CH08-320201
Algorithms and Data Structures

Lecture 5/6 — 20 Feb 2018

Prof. Dr. Michael Sedlmair

Jacobs University
Spring 2018

This & that

- Medical excuse policy refined:
  - http://vis.jacobs-university.de/teaching/ads/18s/
- Anybody signed up anew?
  - email me to add you to Moodle
- Auditing
- Midterm
  - March 20, 9:45am (no lecture on that day then)

Today

- Apply recurrences to Divide & Conquer
  - Power of a number
  - Fibonacci numbers
  - Maximum-subarray problem
  - Matrix multiplication

1.6 Apply recurrences to Divide & Conquer

Recall: Divide & Conquer

Design paradigm:

1. **Divide** the problem (instance) into subproblems.
2. **Conquer** the subproblems by solving them recursively.
3. **Combine** subproblem solutions.

Recall: Merge Sort

1. **Divide**: Trivial.
2. **Conquer**: Recursively sort 2 subarrays.
3. **Combine**: Linear-time merge.

$$T(n) = \underbrace{2}_{\text{\#subproblems}}T(\underbrace{n/2}_{\text{subproblem size}}) + \underbrace{\Theta(n)}_{\text{work dividing and combining}}$$

#subproblems

subproblem size

work dividing
and combining

Master method on Merge Sort

$T(n) = 2T(n/2) + n$
$a = 2,\ b = 2$

$n^{\log_b a} = n$
$f(n) = n$

Case 2: $f(n) = \Theta(n)$,
Thus, $T(n) = \Theta(n \lg n)$.

Power of a number

- Problem:
  - Input: numbers $a \in \mathbf{R}$ and $n \in \mathbf{N}$.
  - Output: a^n
- Naive algorithm:
  - T(n) = Θ(n).
- Divide & Conquer:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

- Recurrence:
  - $T(n) = T(n/2) + \Theta(1)$
- Solution:
  - $a = 1$, $b = 2$, $n^{\log_b a} = 1$, $f(n) = \Theta(1)$ —> **Case 2**.
  - Thus, $\boldsymbol{T(n) = \Theta(\lg n)}$

Fibonacci numbers

Recursive definition:

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

Sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

n 0 1 2 3 4 ...

Output: return the *n*-th Fibonacci number

Fibonacci numbers

Naive algorithm:

Implement the recursion as in the definition.

```
FIBONACCI
if (n < 2)
    return n;
else return FIBONACCI (n - 1) + FIBONACCI (n - 2)
```

$$T(n) = T(n - 1) + T(n - 2) + \Theta(1).$$

Fibonacci numbers

$T(n) = T(n - 1) + T(n - 2) + \Theta(1).$

Lower bound:

$T(n - 1) \approx T(n - 2)$

$T(n) = 2T(n - 2) + \Theta(1)$

$T(n) = 4T(n - 4) + \Theta(1)$

$T(n) = 8T(n - 6) + \Theta(1)$

…

$T(n) = 2^k T(n - 2k) + \Theta(1)$ —> $n - 2k = 0 \Rightarrow k = n/2$

$T(n) = 2^{n/2} T(0) + \Theta(1)$ **—> $\boldsymbol{T(n) = \Omega(2^{n/2})}$,**

i.e., exponential time

Fibonacci numbers

Side note: A tighter lower bound

T(n) = $\Omega(\Phi^n)$,

since #leaves in rec. tree = FIBONACCI (n) x $\Theta(1)$

Φ is the golden ratio

$$\phi = (1+\sqrt{5})/2$$

Fibonacci numbers

Bottom up approach:

Avoid recursion, i.e.,

compute F_0, F_1, F_2, ..., F_n in the given order instead, forming each number by summing the two previous.

$\boldsymbol{T(n) = \Theta(n)}$.

Fibonacci numbers

Use closed form (rounded to next integer):

$$F_n = \phi^n / \sqrt{5} \qquad \phi = (1+\sqrt{5})/2$$

(proof by induction).

Compute by „Power of a number“ recursion.

$\boldsymbol{T(n) = \Theta(\lg n)}$.

But: numerical problems may occur (floating-point arithmetic).

Fibonacci numbers

Use matrix representation:

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

(proof by induction).

Compute by „Power of a number" recursion
(using a generalization to 2x2 matrices).

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

And: uses integers only (no floating-point errors).

Maximum-subarray problem

- Motivation
  Scenario — buy & sell stock
  - Input: a sequence of numbers
  - Output: subsequence that results in the highest profit

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97

Maximum-subarray problem

Brute-Force algorithm

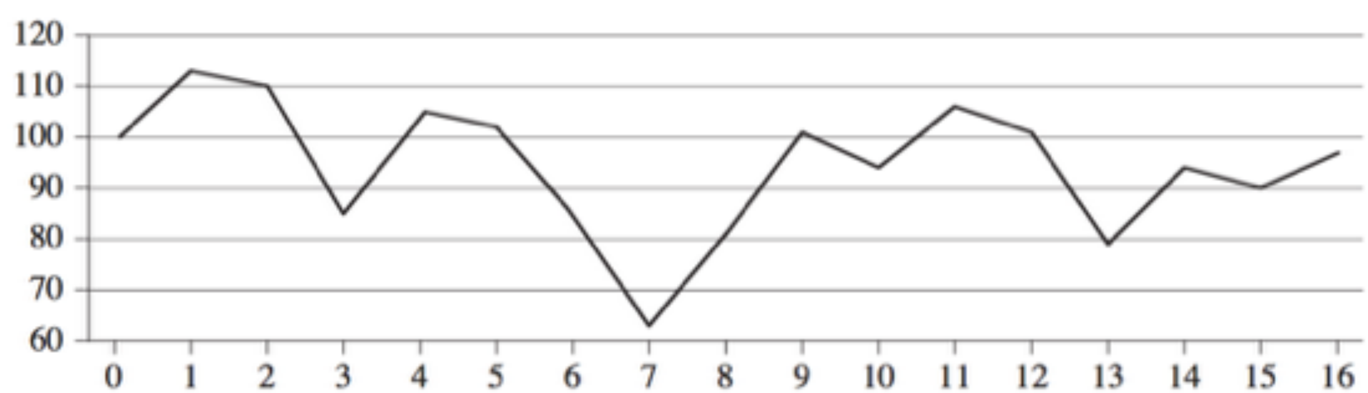
Try every pair of days.

Number of pairs:

$$\binom{n}{k} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2} = \Theta(n^2)$$

Maximum-subarray problem

- Transformation:
  - consider daily change in price instead
- —> **Maximum-subarray problem**
  - Input: a sequence of numbers $\langle a_1, a_2, \ldots, a_n \rangle$ of positive and negative numbers (otherwise it does not make sense)
  - Output: contiguous, non-empty subsequence $\langle a_i, a_{i+1}, \ldots, a_j \rangle$ with $i \geq 1$, $j \leq n$, so that $\sum_{k=i}^{j} a_k$ is maximized



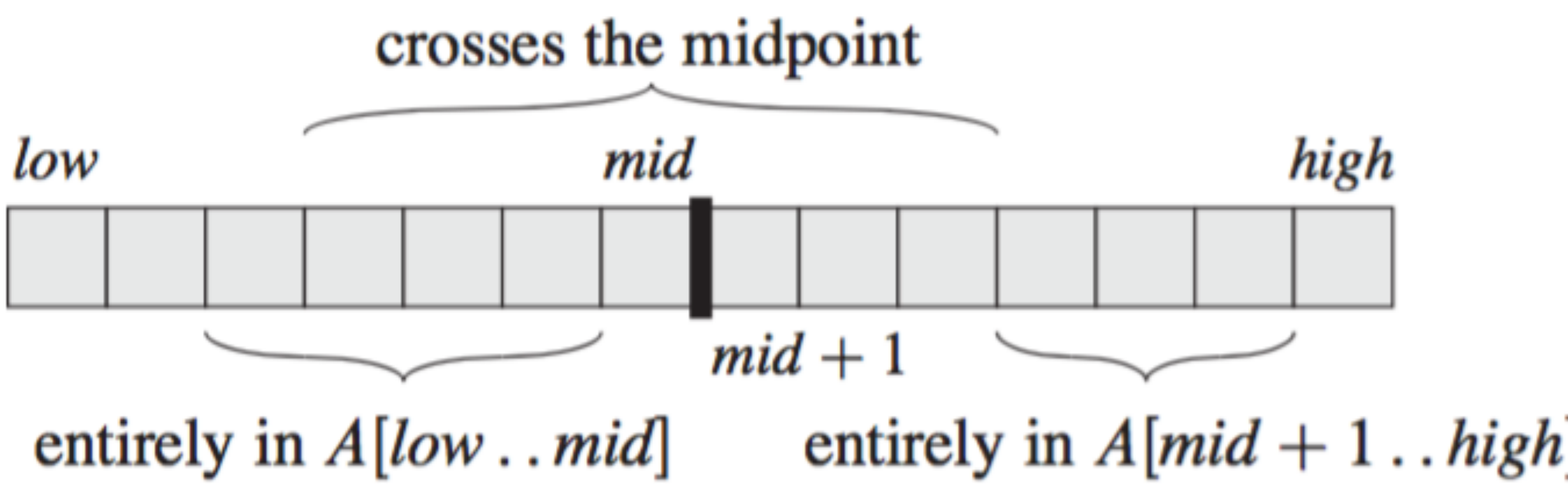
Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Maximum-subarray problem

Divide & Conquer:

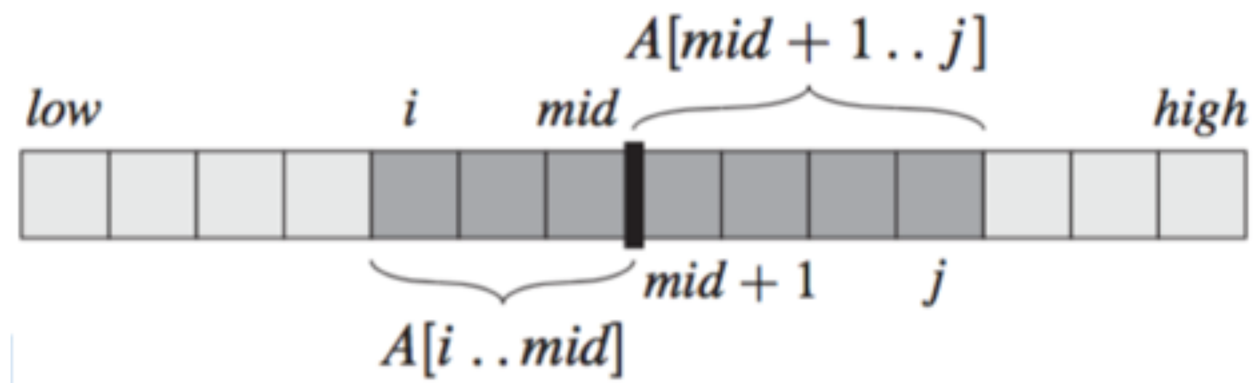
Idea: Divide A into 2 pieces —> maximum subarray is either

1. entirely in the subarray *A*[*low* … *mid*],
2. entirely in the subarray *A*[*mid+1* … *high*], or
3. crossing the midpoint



Maximum-subarray problem

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```



Runtime: $\Theta(n)$.

Maximum-subarray problem

FIND-MAXIMUM-SUBARRAY(*A*, *low*, *high*)

$\Theta(1)$

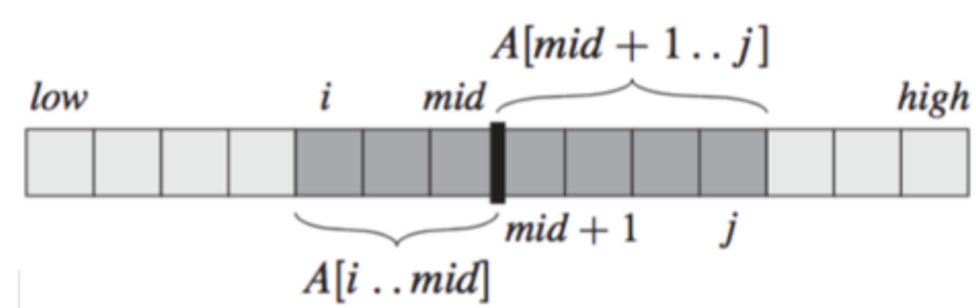
```
if high == low
    return (low, high, A[low])            // base case: only one element
```

$2T(n/2)$

```
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
```

$f(n)$

```
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum
        return (left-low, left-high, left-sum)
    elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-sum)
```



- Recurrence:
  - $T(n) = 2T(n/2) + \Theta(n)$
- Solution:
  - $a = 2$, $b = 2$, $n^{\log_b a} = n$, $f(n) = \Theta(n)$ —> **Case 2**.
  - Thus, $\boldsymbol{T(n) = \Theta(n \lg n)}$

Matrix multiplication

- Problem
  - Input: $A = [a_{ij}], B = [b_{ij}].$
  - Output: $C = [c_{ij}] = A \cdot B.$

$$\left.\begin{array}{l} A = [a_{ij}], B = [b_{ij}]. \\ C = [c_{ij}] = A \cdot B. \end{array}\right\} \quad i, j = 1, 2, \ldots, n.$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Matrix multiplication

Standard algorithm:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

for i = 1 **to** n
    do **for** j = 1 **to** n
        do $c_{ij} = 0$
            for k = 1 **to** n
                do $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$

$$\boldsymbol{T(n) = \Theta(n^3)}$$

Matrix multiplication

Divide & Conquer:

Idea: $n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C \quad = \quad A \quad \cdot \quad B$$

Combining subproblem solutions:

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\} \begin{array}{l} 8 \text{ mults of } (n/2)\times(n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2)\times(n/2) \text{ submatrices} \end{array}$$

Matrix multiplication

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
  n = A.rows
  let C be a new n × n matrix
  if n == 1
      c11 = a11 · b11
  else partition A, B, and C
      C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
            ⊕ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
      C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
            ⊕ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
      C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
            ⊕ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
      C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
            ⊕ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
  return C
```

$\Theta(1)$

$\Theta(1)$*

$\Theta(n^2)$

$8\,T(n/2)$

* with index calculations,
otherwise $\Theta(n^2)$ for copying entries

Matrix multiplication

Recurrence:

$$T(n) = 8T(n/2) + \Theta(n^2)$$

#subproblems

subproblem size

work dividing
and combining

$n^{\log_b a} = n^{\log_2 8} = n^3$

Case 1: $T(n) = \Theta(n^3)$.

No better than the ordinary algorithm.

Matrix multiplication

- Lessons Learned:
  - #additions goes away (constant factor)
  - #multiplications not —> recursive case (they make the tree “bushy”)
- What to do?
  - Try to reduce #multiplications
  - OK to have more additions

Matrix multiplication

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

Strassen‘s idea:

Multiply matrices with 7 multiplications and 18 additions.

$$P_1 = a \cdot (f - h)$$
$$P_2 = (a + b) \cdot h$$
$$P_3 = (c + d) \cdot e$$
$$P_4 = d \cdot (g - e)$$
$$P_5 = (a + d) \cdot (e + h)$$
$$P_6 = (b - d) \cdot (g + h)$$
$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$
$$s = P_1 + P_2$$
$$t = P_3 + P_4$$
$$u = P_5 + P_1 - P_3 - P_7$$

Matrix multiplication

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C \quad = \quad A \quad \cdot \quad B$$

Strassen's idea:

Multiply matrices with 7 multiplications and 18 additions.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad + (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad + bg + bh - dg - dh \\
&= ae + bg
\end{aligned}
$$

Matrix multiplication

Strassen‘s algorithm:

1. **Divide**:
   Partition A and B into $(n/2)\times(n/2)$ submatrices.
   Form terms to be multiplied using + and –.

2. **Conquer**:
   Perform 7 multiplications of $(n/2)\times(n/2)$ submatrices recursively.

3. **Combine**:
   Form C using + and – on $(n/2)\times(n/2)$ submatrices.

Matrix multiplication

Complexity of Strassen‘s algorithm:

Recurrence:

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$n^{\log_b a} = n^{\log_2 7} = n^{2.81}$

Case 1: $\boldsymbol{T(n) = \Theta(n^{\lg 7})}$.

2.81 may not seem much smaller than 3,
but difference is in the exponent,
the impact on running time is significant. Strassen’s algorithm beats the ordinary algorithm for n ≥ 32 or so.

Matrix multiplication

Best known algorithm:

Latest improvement in 2014 [1]:

$O(n^{2.3728639})$

Only of theoretical interest.

Most approaches that are faster than Strassen are not used in practice.

They are only faster for very large n.

One cannot get better than $O(n^2)$, cf. **Case 3**.

[1] Francois LeGall, Powers of Tensors and Fast Matrix Multiplication, 30 Jan 2014.

1.7 Conclusion

Conclusion

- Definitions
- First example of an algorithm (InsertionSort)
- Asymptotic analysis
- First powerful concept (Divide&Conquer)
- Solve recurrences for analysis