$$

a.

$\log_b a = \frac{\log a}{\log b} = \frac{1}{(\frac{\log b}{\log a})} = \frac{1}{\log_a b}$

$\log_b a^{\log_b c} = \log_b a \cdot \log_b c = \log_b c^{\log_b a} \Rightarrow a^{\log_b c} = c^{\log_b a}$ *

3.3-8

Prove by induction that the ith Fibonacci number satisfies the equation

$$F_i = (\phi^i - \hat{\phi}^i)/\sqrt{5},$$

where ϕ is the golden ratio and $\hat{\phi}$ is its conjugate.

Base case:

$$(\phi^0 - \hat{\phi}^0)/\sqrt{5} = 0 = F_0, \quad (\phi^1 - \hat{\phi}^1)/\sqrt{5} = \frac{(1+\sqrt{5}) - (1-\sqrt{5})}{2\sqrt{5}} = 1 = F_1$$

$i = k$:

$$F_{k-2} = (\phi^{k-2} - \hat{\phi}^{k-2})/\sqrt{5}, \quad F_{k-1} = (\phi^{k-1} - \hat{\phi}^{k-1})/\sqrt{5}$$

$$F_k = F_{k-2} + F_{k-1} = \left[(\phi^{k-2} - \hat{\phi}^{k-2}) + (\phi^{k-1} - \hat{\phi}^{k-1})\right]/\sqrt{5}$$

$$= \left[\phi^{k-2}(\phi+1) - \hat{\phi}^{k-2}(\hat{\phi}+1)\right]/\sqrt{5}$$

$$= (\phi^{k-2} \cdot \phi^2 - \hat{\phi}^{k-2}\hat{\phi}^2)/\sqrt{5} \qquad (\phi+1 = \phi^2,\ \hat{\phi}+1 = \hat{\phi}^2)$$

$$= (\phi^k - \hat{\phi}^k)/\sqrt{5} = F_k \quad \#$$

3-2 *Relative asymptotic growths*

Indicate, for each pair of expressions (A, B) in the table below whether A is O, o, Ω, ω, or Θ of B. Assume that $k \geq 1$, $\epsilon > 0$, and $c > 1$ are constants. Write your answer in the form of the table with "yes" or "no" written in each box.

	A		B	O	o	Ω	ω	Θ
a.	$\lg^k n$	$\leq$	n^ϵ	yes	yes	no	no	no
b.	n^k	$\leq$	c^n	yes	yes	no	no	no
c.	$\sqrt{n}$		$n^{\sin n}$	no	no	no	no	no
d.	2^n	$\geq$	$2^{n/2}$	no	no	yes	yes	no
e.	$n^{\lg c}$	$=$	$c^{\lg n}$	yes	no	yes	no	yes
f.	$\lg(n!)$	$=$	$\lg(n^n)$	yes	no	yes	no	yes

$n^1 \leq n^{\sin n} \leq n^1$

a. $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.

false. let $f(n) = n$, $g(n) = n^2$, $n^2 \neq O(n)$

d. $f(n) = O(g(n))$ implies $2^{f(n)} = O\left(2^{g(n)}\right)$.

false. let $f(n) = 2n$, $g(n) = n$, $2^{2n} = (2^n)^2 \neq O(2^n)$

g. $f(n) = \Theta(f(n/2))$.

false. let $f(n) = 2^{2n}$, $f\left(\frac{n}{2}\right) = 2^n \neq \Theta\left((2^n)^2\right)$

4.7-2

Show that $f(n) = n^2$ satisfies the polynomial-growth condition but that $f(n) = 2^n$ does not.

pf: $\frac{f(n)}{d} \leq f(\psi n) \leq d f(n)$, $d > 1$, $1 \leq \psi \leq \phi$, $f(n) = n^2$

$\frac{n^2}{d} \leq (\psi n)^2 \leq d n^2 \leftrightarrow \frac{n^2}{d\psi^2} \leq n^2 \leq \frac{d}{\psi^2} n^2$

the condition is true since $\frac{1}{d} \leq 1 \leq d$, $\forall d \geq 1$

i.e. $\frac{n^2}{d} \leq (\psi n)^2 \leq d n^2$ *

pf: $2^\psi \cdot 2^n \geq \frac{1}{d} \cdot 2^n \geq d f(n)$, so $f(\psi n) \geq d f(n)$

thus $f(n) = 2^n$ doesn't satisfy the condition.

4.7-3

Let $f(n)$ be a function that satisfies the polynomial-growth condition. Prove that $f(n)$ is asymptotically positive, that is, there exists a constant $n_0 \geq 0$ such that $f(n) \geq 0$ for all $n \geq n_0$.

$\frac{1}{d} f(n) \leq f(\psi n) \leq d f(n)$, $d > 1$, $1 < \psi < \phi$

if $f(n) < 0$, $\forall n$ if $\frac{1}{d} f(n) \leq f(\psi n)$, $f(n) \leq d f(\psi n)$

however, if $f(n) \leq d f(\psi n)$, $d f(n) \leq d^2 f(\psi n)$

$\Rightarrow f(\psi n) \leq d f(n) \leq d^2 f(\psi n)$, but $f(\psi n) \not\leq d^2 f(\psi n)$ since $d > 1$ and $f(\psi n) < 0$

so $f(n)$ is asympotically positive.

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.) $T(n) = 2T(n/2) + n^3$. $a=2, b=2, d=3$, $\log_b a = 1 < d$, $T(n) = \Theta(n^3)$

b. $T(n) = T(8n/11) + n$.

(c.) $T(n) = 16T(n/4) + n^2$. $a=16, b=4, d=2$, $\log_b a = d \Rightarrow T(n) = \Theta(n^2 \lg n)$

(d.) $T(n) = 4T(n/2) + n^2 \lg n$. $a=4, b=2$, $n^{\log_b a} < n^2 \lg n \Rightarrow T(n) = \Theta(n^2 \lg n)$

e. $T(n) = 8T(n/3) + n^2$.

(f.) $T(n) = 7T(n/2) + n^2 \lg n$. $a=7, b=2$, $n^{\log_b a} > n^2 \lg n$, $2 < \lg 7 < 3$, $\Rightarrow T(n) = \Theta(n^{\lg 7})$

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

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

$T(n) = 1 \cdot n^2 + 2 \cdot (n-2)^2 + 4(n-3)^2 + \cdots < cn^3$, $T(n) = O(n^3)$

Let $T(n) = \Omega(n^3)$,

$\Rightarrow T(n) > c(n-2)^3 + n^2 = c(n^3 - 6n^2 + 12n - 4) + n^2 \geq cn^3$

$T(n) = \Omega(n^3) = \Theta(n^3)$ ∎

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 $S(m)$.

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

c. Use your solution for $S(m)$ to conclude that $T(n) = \Theta(\lg n \lg\lg n)$.

d. Sketch the recursion tree for recurrence (4.25), and use it to explain intuitively why the solution is $T(n) = \Theta(\lg n \lg\lg n)$.

a.

$m = \lg n,\ \sqrt{n} = 2^{\frac{1}{2}m} \Rightarrow S(m) = 2T(2^{\frac{1}{2}m}) + \Theta(m)$

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

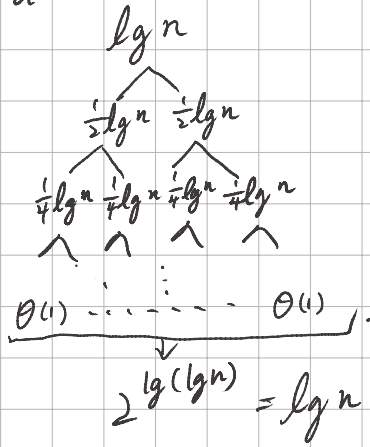
b.

$a = 2,\ b = 2,\ m^{\log_2 2} = m^1 \Rightarrow S(m) = \Theta(m \lg m)$

c.

$T(n) = \Theta(m \lg m) \overset{m=\lg n}{=\!=} \Theta(\lg n \cdot \lg(\lg n))$

d.



→ height = $\lg(\lg n)$

$T(n) = \lg n \lg(\lg n) + \Theta \lg n$

$= \lg n \lg(\lg n)$ ※

$2^{\lg(\lg n)} = \lg n$

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

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

e. $m = \lg n,\ n = 2^m,\ S(m) = T(2^m)$

$T(n) = T(2^m) = S(m)$

$2T(n^{\frac{1}{2}}) = 2T(2^{\frac{1}{2}m}) = 2S(\frac{1}{2}m)$

$S(m) = 2S(\frac{1}{4}m) + \Theta(1)$

$a = 2,\ b = 2 \Rightarrow T(m) = \Theta(m) = \Theta(\lg n)$ ※

f. $m = \log_3 n,\ n = 3^m,\ S(m) = T(3^m)$

$3T(n^{\frac{1}{3}}) = 3T(3^{\frac{1}{3}m}) = 3S(\frac{1}{3}m)$

$\Theta(n) = \Theta(3^m)$

$T(m) = 3^m + 3 \cdot 3^{\frac{1}{3}m} + 3^2 \cdot 3^{\frac{1}{9}m} + \cdots + \Theta(3^m)$

$T(n) = \Theta(n)$