Divide and Conquer

Given a problem of size *n*:

1) ***Divide*** the problem into *k* sub-problems of size *n/k* each
2) ***Conquer*** by solving each sub-problem independently
3) ***Combine*** the *k* solutions to sub-problems into a solution to the original problem

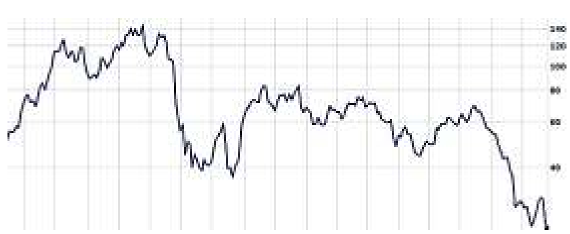
Time? $T(n) = k \cdot T(n/k) + d(n) + c(n)$

Examples. MergeSort, matrix multiplication, maximum contiguous sum, closest pair, etc.

1

Buying and Selling Stock

- Suppose you are given the daily prices of a certain stock over a given period
- With hindsight, what would have been the best time to buy and sell in order to maximize your profit?



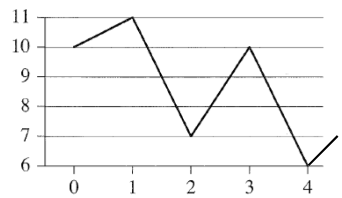
2

Proposed Algorithms

Algorithm 1. Buy at the lowest point and sell at the highest point after it.

Algorithm 2. Sell at the highest point and buy at the lowest point before it.

Algorithm 3. Choose the best of **1** or **2.**



Algorithm 4. Consider all pairs (i,j) of days where $j > i$ and choose the best pair.

3

A Transformation

- What if you work instead with the sequence of daily changes?

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

<u>Goal:</u> find a contiguous subarray whose value has a largest sum

-2	-3	4	-1	-2	1	5	-3
0	1	2	3	4	5	6	7

4 + (-1) + (-2) + 1 + 5 = 7

4

MaxSum Subarray

Algorithm 1. For each subarray $A[i..j]$ find the net change and keep the maximum.

```
MaxSubSum(A, n)
best ← 0
for i ← 1 to n do
    for j ← i  to n do
        sum ← 0
          for k ← i  to j do
            sum ← sum + A[k]
        if sum > best then
            best ← sum
return best
end
```

$T(n) \in \Theta(n^3)$

5

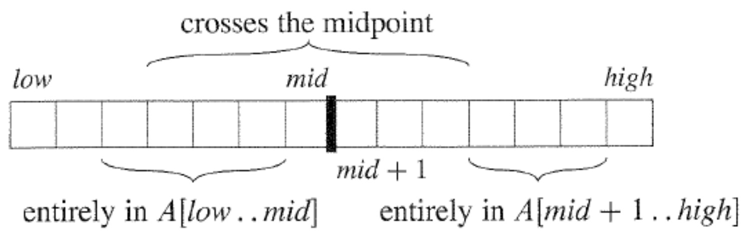
Algorithm 2

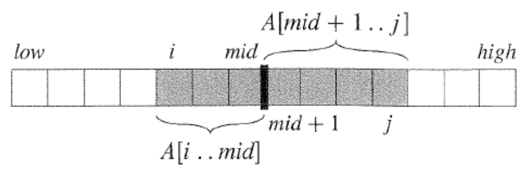
```
MaxSubSum(A, n)
best ← 0
for i ← 1 to n do
    sum ← 0
    for j ← i  to n do
        sum ← sum + A[j]
        if sum > best then
            best ← sum
return best
end
```

$T(n) \in \Theta(n^2)$

6

Algorithm 3: DAC





7

```
MaxSumSeq(A, low, high)
if low = high ▷ Base case
   then if A[low] > 0
           then return A[low]
           else  return 0
▷ Divide
mid ← ⌊(low + high)/2⌋
▷ Conquer
maxLeft ← MaxSumSeq(A, low, mid)
maxRight ← MaxSumSeq(A, mid +1, high)
▷ Combine
maxLeft2Center ← left2Center ← 0
for i ← mid downto low
     do left2Center ← left2Center + A[i]
        maxLeft2Center ← max(left2Center, maxLeft2Center)
maxRight2Center ← right2Center ← 0
for i ← mid +1 to high
     do right2Center ← right2Center + A[i]
        maxRight2Center ← max(right2Center, maxRight2Center)
return max(maxLeft, maxRight, maxLeft2Center + maxright2Center)
```

$T(n) = 2T(n/2) + n \in \Theta(n \log n)$

Exercise

- Design and analyze an *incremental* algorithm to compute the maximum profit you could get by buying and selling a particular stock at the right times

Matrix Multiplication

- Given two $n \times n$ matrices A and B find $C = A \times B$
- Recall: C is also $n \times n$ and

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

Example: $n = 2$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

– How many scalar operations were needed to compute $A \times B$?

A Brute Force Algorithm

Input: two $n \times n$ matrices A and B

SQUARE-MATRIX-MULTIPLY(A, B)

```
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

Input size: $N = 2n^2$

Time? $T(n) \in \Theta(n^3)$ or $T(N) \in \Theta(N\sqrt{N})$

11

A DAC Solution

- Partition each $n \times n$ matrix into four $n/2 \times n/2$ submatrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} , \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} , \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} , \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} . \end{aligned}$$

Recurrence? $T(n) = 8T(n/2) + n^2$

12

A DAC Solution…

```
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 as in equations (4.9)
    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
```

Time? $T(n) = 8T(n/2) + n^2 \in \Theta(n^3)$

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

$= n^2 + 8[(n/2)^2 + 8 \cdot T(n/4)] = n^2 + 2n^2 + 8^2 \cdot T(n/4)$

$= n^2 + 2n^2 + 8^2[(n/4)^2 + 8 \cdot T(n/8)]$

$= n^2 + 2n^2 + 2^2n^2 + 8^3 \cdot T(n/2^3)$

$= n^2 + 2n^2 + 2^2n^2 + \cdots + 2^{k-1}n^2 + 8^k \cdot T(n/2^k)$

$= n^2\left(1 + 2 + \cdots + 2^{k-1}\right) + 2^{3k} \cdot T(n/2^k)$

$= n^2\ (2^k - 1) + 2^{3k} \cdot T(n/2^k)$ Stop when $n = 2^k$

$= n^2\ (n-1) + n^3 \cdot T(1) \quad \in \Theta(n^3)$

A Different DAC Solution (Strassen '68)

$$
\begin{aligned}
S_1 &= B_{12} - B_{22}, & S_6 &= B_{11} + B_{22}, \\
S_2 &= A_{11} + A_{12}, & S_7 &= A_{12} - A_{22}, \\
S_3 &= A_{21} + A_{22}, & S_8 &= B_{21} + B_{22}, \\
S_4 &= B_{21} - B_{11}, & S_9 &= A_{11} - A_{21}, \\
S_5 &= A_{11} + A_{22}, & S_{10} &= B_{11} + B_{12}.
\end{aligned}
$$

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}, \\
P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}, \\
P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}, \\
P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}, \\
P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}, \\
P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}, \\
P_7 &= S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}.
\end{aligned}
$$

$C_{11} = P_5 + P_4 - P_2 + P_6$
$C_{12} = P_1 + P_2$
$C_{21} = P_3 + P_4$
$C_{22} = P_5 + P_1 - P_3 - P_7$

15

Strassen…

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

Recurrence? $T(n) = 7T(n/2) + n^2 \in \Theta(n^{\log_2 7})$

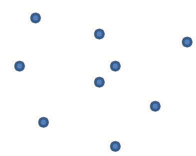
16

Closest Pair

- Given a set $P = \{p_1, p_2, \ldots, p_n\}$ of points on the plane find a and b such that

$$\text{dist}(p_a, p_b) \leq \text{dist}(p_i, p_j), \forall 1 \leq i \neq j \leq n$$

- Brute force takes $\Theta(n^2)$ time
- Can we do better?



17

A Divide and Conquer Solution

To compute $CP(P)$:

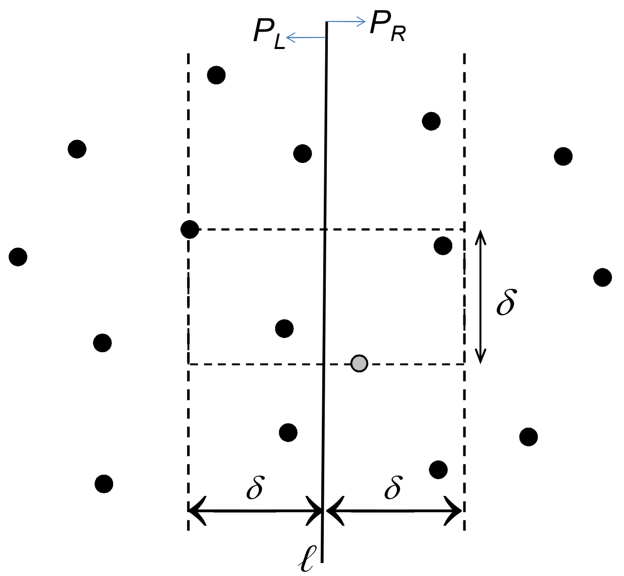
1. Sort P lexicographically, i.e., such that

$$x_i < x_{i+1} \vee x_i = x_{i+1} \wedge y_i \leq y_{i+1}$$

2. <u>Divide</u>: $P_L = \{p_1, \ldots, p_{n/2}\}$ and $P_R = \{p_{n/2+1}, \ldots, p_n\}$
3. <u>Conquer</u>: let $\delta_1 = CP(P_L)$ and $\delta_2 = CP(P_R)$
4. <u>Combine</u>: how? is $\delta = \min(\delta_1, \delta_2)$ the answer?

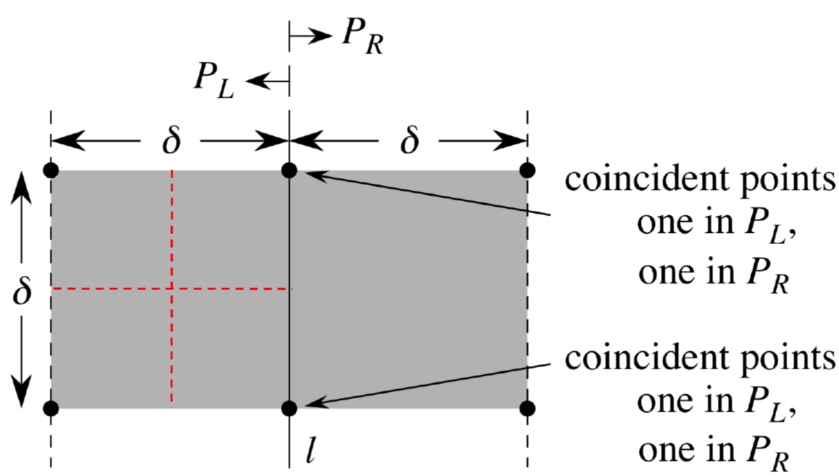
18

Closest Pair: Combine Step



19

How many points can you have in the box?



20

Exercise

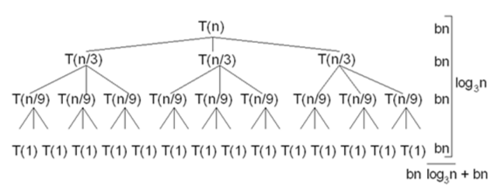
- Design and analyze an efficient incremental algorithm for the closest-pair problem
  - *Hint.* As in the DAC algorithm, transform-and-conquer is useful before you run your incremental solution

21

Solving Recurrences

- Iteration / Recursion Trees

$$
\begin{aligned}
T(n) &= n + 3T(n/3) \\
&= n + 3(n/3 + 3T(n/9)) \\
&= 2n + 9T(n/9) \\
&= 3n + 27T(n/27) \\
&\vdots \\
&= kn + 3^k T(n/3^k)
\end{aligned}
$$



- Substitution (Induction)
- Master Theorem

22

Substitution

- Prove $T(n)=2T(n/2)+\Theta(n) = \Theta(n \log n)$
  - Will show $T(n) \leq bn \log n$, $n \geq 2$ (upper bound)
  - Can assume
    - $T(n) \leq 2T(n/2) + cn$
    - $T(k) \leq bk \log k$, for $2 \leq k < n$

Proof:

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn \\ &\leq 2\left(b\frac{n}{2}\log\frac{n}{2}\right) + cn \\ &= bn(\log n - 1) + cn \\ &= bn\log n - (bn - cn) \\ &< bn\log n, \text{ if } b > c \end{aligned}$$

23

Lower bound?

- Show $T(n) \geq dn \log n$
- What can you assume?
  - $T(n) \geq 2T(n/2) + an$
  - $T(k) \geq dk \log k$, for $k < n$

Proof:

$$\begin{aligned} T(n) &\geq 2T(n/2) + an \\ &\geq 2\left(d\frac{n}{2}\log\frac{n}{2}\right) + an \\ &= dn(\log n - 1) + an \\ &= dn\log n + (an - dn) \\ &\geq dn\log n, \text{ if } a \geq d \end{aligned}$$

24

Master Theorem

Consider a DAC algorithm with running time $T(n) = a\,T(n/b) + f(n)$ where $a \geq 1$ and $b > 1$ are constants and $f(n)$ positive. Then:

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 $f(n) = \Theta(n^{\log_b a} \log^k n)$ with[1] $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ with $\varepsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$

Regularity Condition:

$$af(n/b) \leq cf(n) \text{ for some } c < 1 \text{ and large enough } n$$

[1] In the textbook, $k = 0$.

25

Practice Problems

Solve each problem using the Master Theorem or indicate why the theorem does not apply:

1. $T(n) = 8T(n/2) + n^2$
2. $T(n) = T(n/2) + 2^n$
3. $T(n) = 3T(n/2) + n$
4. $T(n) = 2^n T(n/2) + n^3$
5. $T(n) = 4T(n/2) + n$
6. $T(n) = 2T(n/2) + n/\log n$
7. $T(n) = \sqrt{2}\,T(n/2) + \log n$
8. $T(n) = 2T(n/2) + n\log n$

26

Exercise

- Describe a divide-and-conquer algorithm to compute the max of an array of n integers
- Write a recurrence for your algorithm and solve it using:
  1. Iteration
  2. Substitution (Induction)
  3. Master Method

27

Subset Sum

- Given a set A of positive integers and a target value t, find a subset $S \subseteq A$, whose elements add up to t

  Example: $A = \{1,3,4,5\}, t = 11$
- A DAC algorithm can be built around *two* smaller instances, by including or excluding the first element
- Only one instance is needed, but we don't know which one

```
subsetSumQ(X, n, from, t)
  if t = 0
      then return TRUE
  if t < 0 or from = n
      then return FALSE
  return subsetSumQ(X, n, from + 1, t)
    or subsetSumQ(X, n, from + 1, t − X[from])
```

Time?

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

$\in \Theta(2^n)$

28

Constructing the Subset

```
subsetSum(X, n, from, t)
if t = 0
    then return { }
if t < 0 or from = n
    then return None
Y ← subsetSum(X, from + 1, t)
if Y ≠ None
    then return Y
Y ← subsetSum(X, from + 1, t − X[from])
if Y ≠ None
    then return {X[from]} ∪ Y
```

29

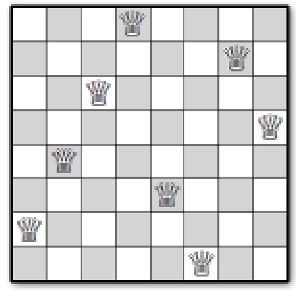
Algorithm Design Paradigms

- The problem with DAC subset sum is that it actually performs a recursive exhaustive search
- ***Backtracking*** constructs and evaluates the solution one component at a time using a ***state-space tree*** whose nodes are generated by DFS and reflect choices made for partial solutions. The root corresponds to start of search
- If a partial solution (a tree node) can be extended without violating the constraints, take the first remaining option for the next component; else, if there is no valid option for the next component, prune the node's subtree, and backtrack to replace the last component with next choice
- In the worst-case may still take exponential time

30

The n-Queens Problem

- Place n queens on an $n \times n$ board so that no two queens occupy the same row, column, or diagonal
- Clearly, need one queen per row
- Solution is an array $Q[1{:}n]$, with $Q[i] =$ the column for queen in row i

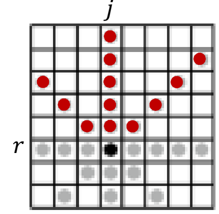


- In a *partial solution,* array Q contains positive values in the first t entries and zeros in the last $n - t$ entries
- Algorithm proceeds row by row, from top to bottom, and recursively enumerates all solutions that are consistent with given partial solution.

31

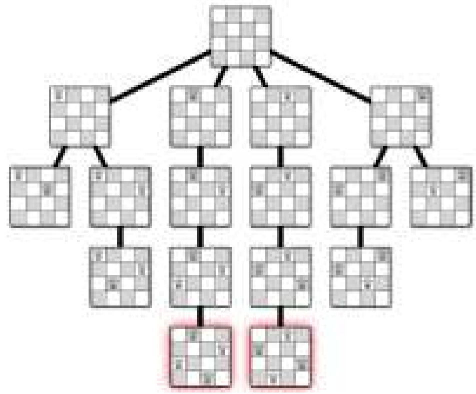
The Algorithm

- Parameter *r* denotes the first empty row, top to bottom
- Calling *nQueens*(*Q*,r) places a queen in row r, and *nQueens*(*Q*,1) recursively solves the problem




```
NQueens(Q, n, r)
if r = n + 1
   then Print(Q)
   else for j ← 1 to n
            do valid ← TRUE
               for i ← 1 to r − 1
                   do if Q[i] = j or Q[i] = j + r − i or Q[i] = j − r + i
                         then valid ← FALSE
                              break
               if valid
                  then Q[r] ← j
                       NQueens(Q, n, r + 1)
```

4-Queens State-Space Tree



- Why is this more efficient than brute-force?
- Can you think of other problems that can be solved by backtracking?

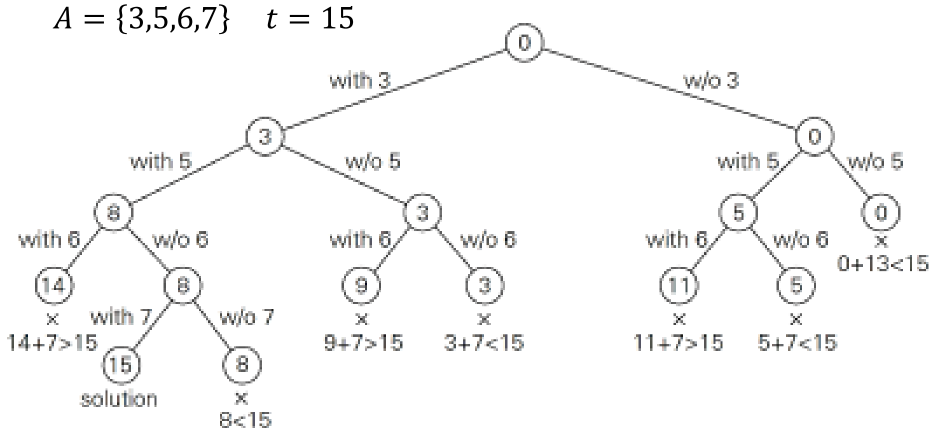
Subset Sum by Backtracking

- Given a set A of positive integers and a target value t, find a subset $S \subseteq A$, whose elements add up to t
- Process A in a ascending order: $a_1 < a_2 < \cdots < a_n$
- The state-space is a binary tree
  - Root represents the empty subset
  - Left (right) children correspond inclusion (exclusion) of the next element of A
  - Ancestors of a node at depth *i*, represent a subset of $a_1, \dots, a_i$
  - Record the sum s of the members of this subset at the node
    - If $s = t$, we found a solution
    - Prune if

$$s + a_i > t \quad \text{or} \quad s + \sum_{j=i+1}^{n} a_j < t$$

Example

$A = \{3,5,6,7\} \quad t = 15$



- How is this better than exhaustive search?

35

General Structure

- The output is a tuple $(x_1, x_2, \ldots, x_k)$ where $x_i \in S_i$
  - In n-Queens, $S_i = \{1, \ldots, n\}$
- Tuples may vary in size and need to satisfy additional constraints. Algorithm generates state-space tree with nodes $X = (x_1, \ldots, x_i)$ of partial solutions representing earlier decisions, and adds x_{i+1} consistent with constraints
- If x_{i+1} does not exist, backtrack and try next x_i, and so on

Backtrack(X, i):
    if $X(1{:}\,i)$ is a solution, report it
    else
        foreach $x \in S_{i+1}$ consistent with $X(1{:}\,i)$ **do**
            $X(i+1) \leftarrow x$
            Backtrack($X, i+1$)

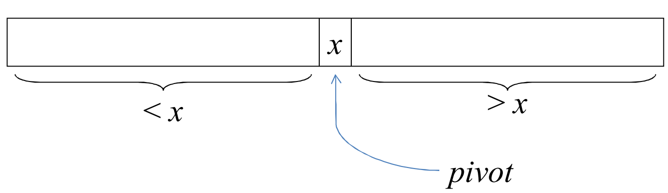
36

How to Shrink the State Space

- Exploit problem symmetry
  - n-Queens: Limit $S_1 = \{1, \dots, \lceil n/2 \rceil\}$, but keep $S_i = \{1, \dots, n\}$ for $i \neq 1$, as other solutions can be obtained by reflection
- Data presorting: rearrange input data
  - In subset sum, process values in ascending order

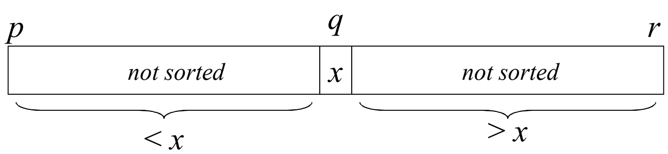
Quicksort

- Worst case is $\Theta(n^2)$
- Best case is $\Theta(n \log n)$
- Average case is $\Theta(n \log n)$
- Constant hidden in Θ-notation is small
- Sorts in place
- Divide-and-conquer
  - Based on linear time *partition* algorithm (a clever divide)



Quicksort…

1. *Divide*. Rearrange $A[p..r]$ into three parts $A[p..q-1]$, $A[q]$, and $A[q+1..r]$ such that each element of first part is < $A[q]$ and each element of third part is > $A[q]$
2. *Conquer*. Recursively sort the two unsorted parts
3. *Combine.* Not needed!



39

Example

								x	
14	27	3	75	41	4	13	65	23	54

Partition

14	3	4	13	23	27	75	41	65	54

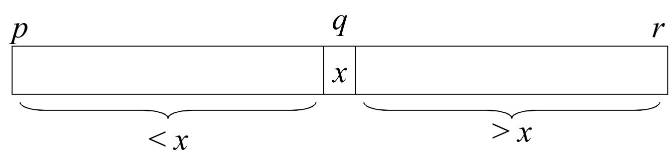
Sort recursively

3	4	13	14	23	27	41	54	65	75

40

Quicksort…

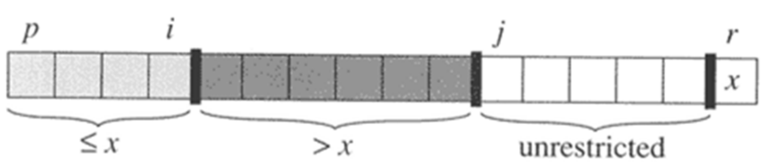
```
QUICKSORT(A, p, r)
if p < r
    then q ← PARTITION(A, p, r)
         QUICKSORT(A, p, q − 1)
         QUICKSORT(A, q + 1, r)
```



41

Partition

- Choose pivot $x = A[r]$
- Scan array once from left to right
- During partition, array consists of four parts
  - Portion already scanned $A[p..j-1]$ is partitioned into two parts $A[p..i]$ and $A[i+1..j-1]$ of elements smaller and bigger than x, respectively



42

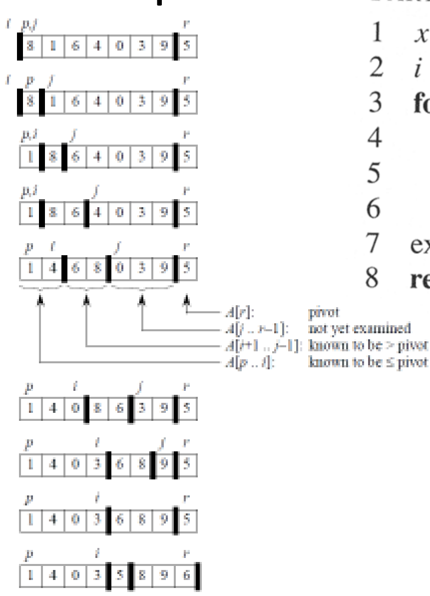
Partition

```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
      do if A[j] ≤ x
            then i ← i + 1
                  exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

- Runs in $\Theta(n)$ time (where $n = r - p + 1$)

43

Example

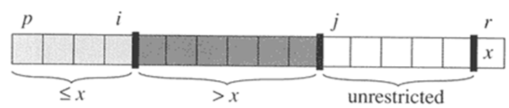



```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
      do if A[j] ≤ x
            then i ← i + 1
                  exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

44

Invariant

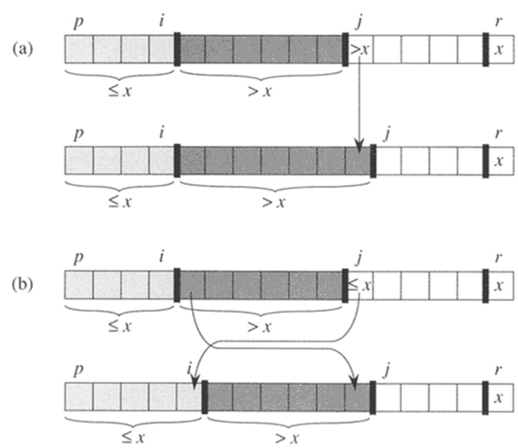
- At the beginning of each iteration (lines 3-6) the following conditions hold
    1. $A[r] = x$
    2. $A[k] \leq x$ for $p \leq k \leq i$
    3. $A[k] > x$ for $i < k < j$
- Verify!
    - Initialization
    - Maintenance
    - Termination



45

Maintaining the Invariant

- Two cases
    a) $A[j] > x$
    b) $A[j] \leq x$



46

Running Time: Worst Case

$T(n) = \max_{0 \le q < n} \{ T(q) + T(n - q - 1) + \Theta(n) \}$

where q = # elements in left part

- When does the worst case happen?
- Can you show $T(n) = \Theta(n^2)$ in the worst case?
  - Prove by substitution (induction)
    Guess $T(n) \le an^2$ to show $T(n) = O(n^2)$

47

Proof of Worst Case

Guess $T(n) \le an^2$
Base case : choose a so $T(1) \le a$
Inductive step :
$T(n) \le T(q) + T(n-q-1) + cn$
$T(n) \le aq^2 + a(n-q-1)^2 + cn$

Max of $f(q) = q^2 + (n-q-1)^2$ in $[0, n-1]$
occurs at $\{0, n-1\}$

$$T(n) \le a(n-1)^2 + cn = an^2 - \underbrace{(2an - a - cn)}_{\ge 0} \le an^2$$

48

How about the best case?

- Is it enough to check the case $q = n/2$?
  - This only shows upper bound for best case!

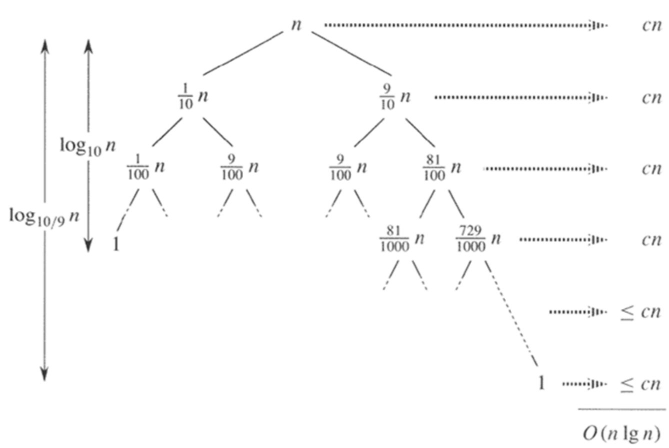
How about the average case?

- Will analyze in next section
- Get some insight by analyzing consistently unbalanced partitions

49

A Pretty Bad Case?

- Suppose you consistently get a 9-to-1 partitioning



50

```
RANDOMIZED-PARTITION(A, p, r)
i ← RANDOM(p, r)
exchange A[r] ↔ A[i]
return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
   then q ← RANDOMIZED-PARTITION(A, p, r)
        RANDOMIZED-QUICKSORT(A, p, q − 1)
        RANDOMIZED-QUICKSORT(A, q + 1, r)
```

51