--------------------------- 4 ---------------------------

Divide-and-Conquer (Recurrences)

***Divide-and-Conquer*:**

Divide: **(into the same problems of smaller size)**

Conquer:

Combine:

Two examples of divide-and-conquer: 4.1, 4.2
Solving recurrences: 4.3, 4.4, 4.5

4.1 The maximum-subarray problem

Input: an array $A[1..n]$ of n numbers

Output: a nonempty subarray $A[i..j]$ having the largest sum $S[i, j] = a_i + a_{i+1} + ... + a_j$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

$\underbrace{18 \quad 20 \quad -7 \quad 12}_{\text{maximum subarray}}$

A brute-force solution

* Examine all $\binom{n}{2}$ possible $S[i, j]$

* Two implementations

(1) compute each $S[i, j]$ in $O(n)$ time $\Rightarrow O(n^3)$ time

(2) compute each $S[i, j+1]$ from $S[i, j]$ in $O(1)$ time
($S[i, i] = A[i]$ and $S[i, j+1] = S[i, j] + A[j+1]$)
$\Rightarrow O(n^2)$ time

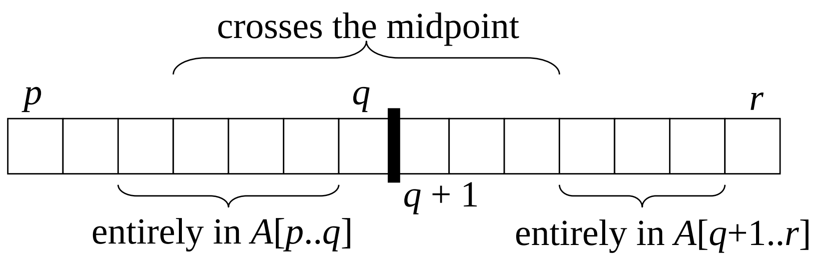
i	1	2	3	4	5	6	7	8	9	10	11	12	13
$A[i]$	13	-15	23	4	-13	-16	-23	18	20	-7	12	-5	-22

$S[2, 2] =$ -15
$S[2, 3] =$ 8
$S[2, 4] =$ 12
$S[2, 5] =$ -1

A divide-and-conquer solution

* Possible locations of a maximum subarray $A[i..j]$ of $A[p..r]$, where $q = \lfloor (p+r)/2 \rfloor$

(1) entirely in $A[p..q]$
(2) entirely in $A[q+1..r]$
(3) crossing the midpoint ($p \le i < q < j \le r$)

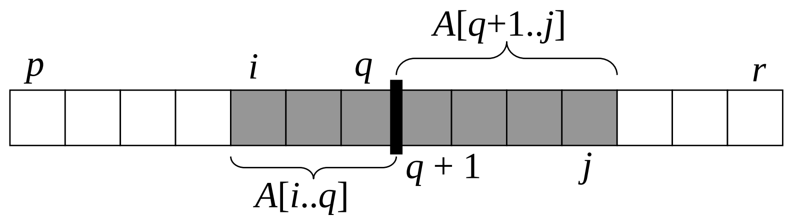


Locations of a maximum subarray $A[i..j]$ of $A[p..r]$

* A divide-and-conquer algorithm

```
FINDMAXSUBARRAY(A, p, r)
if p = r then return (p, p, A[p])     //base case
else
    q ← ⌊(p + r) / 2⌋
    (i1, j1, s1) ← FINDMAXSUBARRAY(A, p, q)
    (i2, j2, s2) ← FINDMAXSUBARRAY(A, q+1, r)
    (ic, jc, sc) ← FINDMAXCROSSING(A, p, q, r)
    if s1 ≥ s2 and s1 ≥ sc then return (i1, j1, s1)
    elseif s2 ≥ sc then return (i2, j2, s2)
    else return (ic, jc, sc)
```


* Find a maximum subarray crossing the midpoint



$A[i..j]$ comprises two subarrays
$A[i..q]$ and $A[q+1..j]$

```
FINDMAXCROSSING(A, p, q, r)
s1 ← −∞
sum ← 0
for i ← q downto p do
    sum ← sum + A[i]
    if sum > s1
       then s1 ← sum
            maxleft ← i
s2 ← −∞
sum ← 0
for j ← q + 1 to r do
    sum ← sum + A[j]
    if sum > s2
       then s2 ← sum
            maxright ← j
return (maxleft, maxright, s1 + s2)
```

Example:

q = 6

A	1	2	3	4	5	6	7	8	9	10	11	12
	-7	8	-5	20	-3	-8	-23	18	20	-7	12	-5

$S[6, 6]$ = -8
$S[5, 6]$ = -11
$S[4, 6]$ = 9
$S[3, 6]$ = 4
$S[2, 6]$ =` 12 $\Leftarrow$ (maxleft = 2)
$S[1..6]$ = 5

q = 6

A	1	2	3	4	5	6	7	8	9	10	11	12
	-7	8	-5	20	-3	-8	-23	18	20	-7	12	-5

$S[7, 7]$ = -23
$S[7, 8]$ = -5
$S[7, 9]$ = 15
$S[7, 10]$ = 8
$S[7, 11]$ =` (maxright = 11) $\Rightarrow$ 20
$S[7, 12]$ = 15

$\Rightarrow$ maximum subarray crossing q is $A[2, 11]$
(with $S[2, 11] = 32$)

* Time complexity

(1) FINDMAXCROSSING: $\Theta(n)$, where $n = r - p + 1$

(2) FINDMAXSUBARRAY:

$$T(n) = 2T(n/2) + \Theta(n) \quad \text{(with } T(1) = \Theta(1))$$
$$= \Theta(n\lg n) \quad \text{(similar to merge-sort)}$$

Remark: See Ex4.1-5 for an $O(n)$-time algorithm.

4.2 Strassen's algorithm for matrix multiplication

Input: two $n \times n$ matrices A and B
Output: $C = AB$, where $c_{i,j} = \sum_{1 \le k \le n} a_{ik} b_{kj}$

An $O(n^3)$ time naive algorithm

```
SQUARE-MATRIX-MULTIPLY(A, B)
n ← rows[A]
let C be an n × n matrix
for i ← 1 to n do
    for j ← 1 to n do
        c_ij ← 0
        for k ← 1 to n do
            c_ij ← c_ij + a_ik • b_kj
return C
```

* Computing $A+B \rightarrow O(n^2)$ time

Strassen's algorithm

* Assume that n is an exact power of 2

* We divide each of A, B, and C into four $n/2 \times n/2$ sub-matrices and rewrite $C = AB$ as

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix} \qquad \text{(EQ-1)}$$

* We have $r = ae + bf$ $\quad s = ag + bh$
  $t = ce + df$ $\quad u = cg + dh$

* A straightforward divide-and-conquer algorithm

$$\begin{aligned} T(n) &= 8T(n/2) + O(n^2) \\ &= O(n^3) \end{aligned}$$

* Let

$$\begin{aligned}
P_1 &= a(g-h) && (=ag-ah) \\
P_2 &= (a+b)h && (=ah+bh) \\
P_3 &= (c+d)e && (=ce+de) \\
P_4 &= d(f-e) && (=df-de) \\
P_5 &= (a+d)(e+h) && (=ae+ah+de+dh) \\
P_6 &= (b-d)(f+h) && (=bf+bh-df-dh) \\
P_7 &= (a-c)(e+g) && (=ae+ag-ce-cg) \text{ (EQ-2)}
\end{aligned}$$

* We have

$$\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7 \qquad \text{(EQ-3)}
\end{aligned}$$

* Strassen's divide-and-conquer algorithm

Step 1: Divide each of A, B, and C into four sub-matrices. (EQ-1)

Step 2: Recursively, compute P_1, P_2, …, P_7. (EQ-2)

Step 3: Compute r, s, t, u according to EQ-3.

* Time complexity

$$\begin{aligned} T(n) &= 7T(n/2) + O(n^2) \\ &= O(n^{\log_2 7}) \\ &= O(n^{2.81}) \end{aligned}$$

Discussion:

1. Strassen's method is largely of theoretical interest. (for $n \geq 45$)

2. Strassen's method is based on the fact that we can multiply two 2×2 matrices using only 7 multiplications (instead of 8). It was showed that it is impossible to multiply two 2×2 matrices using less than 7 multiplications.

3. We can improve Strassen's algorithm by finding an efficient way to multiply two $k \times k$ matrices using a smaller number q of multiplications, where $k > 2$. The time is $T(n) = qT(n/k) + O(n^2)$.

4. The current best upper bound is $O(n^{2.376})$.

4.3 The substitution method

The substitution method: (i) Guess an answer and then (ii) prove it by induction. (for both upper and lower bounds)

Example: Find an upper bound for

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \qquad \text{(with } T(1) = 1\text{)}$$

(i) Guess $T(n) = O(n \lg n)$.

(ii) Try to prove there exist constants c and n_0 such that $T(n) \leq cn \lg n$ for all $n \geq n_0$.

Basis: ($n = n_0$)

For $n = 1$, no constant c satisfies $T(1) \leq cn \lg n = 0$.
For $n \geq 2$, any constant $c \geq T(n)/(n \lg n)$ satisfies $T(n) \leq cn \lg n$. That is, we can choose

(1) $\quad n_0 \geq 2 \quad$ and $\quad c \geq T(n_0)/(n_0 \lg n_0)$.

Induction: ($n > n_0$)

Assume that it holds for all n between n_0 and $n-1$. We have

$$T(n) \le 2(c\lfloor n/2\rfloor \lg \lfloor n/2\rfloor) + n \qquad \textbf{(\textit{Substitution})}$$
$$\le cn \lg (n/2) + n$$
$$= cn \lg n - cn \lg 2 + n$$
$$= cn \lg n - cn + n$$
$$\le cn \lg n,$$

where the last step holds for

(2) $c \ge 1$.

From (1) and (2), we can choose $n_0 = 2, 3$ and $c = \max\{1, T(2)/(2 \lg 2), T(3)/(3 \lg 3)\} = 2$ to make both the *basis* and the *induction* steps holds.

Substitution Method

Step 1. Guess $T(n) = O(g(n))$

Step 2. Prove the guess by induction

Prove $T(n) = O(g(n))$

$\Rightarrow$ Prove that there are c and n_0 such that $T(n) \le cg(n)$ for all $n \ge n_0$ ------------(1)

$\Rightarrow$ If c and n_0 are known, we can prove (1) by induction

(a) Basis step: (1) holds for $n = n_0$
(b) Induction step: (1) holds for $n > n_0$

$\Rightarrow$ How to find c and n_0 satisfying the induction proof?

(i) find the condition of c and n_0 for which the basis step holds
(ii) find the condition of c and n_0 for which the induction step holds
(iii) Combine conditions (i) and (ii)

***Subtleties*:**
(Revise a guess by subtracting a lower-order term.)

Example: $T(n)=T(\lfloor n/2\rfloor)+T(\lceil n/2\rceil)+1$ (with $T(1)=1$)

Guess $T(n) = O(n)$.

Try to prove $T(n) \le cn$.
Basis: ok!

Induction: $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
$\leq c(\lfloor n/2 \rfloor + \lceil n/2 \rceil) + 1$
$= cn + 1$

We can not prove that $T(n) \leq cn$!!!!

Try to prove $T(n) \leq cn - b$.

Induction: $T(n) \leq (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1$
$= cn - 2b + 1$
$\leq cn - b$,

where the last step holds for any constant $b \geq 1$.

Avoiding pitfalls

$T(n) = 2T(\lfloor n/2 \rfloor) + n$

Guess $T(n) = O(n)$. Try to prove $T(n) \leq cn$.

Induction: $T(n) \leq 2c\lfloor n/2 \rfloor + n$
$\leq cn + n$
$= O(n)$ $\Leftarrow$==== ***wrong !!***

Changing variable:

$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n$

For simplicity, assume $n = 2^m$. Then

$T(2^m) = 2T(2^{m/2}) + m$

Let $S(m) = T(2^m)$. We have

$S(m) = 2S(m/2) + m$ (Renaming $m = \lg n$)

Since $O(m \lg m)$ is the solution to $S(m)$, we know that $O(\lg n \lg\lg n)$ is the solution to $T(n)$.

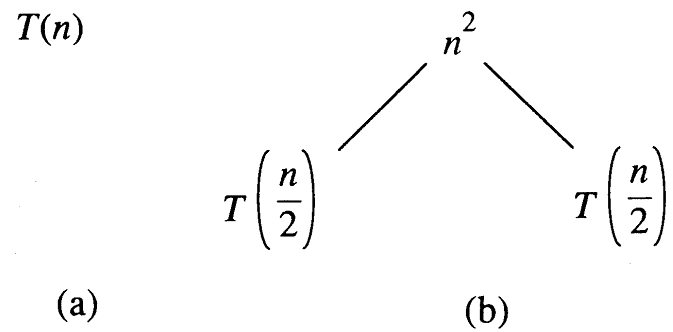
4.4 The iteration (recursion-tree) method

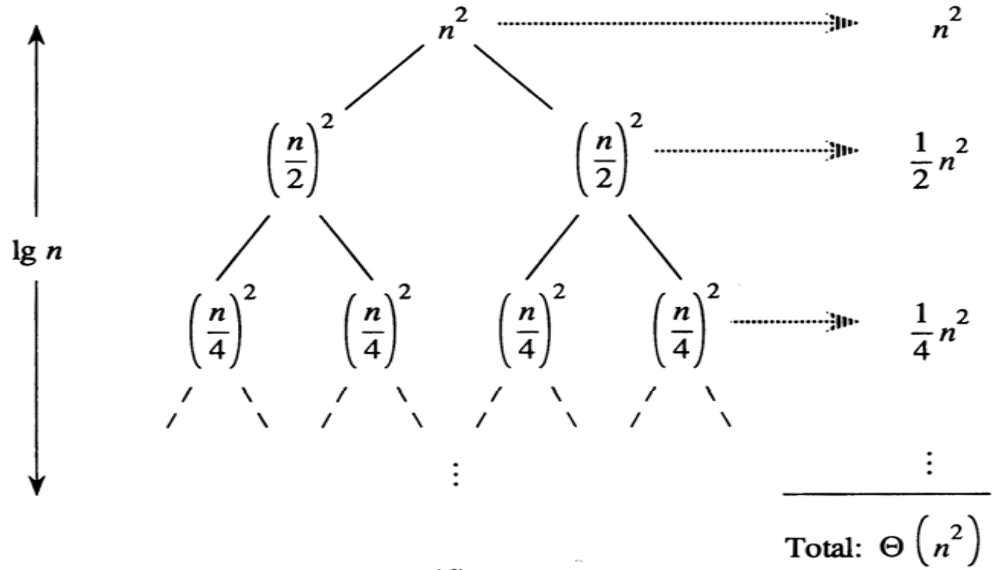
Example: $T(n) = 3T(\lfloor n/4 \rfloor) + n$

$T(n) = n + 3(\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor))$
$= n + 3\lfloor n/4 \rfloor + 9(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor))$
.
. (note that $n/(4^{\log_4 n}) \leq 1$)
$\leq n + 3n/4 + 9n/16 + 27n/64 + \ldots + 3^{\log_4 n}\Theta(1)$
$\leq n\sum_{i=0}^{\infty}(\frac{3}{4})^i + \Theta(n^{\log_4 3}) = 4n + o(n) = O(n)$

Recursion trees: (for visualizing the iteration)

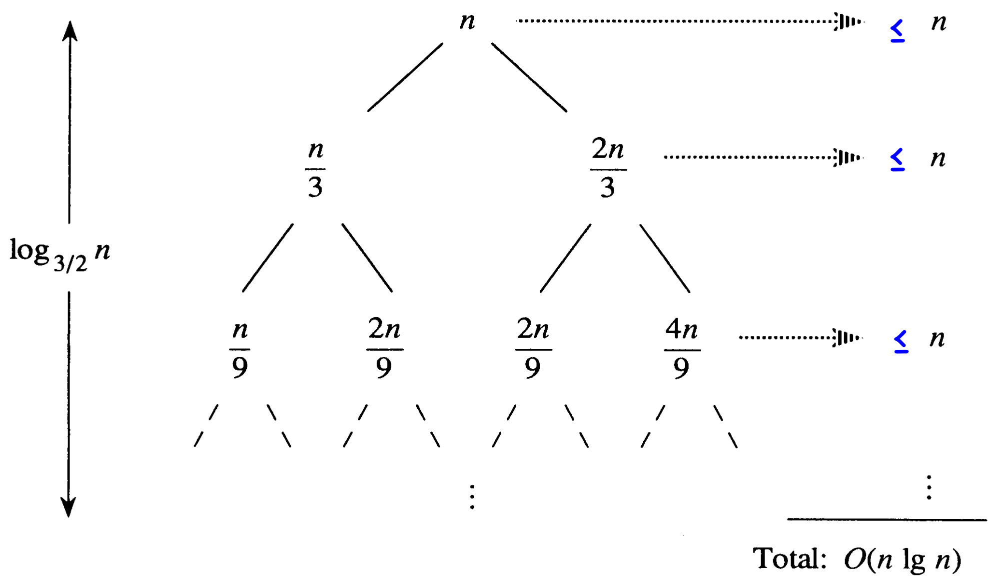
$T(n) = 2T(n/2) + n^2$ (Assume that $n = 2^h$.)





(d)

Example: $T(n) = T(n/3) + T(2n/3) + n$



4.5 The master method

Theorem 4.1 **(*Master theorem*)**
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then, $T(n)$ can be bounded as follows.

1. If $f(n) = O(n^{(\log_b a)-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$
3. If $f(n) = \Omega(n^{(\log_b a)+\varepsilon})$ for some constant $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Example: $T(n) = 9T(n/3) + n$

By applying case 1, we have $T(n) = \Theta(n^2)$.

Example: $T(n) = T(2n/3) + 1$

By applying case 2, we have $T(n) = \Theta(\lg n)$.

Example: $T(n) = 3T(n/4) + n \lg n$

By applying case 3, we have $T(n) = \Theta(n \lg n)$.

Note: The three cases do not cover all the possibilities for $f(n)$. There are gaps between cases 1 and 2, and between cases 2 and 3.

Example: $T(n) = 2T(n/2) + n \lg n$

In this example, both cases 2 and 3 cannot be applied.

Homework: Ex. 4.1-5, 4.2-1, 4.2-4, 4.2-5, 4.2-7, 4.3-5 (using substitution method), 4.4-6, 4.4-9, 4.5-2, and Pro.4-5bc(using substitution method), 4-6de.