4 Divide-and-Conquer

4.1 The maximum-subarray problem

4.1-1

> What does FIND-MAXIMUM-SUBARRAY return when all elements of A are negative?

It will return the greatest element of A.

4.1-2

> Write pseudocode for the brute-force method of solving the maximum-subarray problem. Your procedure should run in $\Theta(n^2)$ time.

```
BRUTE-FORCE-FIND-MAXIMUM-SUBARRAY(A)
    n = A.length
    max-sum = -∞
    for l = 1 to n
        sum = 0
        for h = l to n
            sum = sum + A[h]
            if sum > max-sum
                max-sum = sum
                low = l
                high = h
    return (low, high, max-sum)
```

4.1-3

> Implement both the brute-force and recursive algorithms for the maximum-subarray problem on your own computer. What problem size n_0 gives the crossover point at which the recursive algorithm beats the brute-force algorithm? Then, change the base case of the recursive algorithm to use the brute-force algorithm whenever the problem size is less than n_0. Does that change the crossover point?

On my computer, n_0 is 37.

If the algorithm is modified to used divide and conquer when $n \geq 37$ and the brute-force approach when n is less, the performance at the crossover point almost doubles. The performance at $n_0 - 1$ stays the same, though (or even gets worse, because of the added overhead).

What I find interesting is that if we set $n_0 = 20$ and used the mixed approach to sort 40 elements, it is still faster than both.

4.1-4

> Suppose we change the definition of the maximum-subarray problem to allow the result to be an empty subarray, where the sum of the values of an empty subarray is 0. How would you change any of the algorithms that do not allow empty subarrays to permit an empty subarray to be the result?

If the original algorithm returns a negative sum, returning an empty subarray instead.

4.1-5

> Use the following ideas to develop a nonrecursive, linear-time algorithm for the maximum-subarray problem. Start at the left end of the array, and progress toward the right, keeping track of the maximum subarray seen so far. Knowing a maximum subarray $A[1..j]$, extend the answer to find a maximum subarray ending at index $j + 1$

> by using the following observation: a maximum subarray $A[i..j+1]$, is either a maximum subarray of $A[1..j]$ or a subarray $A[i..j+1]$, for some $1 \le i \le j+1$. Determine a maximum subarray of the form $A[i..j+1]$ in constant time based on knowing a maximum subarray ending at index j.

```
ITERATIVE-FIND-MAXIMUM-SUBARRAY(A)
    n = A.length
    max-sum = -∞
    sum = -∞
    for j = 1 to n
        currentHigh = j
        if sum > 0
            sum = sum + A[j]
        else
            currentLow = j
            sum = A[j]
        if sum > max-sum
            max-sum = sum
            low = currentLow
            high = currentHigh
    return (low, high, max-sum)
```

4.2 Strassen's algorithm for matrix multiplication

4.2-1

> Use Strassen's algorithm to compute the matrix product
>
> $$\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}.$$
>
> Show your work.

The first matrices are

$$\begin{aligned} S_1 &= 6 & S_6 &= 8 \\ S_2 &= 4 & S_7 &= -2 \\ S_3 &= 12 & S_8 &= 6 \\ S_4 &= -2 & S_9 &= -6 \\ S_5 &= 6 & S_{10} &= 14. \end{aligned}$$

The products are

$$\begin{aligned} P_1 &= 1 \cdot 6 = 6 \\ P_2 &= 4 \cdot 2 = 8 \\ P_3 &= 6 \cdot 12 = 72 \\ P_4 &= -2 \cdot 5 = -10 \\ P_5 &= 6 \cdot 8 = 48 \\ P_6 &= -2 \cdot 6 = -12 \\ P_7 &= -6 \cdot 14 = -84. \end{aligned}$$

The four matrices are

$$\begin{aligned} C_{11} &= 48 + (-10) - 8 + (-12) = 18 \\ C_{12} &= 6 + 8 = 14 \\ C_{21} &= 72 + (-10) = 62 \\ C_{22} &= 48 + 6 - 72 - (-84) = 66. \end{aligned}$$

The result is

$$\begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}.$$

4.2-2

> Write pseudocode for Strassen's algorithm.

```
STRASSEN(A, B)
    n = A.rows
    if n == 1
        return a[1, 1] * b[1, 1]
    let C be a new n × n matrix
    A[1, 1] = A[1..n / 2][1..n / 2]
    A[1, 2] = A[1..n / 2][n / 2 + 1..n]
    A[2, 1] = A[n / 2 + 1..n][1..n / 2]
    A[2, 2] = A[n / 2 + 1..n][n / 2 + 1..n]
    B[1, 1] = B[1..n / 2][1..n / 2]
    B[1, 2] = B[1..n / 2][n / 2 + 1..n]
    B[2, 1] = B[n / 2 + 1..n][1..n / 2]
    B[2, 2] = B[n / 2 + 1..n][n / 2 + 1..n]
    S[1] = B[1, 2] - B[2, 2]
    S[2] = A[1, 1] + A[1, 2]
    S[3] = A[2, 1] + A[2, 2]
    S[4] = B[2, 1] - B[1, 1]
    S[5] = A[1, 1] + A[2, 2]
    S[6] = B[1, 1] + B[2, 2]
    S[7] = A[1, 2] - A[2, 2]
    S[8] = B[2, 1] + B[2, 2]
    S[9] = A[1, 1] - A[2, 1]
    S[10] = B[1, 1] + B[1, 2]
    P[1] = STRASSEN(A[1, 1], S[1])
    P[2] = STRASSEN(S[2], B[2, 2])
    P[3] = STRASSEN(S[3], B[1, 1])
    P[4] = STRASSEN(A[2, 2], S[4])
    P[5] = STRASSEN(S[5], S[6])
    P[6] = STRASSEN(S[7], S[8])
    P[7] = STRASSEN(S[9], S[10])
    C[1..n / 2][1..n / 2] = P[5] + P[4] - P[2] + P[6]
    C[1..n / 2][n / 2 + 1..n] = P[1] + P[2]
    C[n / 2 + 1..n][1..n / 2] = P[3] + P[4]
    C[n / 2 + 1..n][n / 2 + 1..n] = P[5] + P[1] - P[3] - P[7]
    return C
```

4.2-3

> How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2? Show that the resulting algorithm runs in time $\Theta(n^{\lg 7})$.

We can just extend it to an $n \times n$ matrix and pad it with zeroes. It's obviously $\Theta(n^{\lg 7})$.

4.2-4

> What is the largest k such that if you can multiply 3×3 matrices using k multiplications (not assuming commutativity of multiplication), then you can multiply $n \times n$ matrices is time $o(n^{\lg 7})$? What would the running time of this algorithm be?

Assume $n = 3^m$ for some m. Then, using block matrix multiplication, we obtain the recursive running time $T(n) = kT(n/3) + O(1)$.

By master theorem, we can find the largest k to satisfy $\log_3 k < \lg 7$ is $k = 21$.

4.2-5

> V. Pan has discovered a way of multiplying 68×68 matrices using 132464 multiplications, a way of multiplying 70×70 matrices using 143640 multiplications, and a way of multiplying 72×72 matrices using 155424 multiplications. Which method yields the best asymptotic running time when used in a divide-and-conquer matrix-multiplication algorithm? How does it compare to Strassen's algorithm?

Using what we know from the last exercise, we need to pick the smallest of the following

$$\begin{aligned}\log_{68} 132464 &\approx 2.795128\\ \log_{70} 143640 &\approx 2.795122\\ \log_{72} 155424 &\approx 2.795147.\end{aligned}$$

The fastest one asymptotically is 70×70 using 143640.

4.2-6

> How quickly can you multiply a $kn \times n$ matrix by an $n \times kn$ matrix, using Strassen's algorithm as a subroutine? Answer the same question with the order of the input matrices reversed.

- $(kn \times n)(n \times kn)$ produces a $kn \times kn$ matrix. This produces k^2 multiplications of $n \times n$ matrices.
- $(n \times kn)(kn \times n)$ produces an $n \times n$ matrix. This produces k multiplications and $k - 1$ additions.

4.2-7

> Show how to multiply the complex numbers $a + bi$ and $c + di$ using only three multiplications of real numbers. The algorithm should take a, b, c and d as input and produce the real component $ac - bd$ and the imaginary component $ad + bc$ separately.

The three matrices are

$$\begin{aligned}A &= (a + b)(c + d) = ac + ad + bc + bd\\ B &= ac\\ C &= bd.\end{aligned}$$

The result is

$$(B - C) + (A - B - C)i.$$

4.3 The substitution method for solving recurrences

4.3-1

> Show that the solution of $T(n) = T(n - 1) + n$ is $O(n^2)$.

We guess $T(n) \le cn^2$,

$$\begin{aligned}T(n) &\le c(n - 1)^2 + n\\ &= cn^2 - 2cn + c + n\\ &= cn^2 + n(1 - 2c) + c\\ &\le cn^2,\end{aligned}$$

where the last step holds for $c > \frac{1}{2}$.

4.3-2

> Show that the solution of $T(n) = T(\lceil n/2 \rceil) + 1$ is $O(\lg n)$.

We guess $T(n) \le c \lg(n - a)$,

$$\begin{aligned}
T(n) &\le c\lg(\lceil n/2\rceil - a) + 1 \\
&\le c\lg((n+1)/2 - a) + 1 \\
&= c\lg((n+1-2a)/2) + 1 \\
&= c\lg(n+1-2a) - c\lg 2 + 1 && (c \ge 1) \\
&\le c\lg(n+1-2a) && (a \ge 1) \\
&\le c\lg(n-a),
\end{aligned}$$

4.3-3

> We saw that the solution of $T(n) = 2T(\lfloor n/2 \rfloor) + n$ is $O(n\lg n)$. Show that the solution of this recurrence is also $\Omega(n\lg n)$. Conclude that the solution is $\Theta(n\lg n)$.

First, we guess $T(n) \le cn\lg n$,

$$\begin{aligned}
T(n) &\le 2c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + n \\
&\le cn\lg(n/2) + n \\
&= cn\lg n - cn\lg 2 + n \\
&= cn\lg n + (1-c)n \\
&\le cn\lg n,
\end{aligned}$$

where the last step holds for $c \ge 1$.

Next, we guess $T(n) \ge c(n+a)\lg(n+a)$,

$$\begin{aligned}
T(n) &\ge 2c(\lfloor n/2 \rfloor + a)(\lg(\lfloor n/2 \rfloor + a) + n \\
&\ge 2c((n-1)/2 + a)(\lg((n-1)/2 + a)) + n \\
&= 2c\frac{n-1+2a}{2}\lg\frac{n-1+2a}{2} + n \\
&= c(n-1+2a)\lg(n-1+2a) - c(n-1+2a)\lg 2 + n \\
&= c(n-1+2a)\lg(n-1+2a) + (1-c)n - (2a-1)c && (0 \le c < 1, n \ge \frac{(2a-1)c}{1-c}) \\
&\ge c(n-1+2a)\lg(n-1+2a) && (a \ge 1) \\
&\ge c(n+a)\lg(n+a),
\end{aligned}$$

4.3-4

> Show that by making a different inductive hyptohesis, we can overcome the difficulty with the boundary condition $T(1) = 1$ for recurrence (4.19) without adjusting the boundary conditions for the inductive proof.

We guess $T(n) \le n\lg n + n$,

$$\begin{aligned}
T(n) &\le 2(c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + \lfloor n/2 \rfloor) + n \\
&\le 2c(n/2)\lg(n/2) + 2(n/2) + n \\
&= cn\lg(n/2) + 2n \\
&= cn\lg n - cn\lg 2 + 2n \\
&= cn\lg n + (2-c)n \\
&\le cn\lg n + n,
\end{aligned}$$

where the last step holds for $c \ge 1$.

This time, the boundary condition is

$$T(1) = 1 \le cn\lg n + n = 0 + 1 = 1.$$

4.3-5

> Show that $\Theta(n\lg n)$ is the solution to the "exact" recurrence (4.3) for merge sort.

The recurrence is

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \tag{4.3}$$

To show Θ bound, separately show O and Ω bounds.

- For $O(n \lg n)$, we guess $T(n) \le c(n-2)\lg(n-2) - 2c$,

$$\begin{aligned}
T(n) &\le c(\lceil n/2 \rceil - 2)\lg(\lceil n/2 \rceil - 2) + c(\lfloor n/2 \rfloor - 2)\lg(\lfloor n/2 \rfloor - 2) + dn \\
&\le c(n/2 + 1 - 2)\lg(n/2 + 1 - 2) - 2c + c(n/2 - 2)\lg(n/2 - 2) - 2c + dn \\
&\le c(n/2 - 1)\lg(n/2 - 1) + c(n/2 - 1)\lg(n/2 - 1) + dn \\
&= c\frac{n-2}{2}\lg\frac{n-2}{2} + c\frac{n-2}{2}\lg\frac{n-2}{2} - 4c + dn \\
&= c(n-2)\lg\frac{n-2}{2} - 4c + dn \\
&= c(n-2)\lg(n-2) - c(n-2) - 4c + dn \\
&= c(n-2)\lg(n-2) + (d-c)n + 2c - 4c \\
&\le c(n-2)\lg(n-2) - 2c,
\end{aligned}$$

  where the last step holds for $c > d$.

- For $\Omega(n \lg n)$, we guess $T(n) \ge c(n+2)\lg(n+2) + 2c$,

$$\begin{aligned}
T(n) &\ge c(\lceil n/2 \rceil + 2)\lg(\lceil n/2 \rceil + 2) + c(\lfloor n/2 \rfloor + 2)\lg(\lfloor n/2 \rfloor + 2) + dn \\
&\ge c(n/2 + 2)\lg(n/2 + 2) + 2c + c(n/2 - 1 + 2)\lg(n/2 - 1 + 2) + 2c + dn \\
&\ge c(n/2 + 1)\lg(n/2 + 1) + c(n/2 + 1)\lg(n/2 + 1) + 4c + dn \\
&\ge c\frac{n+2}{2}\lg\frac{n+2}{2} + c\frac{n+2}{2}\lg\frac{n+2}{2} + 4c + dn \\
&= c(n+2)\lg\frac{n+2}{2} + 4c + dn \\
&= c(n+2)\lg(n+2) - c(n+2) + 4c + dn \\
&= c(n+2)\lg(n+2) + (d-c)n - 2c + 4c \\
&\ge c(n+2)\lg(n+2) + 2c,
\end{aligned}$$

  where the last step holds for $d > c$.

4.3-6

> Show that the solution to $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ is $O(n \lg n)$.

We guess $T(n) \le c(n-a)\lg(n-a)$,

$$\begin{aligned}
T(n) &\le 2c(\lfloor n/2 \rfloor + 17 - a)\lg(\lfloor n/2 \rfloor + 17 - a) + n \\
&\le 2c(n/2 + 17 - a)\lg(n/2 + 17 - a) + n \\
&= c(n + 34 - 2a)\lg\frac{n + 34 - 2a}{2} + n \\
&= c(n + 34 - 2a)\lg(n + 34 - 2a) - c(n + 34 - 2a) + n && (c > 1, n > n_0 = f(a)) \\
&\le c(n + 34 - 2a)\lg(n + 34 - 2a) && (a \ge 34) \\
&\le c(n-a)\lg(n-a).
\end{aligned}$$

4.3-7

> Using the master method in Section 4.5, you can show that the solution to the recurrence $T(n) = 4T(n/3) + n$ is $T(n) = \Theta(n^{\log_3 4})$. Show that a substitution proof with the assumption $T(n) \le cn^{\log_3 4}$ fails. Then show how to subtract off a lower-order term to make the substitution proof work.

We guess $T(n) \le cn^{\log_3 4}$ first,

$$\begin{aligned}
T(n) &\le 4c(n/3)^{\log_3 4} + n \\
&= cn^{\log_3 4} + n.
\end{aligned}$$

We stuck here.

We guess $T(n) \le cn^{\log_3 4} - dn$ again,

$$\begin{aligned} T(n) &\le 4(c(n/3)^{\log_3 4} - dn/3) + n \\ &= 4(cn^{\log_3 4}/4 - dn/3) + n \\ &= cn^{\log_3 4} - \frac{4}{3}dn + n \\ &\le cn^{\log_3 4} - dn, \end{aligned}$$

where the last step holds for $d \ge 3$.

4.3-8

> Using the master method in Section 4.5, you can show that the solution to the recurrence $T(n) = 4T(n/2) + n$ is $T(n) = \Theta(n^2)$. Show that a substitution proof with the assumption $T(n) \le cn^2$ fails. Then show how to subtract off a lower-order term to make the substitution proof work.

First, let's try the guess $T(n) \le cn^2$. Then, we have

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4c(n/2)^2 + n \\ &= cn^2 + n. \end{aligned}$$

We can't proceed any further from the inequality above to conclude $T(n) \le cn^2$.

Alternatively, let us try the guess

$$T(n) \le cn^2 - cn,$$

which subtracts off a lower-order term. Now we have

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &= 4\left(c(n/2)^2 - c(n/2)\right) + n \\ &= 4c(n/2)^2 - 4c(n/2) + n \\ &= cn^2 + (1 - 2c)n \\ &\le cn^2, \end{aligned}$$

where the last step holds for $c \ge 1/2$.

4.3-9

> Solve the recurrence $T(n) = 3T(\sqrt{n}) + \log n$ by making a change of variables. Your solution should be asymptotically tight. Do not worry about whether values are integral.

First,

$$\begin{aligned} T(n) &= 3T(\sqrt{n}) + \lg n \qquad \text{let } m = \lg n \\ T(2^m) &= 3T(2^{m/2}) + m \\ S(m) &= 3S(m/2) + m. \end{aligned}$$

Now we guess $S(m) \le cm^{\lg 3} + dm$,

$$\begin{aligned} S(m) &\le 3\left(c(m/2)^{\lg 3} + d(m/2)\right) + m \\ &\le cm^{\lg 3} + (\frac{3}{2}d + 1)m \qquad (d \le -2) \\ &\le cm^{\lg 3} + dm. \end{aligned}$$

Then we guess $S(m) \ge cm^{\lg 3} + dm$,

$$\begin{aligned} S(m) &\geq 3\Big(c(m/2)^{\lg 3} + d(m/2)\Big) + m \\ &\geq cm^{\lg 3} + (\frac{3}{2}d + 1)m \qquad\qquad (d \geq -2) \\ &\geq cm^{\lg 3} + dm. \end{aligned}$$

Thus,

$$\begin{aligned} S(m) &= \Theta(m^{\lg 3}) \\ T(n) &= \Theta(\lg^{\lg 3} n). \end{aligned}$$

4.4 The recursion-tree method for solving recurrences

4.4-1

> Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 3T(\lfloor n/2 \rfloor) + n$. Use the substitution method to verify your answer.

- The subproblem size for a node at depth i is $n/2^i$.

  Thus, the tree has $\lg n + 1$ levels and $3^{\lg n} = n^{\lg 3}$ leaves.

  The total cost over all nodes at depth i, for $i = 0, 1, 2, \ldots, \lg n - 1$, is $3^i(n/2^i) = (3/2)^i n$.

$$\begin{aligned} T(n) &= n + \frac{3}{2}n + \left(\frac{3}{2}\right)^2 n + \cdots + \left(\frac{3}{2}\right)^{\lg n - 1} n + \Theta(n^{\lg 3}) \\ &= \sum_{i=0}^{\lg n - 1} \left(\frac{3}{2}\right)^i n + \Theta(n^{\lg 3}) \\ &= \frac{(3/2)^{\lg n} - 1}{(3/2) - 1} n + \Theta(n^{\lg 3}) \\ &= 2[(3/2)^{\lg n} - 1]n + \Theta(n^{\lg 3}) \\ &= 2[n^{\lg(3/2)} - 1]n + \Theta(n^{\lg 3}) \\ &= 2[n^{\lg 3 - \lg 2} - 1]n + \Theta(n^{\lg 3}) \\ &= 2[n^{\lg 3 - 1 + 1} - n] + \Theta(n^{\lg 3}) \\ &= O(n^{\lg 3}). \end{aligned}$$

- We guess $T(n) \leq cn^{\lg 3} - dn$,

$$\begin{aligned} T(n) &= 3T(\lfloor n/2 \rfloor) + n \\ &\leq 3 \cdot (c(n/2)^{\lg 3} - d(n/2)) + n \\ &= (3/2^{\lg 3})cn^{\lg 3} - (3d/2)n + n \\ &= cn^{\lg 3} + (1 - 3d/2)n, \end{aligned}$$

  where the last step holds for $d \geq 2$.

4.4-2

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

- The subproblem size for a node at depth i is $n/2^i$.

  Thus, the tree has $\lg n + 1$ levels and $1^{\lg n} = 1$ leaf.

  The total cost over all nodes at depth i, for $i = 0, 1, 2, \ldots, \lg n - 1$, is $1^i(n/2^i)^2 = (1/4)^i n^2$.

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\lg n-1} \left(\frac{1}{4}\right)^i n^2 + \Theta(1) \\
&< \sum_{i=0}^{\infty} \left(\frac{1}{4}\right)^i n^2 + \Theta(1) \\
&= \frac{1}{1-(1/4)} n^2 + \Theta(1) \\
&= \Theta(n^2).
\end{aligned}$$

- We guess $T(n) \le cn^2$,

$$\begin{aligned}
T(n) &\le c(n/2)^2 + n^2 \\
&= cn^2/4 + n^2 \\
&= (c/4+1)n^2 \\
&\le cn^2,
\end{aligned}$$

  where the last step holds for $c \ge 4/3$.

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.

- The subproblem size for a node at depth i is $n/2^i$.

  Thus, the tree has $\lg n + 1$ levels and $4^{\lg n} = n^2$ leaves.

  The total cost over all nodes at depth i, for $i = 0, 1, 2, \ldots, \lg n - 1$, is $4^i(n/2^i + 2) = 2^i n + 2 \cdot 4^i$.

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\lg n-1} (2^i n + 2 \cdot 4^i) + \Theta(n^2) \\
&= \sum_{i=0}^{\lg n-1} 2^i n + \sum_{i=0}^{\lg n-1} 2 \cdot 4^i + \Theta(n^2) \\
&= \frac{2^{\lg n}-1}{2-1} n + 2 \cdot \frac{4^{\lg n}-1}{4-1} + \Theta(n^2) \\
&= (2^{\lg n}-1)n + \frac{2}{3}(4^{\lg n}-1) + \Theta(n^2) \\
&= (n-1)n + \frac{2}{3}(n^2-1) + \Theta(n^2) \\
&= \Theta(n^2).
\end{aligned}$$

- We guess $T(n) \le c(n^2 - dn)$,

$$\begin{aligned}
T(n) &= 4T(n/2+2) + n \\
&\le 4c[(n/2+2)^2 - d(n/2+2)] + n \\
&= 4c(n^2/4 + 2n + 4 - dn/2 - 2d) + n \\
&= cn^2 + 8cn + 16c - 2cdn - 8cd + n \\
&= cn^2 - cdn + 8cn + 16c - cdn - 8cd + n \\
&= c(n^2 - dn) - (cd - 8c - 1)n - (d-2) \cdot 8c \\
&\le c(n^2 - dn),
\end{aligned}$$

  where the last step holds for $cd - 8c - 1 \ge 0$.

4.4-4

> Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 2T(n-1) + 1$. Use the substitution method to verify your answer.

- The subproblem size for a node at depth i is $n - i$.

  Thus, the tree has $n + 1$ levels ($i = 0, 1, 2, \dots, n$) and 2^n leaves.

  The total cost over all nodes at depth i, for $i = 0, 1, 2, \dots, n-1$, is 2^i.

  The n-th level has 2^n leaves each with cost $\Theta(1)$, so the total cost of the n-th level is $\Theta(2^n)$.

  Adding the costs of all the levels of the recursion tree we get the following:

$$\begin{aligned} T(n) &= \sum_{i=0}^{n-1} 2^i + \Theta(2^n) \\ &= \frac{2^n - 1}{2 - 1} + \Theta(2^n) \\ &= 2^n - 1 + \Theta(2^n) \\ &= \Theta(2^n). \end{aligned}$$

- We guess $T(n) \le c2^n - d$,

$$\begin{aligned} T(n) &\le 2(c2^{n-1} - d) + 1 \\ &= c2^n - 2d + 1 \\ &\le c2^n - d \end{aligned}$$

  Where the last step holds for $d \ge 1$. Thus $T(n) = O(n^2)$.

4.4-5

> Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = T(n-1) + T(n/2) + n$. Use the substitution method to verify your answer.

This is a curious one. The tree makes it look like it is exponential in the worst case. The tree is not full (not a complete binary tree of height n), but it is not polynomial either. It's easy to show $O(2^n)$ and $\Omega(n^2)$.

To justify that this is a pretty tight upper bound, we'll show that we can't have any other choice. If we have that $T(n) \le cn^k$, when we substitue into the recurrence, the new coefficient for n^k can be as high as $c(1 + \frac{1}{2^k})$ which is bigger than c regardless of how we choose the value c.

- We guess $T(n) \le c2^n - 4n$,

$$\begin{aligned} T(n) &\le c2^{n-1} - 4(n-1) + c2^{n/2} - 4n/2 + n \\ &= c(2^{n-1} + 2^{n/2}) - 5n + 4 && (n \ge 1/4) \\ &\le c(2^{n-1} + 2^{n/2}) - 4n && (n \ge 2) \\ &= c(2^{n-1} + 2^{n-1}) - 4n \\ &\le c2^n - 4n \\ &= O(2^n). \end{aligned}$$

- We guess $T(n) \ge cn^2$,

$$\begin{aligned} T(n) &\ge c(n-1)^2 + c(n/2)^2 + n \\ &= cn^2 - 2cn + c + cn^2/4 + n \\ &= (5/4)cn^2 + (1 - 2c)n + c \\ &\ge cn^2 + (1 - 2c)n + c && (c \le 1/2) \\ &\ge cn^2 \\ &= \Omega(n^2). \end{aligned}$$

4.4-6

> Argue that the solution to the recurrence $T(n) = T(n/3) + T(2n/3) + cn$, where c is a constant, is $\Omega(n \lg n)$ by appealing to the recursion tree.

We know that the cost at each level of the tree is cn by examining the tree in figure 4.6. To find a lower bound on the cost of the algorithm, we need a lower bound on the height of the tree.

The shortest simple path from root to leaf is found by following the leftest child at each node. Since we divide by 3 at each step, we see that this path has length $\log_3 n$. Therefore, the cost of the algorithm is

$$cn(\log_3 n + 1) \geq cn \log_3 n = \frac{c}{\log_3} n \log n = \Omega(n \log n).$$

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 answer with the substitution method.

- The subproblem size for a node at depth i is $n/2^i$.

  Thus, the tree has $\lg n + 1$ levels and $4^{\lg n} = n^{\lg 4} = n^2$ leaves.

  The total cost over all nodes at depth i, for $i = 0, 1, 2, \ldots, \lg n - 1$, is $4^i(cn/2^i) = 2^i cn$.

$$\begin{aligned} T(n) &= \sum_{i=0}^{\lg n - 1} 2^i cn + \Theta(n^2) \\ &= \frac{2^{\lg n} - 1}{2 - 1} cn + \Theta(n^2) \\ &= \Theta(n^2). \end{aligned}$$

- For $O(n^2)$, we guess $T(n) \leq dn^2 - cn$,

$$\begin{aligned} T(n) &\leq 4d(n/2)^2 - 4c(n/2) + cn \\ &= dn^2 - cn. \end{aligned}$$

- For $\Omega(n^2)$, we guess $T(n) \geq dn^2 - cn$,

$$\begin{aligned} T(n) &\geq 4d(n/2)^2 - 4c(n/2) + cn \\ &= dn^2 - cn. \end{aligned}$$

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.

- The tree has $n/a + 1$ levels.

  The total cost over all nodes at depth i, for $i = 0, 1, 2, \ldots, n/a - 1$, is $c(n - ia)$.

$$\begin{aligned} T(n) &= \sum_{i=0}^{n/a} c(n - ia) + (n/a)ca \\ &= \sum_{i=0}^{n/a} cn - \sum_{i=0}^{n/a} cia + (n/a)ca \\ &= cn^2/a - \Theta(n) + \Theta(n) \\ &= \Theta(n^2). \end{aligned}$$

- For $O(n^2)$, we guess $T(n) \leq cn^2$,

$$\begin{aligned} T(n) &\le c(n-a)^2 + ca + cn \\ &\le cn^2 - 2can + ca + cn \\ &\le cn^2 - c(2an - a - n) && (a > 1/2, n > 2a) \\ &\le cn^2 - cn \\ &\le cn^2 \\ &= \Theta(n^2). \end{aligned}$$

- For $\Omega(n^2)$, we guess $T(n) \ge cn^2$,

$$\begin{aligned} T(n) &\ge c(n-a)^2 + ca + cn \\ &\ge cn^2 - 2acn + ca + cn \\ &\ge cn^2 - c(2an - a - n) && (a < 1/2, n > 2a) \\ &\ge cn^2 + cn \\ &\ge cn^2 \\ &= \Theta(n^2). \end{aligned}$$

4.4-9

> Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(\alpha n) + T((1-\alpha)n) + cn$, where α is a constant in the range $0 < \alpha < 1$, and $c > 0$ is also a constant.

We can assume that $0 < \alpha \le 1/2$, since otherwise we can let $\beta = 1 - \alpha$ and solve it for β.

Thus, the depth of the tree is $\log_{1/\alpha} n$ and each level costs cn. And let's guess that the leaves are $\Theta(n)$,

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_{1/\alpha} n} cn + \Theta(n) \\ &= cn \log_{1/\alpha} n + \Theta(n) \\ &= \Theta(n \lg n). \end{aligned}$$

We can also show $T(n) = \Theta(n \lg n)$ by substitution.

To prove the upper bound, we guess that $T(n) \le dn \lg n$ for a constant $d > 0$,

$$\begin{aligned} T(n) &= T(\alpha n) + T((1-\alpha)n) + cn \\ &\le d\alpha n \lg(\alpha n) + d(1-\alpha)n \lg((1-\alpha)n) + cn \\ &= d\alpha n \lg \alpha + d\alpha n \lg n + d(1-\alpha)n \lg(1-\alpha) + d(1-\alpha)n \lg n + cn \\ &= dn \lg n + dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn \\ &\le dn \lg n, \end{aligned}$$

where the last step holds when $d \ge \frac{-c}{\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)}$.

We can achieve this result by solving the inequality

$$\begin{aligned} dn \lg n + dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn &\le dn \lg n \\ \implies dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn &\le 0 \\ \implies d(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) &\le -c \\ \implies d \ge \frac{-c}{\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)}&, \end{aligned}$$

To prove the lower bound, we guess that $T(n) \ge dn \lg n$ for a constant $d > 0$,

$$\begin{aligned} T(n) &= T(\alpha n) + T((1-\alpha)n) + cn \\ &\ge d\alpha n \lg(\alpha n) + d(1-\alpha)n \lg((1-\alpha)n) + cn \\ &= d\alpha n \lg \alpha + d\alpha n \lg n + d(1-\alpha)n \lg(1-\alpha) + d(1-\alpha)n \lg n + cn \\ &= dn \lg n + dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn \\ &\ge dn \lg n, \end{aligned}$$

where the last step holds when $0 < d \le \frac{-c}{\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)}$.

We can achieve this result by solving the inequality

$$\begin{aligned} dn\lg n + dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn &\ge dn \lg n \\ \implies dn(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) + cn &\ge 0 \\ \implies d(\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)) &\ge -c \\ \implies 0 < d &\le \frac{-c}{\alpha \lg \alpha + (1-\alpha)\lg(1-\alpha)}, \end{aligned}$$

Therefore, $T(n) = \Theta(n \lg n)$.

4.5 The master method for solving recurrences

4.5-1

> Use the master method to give tight asymptotic bounds for the following recurrences:
>
> **a.** $T(n) = 2T(n/4) + 1$.
>
> **b.** $T(n) = 2T(n/4) + \sqrt{n}$.
>
> **c.** $T(n) = 2T(n/4) + n$.
>
> **d.** $T(n) = 2T(n/4) + n^2$.

a. $\Theta(n^{\log_4 2}) = \Theta(\sqrt{n})$.

b. $\Theta(n^{\log_4 2} \lg n) = \Theta(\sqrt{n} \lg n)$.

c. $\Theta(n)$.

d. $\Theta(n^2)$.

4.5-2

> Professor Caesar wishes to develop a matrix-multiplication algorithm that is asymptotically faster than Strassen's algorithm. His algorithm will use the divide-and-conquer method, dividing each matrix into pieces of size $n/4 \times n/4$, and the divide and combine steps together will take $\Theta(n^2)$ time. He needs to determine how many subproblems his algorithm has to create in order to beat Strassen's algorithm. If his algorithm creates a subproblems, then the recurrence for the running time $T(n)$ becomes $T(n) = aT(n/4) + \Theta(n^2)$. What is the largest integer value of a for which Professor Caesar's algorithm would be asymptotically faster than Strassen's algorithm?

Strassen's algorithm has running time of $\Theta(n^{\lg 7})$.

The largest integer a such that $\log_4 a < \lg 7$ is $a = 48$.

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 2.3-5 for a description of binary search.)

$$\begin{aligned} a &= 1, b = 2, \\ f(n) &= \Theta(n^{\lg 1}) = \Theta(1), \\ T(n) &= \Theta(\lg n). \end{aligned}$$

4.5-4

> Can the master method be applied to the recurrence $T(n) = 4T(n/2) + n^2 \lg n$? Why or why not? Give an asymptotic upper bound for this recurrence.

With $a = 4$, $b = 2$, we have $f(n) = n^2 \lg n \neq O(n^{2-\epsilon}) \neq \Omega(n^{2+\epsilon})$, so we cannot apply the master method.

We guess $T(n) \leq cn^2 \lg^2 n$, subsituting $T(n/2) \leq c(n/2)^2 \lg^2(n/2)$ into the recurrence yields

$$\begin{aligned}
T(n) &= 4T(n/2) + n^2 \lg n \\
&\leq 4c(n/2)^2 \lg^2(n/2) + n^2 \lg n \\
&= cn^2 \lg(n/2) \lg n - cn^2 \lg(n/2) \lg 2 + n^2 \lg n \\
&= cn^2 \lg^2 n - cn^2 \lg n \lg 2 - cn^2 \lg(n/2) \lg 2 + n^2 \lg n \\
&= cn^2 \lg^2 n + (1 - c \lg 2) n^2 \lg n - cn^2 \lg(n/2) \lg 2 \qquad (c \geq 1/\lg 2) \\
&\leq cn^2 \lg^2 n - cn^2 \lg(n/2) \lg 2 \\
&\leq cn^2 \lg^2 n.
\end{aligned}$$

Exercise 4.6-2 is the general case for this.

4.5-5 ⋆

> Consider the regularity condition $af(n/b) \leq cf(n)$ for some constant $c < 1$, which is part of case 3 of the master theorem. Give an example of constants $a \geq 1$ and $b > 1$ and a function $f(n)$ that satisfies all the conditions in case 3 of the master theorem, except the regularity condition.

$a = 1$, $b = 2$ and $f(n) = n(2 - \cos n)$.

If we try to prove it,

$$\begin{aligned}
\frac{n}{2}(2 - \cos\frac{n}{2}) &< cn \\
\frac{1 - \cos(n/2)}{2} &< c \\
1 - \frac{\cos(n/2)}{2} &\leq c.
\end{aligned}$$

Since $\min \cos(n/2) = -1$, this implies that $c \geq 3/2$. But $c < 1$.

4.6 Proof of the master theorem

4.6-1 ⋆

> Give a simple and exact expression for n_j in equation (4.27) for the case in which b is a positive integer instead of an arbitrary real number.

We state that $\forall j \geq 0, n_j = \left\lceil \frac{n}{b^j} \right\rceil$.

Indeed, for $j = 0$ we have from the recurrence's base case that $n_0 = n = \left\lceil \frac{n}{b^0} \right\rceil$.

Now, suppose $n_{j-1} = \left\lceil \frac{n}{b^{j-1}} \right\rceil$ for some $j > 0$. By definition, $n_j = \left\lceil \frac{n_{j-1}}{b} \right\rceil$.

It follows from the induction hypothesis that $n_j = \left\lceil \frac{\left\lceil \frac{n}{b^{j-1}} \right\rceil}{b} \right\rceil$.

Since b is a positive integer, equation (3.4) implies that $\left\lceil \frac{\left\lceil \frac{n}{b^{j-1}} \right\rceil}{b} \right\rceil = \left\lceil \frac{n}{b^j} \right\rceil$.

Therefore, $n_j = \left\lceil \frac{n}{b^j} \right\rceil$.

P.S. n_j is obtained by shifting the base b representation j positions to the right, and adding 1 if any of the j least significant positions are non-zero.

4.6-2 ⋆

> Show that if $f(n) = \Theta(n^{\log_b a} \lg^k n)$, where $k \geq 0$, then the master recurrence has solution $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$. For simplicity, confine your analysis to exact powers of b.

$$
\begin{aligned}
g(n) &= \sum_{j=0}^{\log_b n-1} a^j f(n/b^j) \\
f(n/b^j) &= \Theta\left((n/b^j)^{\log_b a} \lg^k (n/b^j)\right) \\
g(n) &= \Theta\left(\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} \lg^k \left(\frac{n}{b^j}\right)\right) \\
&= \Theta(A) \\
A &= \sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} \lg^k \frac{n}{b^j} \\
&= n^{\log_b a} \sum_{j=0}^{\log_b n-1} \left(\frac{a}{b^{\log_b a}}\right)^j \lg^k \frac{n}{b^j} \\
&= n^{\log_b a} \sum_{j=0}^{\log_b n-1} \lg^k \frac{n}{b^j} \\
&= n^{\log_b a} B \\
\lg^k \frac{n}{d} &= (\lg n - \lg d)^k = \lg^k n + o(\lg^k n) \\
B &= \sum_{j=0}^{\log_b n-1} \lg^k \frac{n}{b^j} \\
&= \sum_{j=0}^{\log_b n-1} \left(\lg^k n - o(\lg^k n)\right) \\
&= \log_b n \lg^k n + \log_b n \cdot o(\lg^k n) \\
&= \Theta(\log_b n \lg^k n) \\
&= \Theta(\lg^{k+1} n) \\
g(n) &= \Theta(A) \\
&= \Theta(n^{\log_b a} B) \\
&= \Theta(n^{\log_b a} \lg^{k+1} n).
\end{aligned}
$$

4.6-3 ⋆

> Show that case 3 of the master method is overstated, in the sense that the regularity condition $af(n/b) \leq cf(n)$ for some constant $c < 1$ implies that there exists a constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$.

$$
\begin{aligned}
af(n/b) &\leq cf(n) \\
\Rightarrow f(n/b) &\leq \frac{c}{a} f(n) \\
\Rightarrow f(n) &\leq \frac{c}{a} f(bn) \\
&= \frac{c}{a}\left(\frac{c}{a} f(b^2 n)\right) \\
&= \frac{c}{a}\left(\frac{c}{a}\left(\frac{c}{a} f(b^3 n)\right)\right) \\
&= \left(\frac{c}{a}\right)^i f(b^i n) \\
\Rightarrow f(b^i n) &\geq \left(\frac{a}{c}\right)^i f(n).
\end{aligned}
$$

Let $n = 1$, then we have

$$f(b^i) \geq \left(\frac{a}{c}\right)^i f(1) \quad (*).$$

Let $b^i = n \Rightarrow i = \log_b n$, then substitue back to equation $(*)$,

$$\begin{aligned} f(n) &\geq \left(\frac{a}{c}\right)^{\log_b n} f(1) \\ &\geq n^{\log_b \frac{a}{c}} f(1) \\ &\geq n^{\log_b a+\epsilon} \qquad \text{where } \epsilon > 0 \text{ because } \frac{a}{c} > a \text{ (recall that } c < 1) \\ &= \Omega(n^{\log_b a+\epsilon}). \end{aligned}$$

Problem 4-1 Recurrence examples

> Give asymptotic upper and lower bound 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.** $T(n) = T(n-2) + n^2$.

a. By master theorem, $T(n) = \Theta(n^4)$.

b. By master theorem, $T(n) = \Theta(n)$.

c. By master theorem, $T(n) = \Theta(n^2 \lg n)$.

d. By master theorem, $T(n) = \Theta(n^2)$.

e. By master theorem, $T(n) = \Theta(n^{\lg 7})$.

f. By master theorem, $T(n) = \Theta(\sqrt{n} \lg n)$.

g. Let $d = m \mod 2$,

$$\begin{aligned} T(n) &= \sum_{j=1}^{j=n/2} (2j + d)^2 \\ &= \sum_{j=1}^{n/2} 4j^2 + 4jd + d^2 \\ &= \frac{n(n+2)(n+1)}{6} + \frac{n(n+2)d}{2} + \frac{d^2 n}{2} \\ &= \Theta(n^3). \end{aligned}$$

Problem 4-2 Parameter-passing costs

> Throughout this book, we assume that parameter passing during procedure calls takes constant time, even if an N-element array is being passed. This assumption is valid in most systems because a pointer to the array is passed, not the array itself. This problem examines the implications of three parameter-passing strategies:

> 1. An array is passed by pointer. Time = $\Theta(1)$.
> 2. An array is passed by copying. Time = $\Theta(N)$, where N is the size of the array.
> 3. An array is passed by copying only the subrage that might be accessed by the called procedure. Time = $\Theta(q-p+1)$ if the subarray $A[p..q]$ is passed.
>
> **a.** Consider the recursive binary search algorithm for finding a number in a sorted array (see Exercise 2.3-5). Give recurrences for the worst-case running times of binary search when arrays are passed using each of the three methods above, and give good upper bounds on the solutions of the recurrences. Let N be the size of the original problems and n be the size of a subproblem.
>
> **b.** Redo part (a) for the MERGE-SORT algorithm from Section 2.3.1.

a.

1. $T(n) = T(n/2) + c = \Theta(\lg n)$. (master method)
2. $\Theta(n \lg n)$.

$$\begin{aligned} T(n) &= T(n/2) + cN \\ &= 2cN + T(n/4) \\ &= 3cN + T(n/8) \\ &= \sum_{i=0}^{\lg n - 1} (2^i cN/2^i) \\ &= cN \lg n \\ &= \Theta(n \lg n). \end{aligned}$$

3. $T(n) = T(n/2) + cn = \Theta(n)$. (master method)

b.

1. $T(n) = 2T(n/2) + cn = \Theta(n \lg n)$. (master method)
2. $\Theta(n^2)$.

$$\begin{aligned} T(n) &= 2T(n/2) + cn + 2N = 4N + cn + 2c(n/2) + 4T(n/4) \\ &= 8N + 2cn + 4c(n/4) + 8T(n/8) \\ &= \sum_{i=0}^{\lg n - 1} (cn + 2^i N) \\ &= \sum_{i=0}^{\lg n - 1} cn + N \sum_{i=0}^{\lg n - 1} 2^i \\ &= cn \lg n + N \frac{2^{\lg n} - 1}{2 - 1} \\ &= cn \lg n + nN - N = \Theta(nN) \\ &= \Theta(n^2). \end{aligned}$$

3. $\Theta(n \lg n)$.

$$\begin{aligned} T(n) &= 2T(n/2) + cn + 2n/2 \\ &= 2T(n/2) + (c+1)n \\ &= \Theta(n \lg n). \end{aligned}$$

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

a. By master theorem, $T(n) = \Theta(n^{\log_3 4})$.

b.

By the recursion-tree method, we can guess that $T(n) = \Theta(n \log_3 \log_3 n)$.

We start by proving the upper bound.

Suppose $k < n \implies T(k) \leq ck \log_3 \log_3 k - k$, where we subtract a lower order term to strengthen our induction hypothesis.

It follows that

$$\begin{aligned} T(n) &\leq 3(c\frac{n}{3}\log_3 \log_3 \frac{n}{3} - \frac{n}{3}) + \frac{n}{\lg n} \\ &\leq cn \log_3 \log_3 n - n + \frac{n}{\lg n} \\ &\leq cn \log_3 \log_3 n, \end{aligned}$$

if n is sufficiently large.

The lower bound can proved analogously.

c. By master theorem, $T(n) = \Theta(n^{2.5})$.

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.

e. By the same reasoning as part (b), the function is $O(n \lg n)$ and $\Omega(n^{1-\epsilon})$ for every ϵ and so is $\tilde{O}(n)$, see Problem 3-5.

f. We guess $T(n) \leq cn$,

$$\begin{aligned} T(n) &= T(n/2) + T(n/4) + T(n/8) + n \\ &\leq \frac{7}{8}cn + n \leq cn. \end{aligned}$$

where the last step holds for $c \geq 8$.

g. Recall that χ_A denotes the indicator function of A. We see that the sum is

$$T(0) + \sum_{j=1}^{n} \frac{1}{j} = T(0) + \int_1^{n+1} \sum_{j=1}^{n+1} \frac{\chi_{j,j+1}(x)}{j} dx.$$

Since $\frac{1}{x}$ is monatonically decreasing, we have that for every $i \in \mathbb{Z}^+$,

$$\sup_{x\in(i,i+1)} \sum_{j=1}^{n+1} \frac{\chi_{j,j+1}(x)}{j} - \frac{1}{x} = \frac{1}{i} - \frac{1}{i+1} = \frac{1}{i(i+1)}.$$

Our expression for $T(n)$ becomes

$$T(N) = T(0) + \int_1^{n+1} \left(\frac{1}{x} + O(\frac{1}{\lfloor x \rfloor(\lfloor x \rfloor + 1)})\right)dx.$$

We deal with the error term by first chopping out the constant amount between 1 and 2 and then bound the error term by $O(\frac{1}{x(x-1)})$ which has an anti-derivative (by method of partial fractions) that is $O(\frac{1}{n})$,

$$\begin{aligned} T(N) &= \int_1^{n+1} \frac{dx}{x} + O(\frac{1}{n}) \\ &= \lg n + T(0) + \frac{1}{2} + O(\frac{1}{n}). \end{aligned}$$

This gets us our final answer of $T(n) = \Theta(\lg n)$.

h. We see that we explicity have

$$\begin{aligned} T(n) &= T(0) + \sum_{j=1}^{n} \lg j \\ &= T(0) + \int_1^{n+1} \sum_{j=1}^{n+1} \chi_{(j,j+1)}(x) \lg j dx. \end{aligned}$$

Similarly to above, we will relate this sum to the integral of $\lg x$.

$$\sup_{x\in(i,i+1)} \sum_{j=1}^{n+1} \chi_{(j,j+1)}(x) \lg j - \lg x = \lg(j+1) - \lg j = \lg\left(\frac{j+1}{j}\right).$$

Therefore,

$$\begin{aligned} T(n) &\le \int_i^n \lg(x+2) + \lg x - \lg(x+1) dx \\ &(1 + O(\frac{1}{\lg n}))\Theta(n \lg n). \end{aligned}$$

i. See the approach used in the previous two parts, we will get $T(n) = \Theta(\frac{n}{\lg n})$.

j. Let i be the smallest i so that $n^{\frac{1}{2^i}} < 2$. We recall from a previous problem (3-6.e) that this is $\lg\lg n$ Expanding the recurrence, we have that it is

$$\begin{aligned} T(n) &= n^{1-\frac{1}{2^i}} T(2) + n + n \sum_{j=1}^{i} \\ &= \Theta(n \lg\lg n). \end{aligned}$$

Problem 4-4 Fibonacci numbers

> This problem develops properties of the Fibonacci numbers, which are defined by recurrence (3.22). We shall use the technique of generating functions to solve the Fibonacci recurrence. Define the ***generating function*** (or ***formal power series***) $\square$ as
>
> $$\begin{aligned} \square(z) &= \sum_{i=0}^{\infty} F_i z^i \\ &= 0 + z + z^2 + 2z^3 + 3z^4 + 5z^5 + 8z^6 + 13z^7 + 21z^8 + \cdots, \end{aligned}$$

where F_i is the ith Fibonacci number.

a. Show that $\square(z) = z + z\square(z) + z^2\square$.

b. Show that

$$\begin{aligned}\square(z) &= \frac{z}{1 - z - z^2} \\ &= \frac{z}{(1 - \phi z)(1 - \hat{\phi} z)} \\ &= \frac{1}{\sqrt{5}}\left(\frac{1}{1 - \phi z} - \frac{1}{1 - \hat{\phi} z}\right),\end{aligned}$$

where

$\phi = \frac{1+\sqrt{5}}{2} = 1.61803\ldots$

and

$\hat{\phi} = \frac{1-\sqrt{5}}{2} = -0.61803\ldots$

c. Show that

$$\square(z) = \sum_{i=0}^{\infty} \frac{1}{\sqrt{5}}(\phi^i - \hat{\phi}^i)z^i.$$

d. Use part (c) to prove that $F_i = \phi^i/\sqrt{5}$ for $i > 0$, rounded to the nearest integer. (Hint: Observe that $|\hat{\phi}| < 1$.)

a.

$$\begin{aligned}z + z\square(z) + z^2\square(Z) &= z + z\sum_{i=0}^{\infty} F_i z^i + z^2 \sum_{i=0}^{\infty} F_i z^i \\ &= z + \sum_{i=1}^{\infty} F_{i-1} z^i + \sum_{i=2}^{\infty} F_{i-2} z^i \\ &= z + F_1 z + \sum_{i=2}^{\infty} (F_{i-1} + F_{i-2}) z^i \\ &= z + F_1 z + \sum_{i=2}^{\infty} F_i z^i \\ &= \square(z).\end{aligned}$$

b. Note that $\phi - \hat{\phi} = \sqrt{5}$, $\phi + \hat{\phi} = 1$ and $\phi\hat{\phi} = -1$.

$$\begin{aligned}
\square(z) &= \frac{\square(z)(1-z-z^2)}{1-z-z^2} \\
&= \frac{\square(z)-z\square(z)-z^2\square(z)-z+z}{1-z-z^2} \\
&= \frac{\square(z)-\square(z)+z}{1-z-z^2} \\
&= \frac{z}{1-z-z^2} \\
&= \frac{z}{1-(\phi+\hat{\phi})z+\phi\hat{\phi}z^2} \\
&= \frac{z}{(1-\phi z)(1-\hat{\phi}z)} \\
&= \frac{\sqrt{5}z}{\sqrt{5}(1-\phi z)(1-\hat{\phi}z)} \\
&= \frac{(\phi-\hat{\phi})z+1-1}{\sqrt{5}(1-\phi z)(1-\hat{\phi}z)} \\
&= \frac{(1-\hat{\phi}z)-(1-\phi z)}{\sqrt{5}(1-\phi z)(1-\hat{\phi}z)} \\
&= \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z}-\frac{1}{1-\hat{\phi}z}\right).
\end{aligned}$$

c. We have $\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k$, when $|x| < 1$, thus

$$\begin{aligned}
\square(n) &= \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z}-\frac{1}{1-\hat{\phi}z}\right) \\
&= \frac{1}{\sqrt{5}}\left(\sum_{i=0}^{\infty}\phi^i z^i - \sum_{i=0}^{\infty}\hat{\phi}^i z^i\right) \\
&= \sum_{i=0}^{\infty}\frac{1}{\sqrt{5}}(\phi^i-\hat{\phi}^i)z^i.
\end{aligned}$$

d. $\square(z) = \sum_{i=0}^{\infty}\alpha_i z^i$ where $\alpha_i = \frac{\phi^i-\hat{\phi}^i}{\sqrt{5}}$. From this follows that $\alpha_i = F_i$, that is

$$F_i = \frac{\phi^i-\hat{\phi}^i}{\sqrt{5}} = \frac{\phi^i}{\sqrt{5}} - \frac{\hat{\phi}^i}{\sqrt{5}},$$

For $i = 1$, $\phi/\sqrt{5} = (\sqrt{5}+5)/10 > 0.5$. For $i > 2$, $|\hat{\phi}^i| < 0.5$.

Problem 4-5 Chip testing

> Professor Diogenes has n supposedly identical integrated-circuit chips that in principle are capable of testing each other. The professor's test jig accommodates two chips at a time. When the jig is loaded, each chip tests the other and reports whether it is good or bad. A good chip always reports accurately whether the other chip is good or bad, but the professor cannot trust the answer of a bad chip. Thus, the four possible outcomes of a test are as follows:

Chip A says	Chip B says	Conclusion
B is good	A is good	both are good, or both are bad
B is good	A is bad	at least one is bad
B is bad	A is good	at least one is bad
B is bad	A is bad	at least one is bad

> **a.** Show that if more than $n/2$ chips are bad, the professor cannot necessarily determine which chips are good using any strategy based on this kind of pairwise test. Assume that the bad chips can conspire to fool the professor.
>
> **b.** Consider the problem of finding a single good chip from among n chips, assuming that more than $n/2$ of the chips are good. Show that $\lfloor n/2 \rfloor$ pairwise tests are sufficient to reduce the problem to one of nearly half the size.
>
> **c.** Show that the good chips can be identified with $\Theta(n)$ pairwise tests, assuming that more than $n/2$ chips are good. Give and solve the recurrence that describes the number of tests.

a. Lets say that there are $g < n/2$ good chips and $n - g$ bad chips.

From this assumption, we can always find a set of good chips G and a set of bad chips B of equal size g since $n - g \geq g$.

Now, assume that chips in B always conspire to fool the professor in the following:

"for any test made by the professor, chips in B declare chips in B as 'good' and chips in G as 'bad'."

Since the chips in G always report correct answers thus there exists symmetric behaviors, it is not possible to distinguish bad chips from good ones.

b.

Generalize the original problem to: "Assume there are more good chips than bad chips."

Algorithm:

1. Pairwise test them, and leave the last one alone if the number of chips is odd.
    - If the report says at least one of them is bad, throw both chips away;
    - otherwise, throw one away from each pair.
2. Recursively find one good chip among the remaining chips. The recursion ends when the number of remaining chips is 1 or 2.
    - If there is only 1 chip left, then it is the good chip we desire.
    - If there are 2 chips left, we make a pairwise test between them. If the report says both are good, we can conclude that both are good chips. Otherwise, one is good and the other is bad and we throw both away. The chip we left alone at step 1 is a good chip.

Explanation:

1. If the number of chips is odd, from assumption we know the number of good chips must be greater than the number of bad chips. We randomly leave one chip alone from the chips, in which good chips are not less than bad chips.
2. Chip pairs that do not say each other is good either have one bad chip or have two bad chips, throwing them away doesn't change the fact that good chips are not less than bad chips.
3. The remaining chip pairs are either both good chips or bad chips, after throwing one chip away in every those pairs, we have reduced the size of the problem to at most half of the original problem size.
4. If the number of good chips is n ($n > 1$) more than that of bad chips, we just throw away the chip we left alone when the number of chips is odd. In this case, the number of good chips is at least one more than that of bad chips, and we can eventually find a good chip as our algorithm claims.
5. If the number of good chips is exactly one more than that of bad chips, there are 2 cases.
    - We left alone the good chip, and remaining chips are one half good and one half bad. In this case, all the chips will be thrown away eventually. And the chip left alone is the one we desire.
    - We left alone the bad chip, there are more good chips than bad chips in the remaining chips. In this case, we can recursively find a good chip in the remaining chips and the left bad chip will be thrown away at the end.

c. As the solution provided in (b), we can find one good chip in

$$T(n) \leq T(\lceil n/2 \rceil) + \lfloor n/2 \rfloor .$$

By the master theorem, we have $T(n) = O(n)$. After finding a good chip, we can identify all good chips with that good chip we just found in $n - 1$ tests, so the total number of tests is

$$O(n) + n - 1 = \Theta(n).$$

Problem 4-6 Monge arrays

An $m \times n$ array A of real numbers is a **Monge array** if for all i, j, k, and l such that $1 \le i < k \le m$ and $1 \le j < l \le n$, we have

$$A[i,j] + A[k,l] \le A[i,l] + A[k,j].$$

In other words, whenever we pick two rows and two columns of a Monge array and consider the four elements at the intersections of the rows and columns, the sum of the upper-left and lower-right elements is less than or equal to the sum of the lower-left and upper-right elements. For example, the following array is Monge:

$$\begin{array}{ccccc}
10 & 17 & 13 & 28 & 23 \\
17 & 22 & 16 & 29 & 23 \\
24 & 28 & 22 & 34 & 24 \\
11 & 13 & 6 & 17 & 7 \\
45 & 44 & 32 & 37 & 23 \\
36 & 33 & 19 & 21 & 6 \\
75 & 66 & 51 & 53 & 34
\end{array}$$

a. Prove that an array is Monge if and only if for all $i = 1, 2, \ldots, m - 1$, and $j = 1, 2, \ldots, n - 1$ we have

$$A[i,j] + A[i+1,j+1] \le A[i,j+1] + A[i+1,j].$$

(Hint: For the "if" part, use induction seperately on rows and columns.)

b. The following array is not Monge. Change one element in order to make it Monge. (Hint: Use part (a).)

$$\begin{array}{cccc}
37 & 23 & 22 & 32 \\
21 & 6 & 7 & 10 \\
53 & 34 & 30 & 31 \\
32 & 13 & 9 & 6 \\
43 & 21 & 15 & 8
\end{array}$$

c. Let $f(i)$ be the index of the column containing the leftmost minimum element of row i. Prove that $f(1) \le f(2) \le \cdots \le f(m)$ for any $m \times n$ Monge array.

d. Here is a description of a divide-and-conquer algorithm that computes the leftmost minimum element in each row of an $m \times n$ Monge array A:

Construct a submatrix A' of A consisting of the even-numbered rows of A. Recursively determine the leftmost minimum for each row in A'. Then compute the leftmost minimum in the odd-numbered rows of A.

Explain how to compute the leftmost minimum in the odd-numbered rows of A (given that the leftmost minimum of the even-numbered rows is known) in $O(m + n)$ time.

e. Write the recurrence describing the running time of the algorithm described in part (d). Show that its solution is $O(m + n \log m)$.

a. The "only if" part is trivial, it follows form the definition of Monge array.

As for the "if" part, let's first prove that

$$\begin{aligned}
&A[i,j] + A[i+1,j+1] \le A[i,j+1] + A[i+1,j] \\
\Rightarrow\ &A[i,j] + A[k,j+1] \le A[i,j+1] + A[k,j],
\end{aligned}$$

where $i < k$.

Let's prove it by induction. The base case of $k = i + 1$ is given. As for the inductive step, we assume it holds for $k = i + n$ and we want to prove it for $k + 1 = i + n + 1$. If we add the given to the assumption, we get

$$\begin{aligned} A[i,j] + A[k,j+1] &\le A[i,j+1] + A[k,j] && \text{(assumption)} \\ A[k,j] + A[k+1,j+1] &\le A[k,j+1] + A[k+1,j] && \text{(given)} \\ \Rightarrow A[i,j] + A[k,j+1] + A[k,j] + A[k+1,j+1] &\le A[i,j+1] + A[k,j] + A[k,j+1] + A[k+1,j] \\ \Rightarrow A[i,j] + A[k+1,j+1] &\le A[i,j+1] + A[k+1,j] \end{aligned}$$

b.

$$\begin{array}{cccc} 37 & 23 & \mathbf{24} & 32 \\ 21 & 6 & 7 & 10 \\ 53 & 34 & 30 & 31 \\ 32 & 13 & 9 & 6 \\ 43 & 21 & 15 & 8 \end{array}$$

c. Let a_i and b_j be the leftmost minimal elements on rows a and b and let's assume that $i > j$. Then we have

$$A[j,a] + A[i,b] \le A[i,a] + A[j,b].$$

But

$$\begin{aligned} A[j,a] &\ge A[i,a] \,(a_i \text{ is minimal}) \\ A[i,b] &\ge A[j,b] \,(b_j \text{ is minimal}) \end{aligned}$$

Which implies that

$$\begin{aligned} A[j,a] + A[i,b] &\ge A[i,a] + A[j,b] \\ A[j,a] + A[i,b] &= A[i,a] + A[j,b] \end{aligned}$$

Which in turn implies that either:

$$\begin{aligned} A[j,b] < A[i,b] &\Rightarrow A[i,a] > A[j,a] \Rightarrow a_i \text{ is not minimal} \\ A[j,b] = A[i,b] &\Rightarrow b_j \text{ is not the leftmost minimal} \end{aligned}$$

d. If μ_i is the index of the i-th row's leftmost minimum, then we have

$$\mu_{i-1} \le \mu_i \le \mu_{i+1}.$$

For $i = 2k + 1$, $k \ge 0$, finding μ_i takes $\mu_{i+1} - \mu_{i-1} + 1$ steps at most, since we only need to compare with those numbers. Thus

$$\begin{aligned} T(m,n) &= \sum_{i=0}^{m/2-1} (\mu_{2i+2} - \mu_{2i} + 1) \\ &= \sum_{i=0}^{m/2-1} \mu_{2i+2} - \sum_{i=0}^{m/2-1} \mu_{2i} + m/2 \\ &= \sum_{i=1}^{m/2} \mu_{2i} - \sum_{i=0}^{m/2-1} \mu_{2i} + m/2 \\ &= \mu_m - \mu_0 + m/2 \\ &= n + m/2 \\ &= O(m+n). \end{aligned}$$

e. The divide time is $O(1)$, the conquer part is $T(m/2)$ and the merge part is $O(m+n)$. Thus,

$$
\begin{aligned}
T(m) &= T(m/2) + cn + dm \\
&= cn + dm + cn + dm/2 + cn + dm/4 + \cdots \\
&= \sum_{i=0}^{\lg m - 1} cn + \sum_{i=0}^{\lg m - 1} \frac{dm}{2^i} \\
&= cn \lg m + dm \sum_{i=0}^{\lg m - 1} \\
&< cn \lg m + 2dm \\
&= O(n \lg m + m).
\end{aligned}
$$