

- Algorithm: Sequence of (computational) steps to solve a (computational) problem
- Analyze what?
 - Performance
 - exact/approximate solution
 - Resources
 - time
 - space
 - memory transfers

Analysis Result

- Alg is correct
- Running time is $O(\)$
 - not in secs
 - Theoretical
 - good for any computer
 - scalability
 - Worst case

Analysis Result

- Correctness
- Worst-case asymptotic running time
 - $O(\)$
- Design result
 - “analyzable”
 - correct
 - fast



Analysis of algorithms

The theoretical study of computer-program performance and resource usage.

What's more important than performance?

- modularity
- correctness
- maintainability
- functionality
- robustness
- user-friendliness
- programmer time
- simplicity
- extensibility
- reliability

September 7, 2005 Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson
Introduction to Algorithms

L1.3

Pseudocode

Do!
Stop!
Go!
Subtract!
Listen to me!

```
public void point(Graphics g) {
    super.paint(g);
    g.setColor(Color.BLUE);
    g.setFont(font);
    g.drawString(resBundle.getString("mess"),
    ...
}

public class newActionClass extends Abstract
```

Algorithm for Robust Shortest Path
Let C be the maximum second stage cost of some fixed connected optimal solution.
 $T = \{t_1, t_2, \dots, t_n\}$ are the terminals, $r \leftarrow \text{root}$, $\alpha \leftarrow 1.775$, $V^* \leftarrow \emptyset$.

1. $V^* := \{t_i \mid \text{dist}(t_i, r) > \frac{C}{\alpha}\}$
2. $B := \{B_i = B(t_i, \frac{C}{2\alpha}) \mid t_i \in V^*\}$, where $B(t, d)$ is a ball of radius d around t with respect to cost c . Choose a maximal set B_S of non-intersecting balls from B in order of non-decreasing radii.
3. Guess $R^* := \{t_i \mid B_i \in B_S\}$.
4. First stage solution: $E_S \leftarrow$ The Steiner tree on terminals $R^* \cup \{r\}$ output by the best approximation algorithm available.
5. Second stage solution for scenario i : $E_i \leftarrow$ Shortest path from t_i to the closest node in the tree E_S .

Algorithm FINDINTERSECTIONS(S)
Input: A set S of line segments in the plane.
Output: The set of intersection points among the segments in S , with for each intersection point the segments that contain it.

1. Initialize an empty queue Q . Next, insert the segment endpoints into Q ; when an upper endpoint is inserted, the corresponding segment should be moved with it.
2. Initialize an empty stack structure \mathcal{I} .
3. while Q is not empty
4. do Determine the next event point p in Q and delete it.
5. HANDLEEVENTPOINT(p)

INSERTION-SORT(A)
for $j \leftarrow 2$ to n
 do $key \leftarrow A[j]$
 $i \leftarrow j - 1$
 while $i > 0$ and $A[i] > key$
 do $A[i + 1] \leftarrow A[i]$
 $i \leftarrow i - 1$
 $A[i + 1] \leftarrow key$

INSERTION-SORT(A, n)
for $j \leftarrow 2$ to n
 $key \leftarrow A[j]$
 // Insert $A[j]$ into the sorted sequence $A[1 \dots j - 1]$.
 $i \leftarrow j - 1$
 while $i > 0$ and $A[i] > key$
 $A[i + 1] \leftarrow A[i]$
 $i \leftarrow i - 1$
 $A[i + 1] \leftarrow key$

Pseudocode

- $A[1 \dots n]$ or $A[0 \dots n]$
- $A[3]$ or $A(3)$
- $A[5:17]$ or $A[5 \dots 17]$
- var.attr or $\text{var} \rightarrow \text{attr}$
- begin, do, while, until, repeat, enddo, endif, case, goto, k++
- Refer to other procedures

- Capture ALL essentials
 - INPUT, OUTPUT
 - what to do
 - helps analyzing

Pseudocode alone is not enough
Plain English is a must

Matrix multiplication

$C = \text{Matrix-Multiply}(A, B)$

$C = 0$

for $i = 1..n$

for $j = 1..n$

for $k = 1..n$

$C_{ij} = C_{ij} + a_{ik} b_{kj}$

- Design result
 - “analyzable”
 - correct
 - fast

Divide and Conquer

- Divide
 - into subproblems
- Conquer
 - solve subproblems
- Combine
 - subproblems solutions

Matrix multiplication

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

$$\bullet T(1) = O(1) \text{ -- ignore for asymptotics}$$

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

Recurrence

- $T(n)$ = function of n , $T(n')$, $T(n'')$, ...
 - $n', n'', \dots < n$
- Iteration
- Recursion tree
- Substitution (guess, prove by induction)
- Master method

$$T(n) = 8 T(n/2) + O(n^2)$$

Last time

- Prerequisites
 - Big-O
 - small-o, big-Omega, Theta...
 - Logs, geometric progressions, etc. (no trigonometry)
 - Matrix multiplication
 - definition
 - properties
- Alg analysis
 - running time $O(n^c \log n + m^c)$
 - correct: $O(n^3)$
 - incorrect: $O(2^{3 \log n})$
- D&C -> recurrence -> master method and others

Master method

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

$$- f(n) = O(n^{\log_b a - e}) \Rightarrow T(n) = \Theta(n^{\log_b a})$$

$$- f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \log n)$$

$$- f(n) = \Omega(n^{\log_b a + e}), af(n/b) < cf(n) \Rightarrow T(n) = \Theta(f(n))$$

$a \geq 1, b > 1, e > 0, c < 1$

Why $n^{\log_b a}$?

a subproblems
size n/b

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

D&C

- Recurrence
 - master method

$$\begin{aligned} T(n) &= aT(n/b) + f(n) \\ -f(n) &= O(n^{\log_b a - e}) \Rightarrow T(n) = \Theta(n^{\log_b a}) \\ -f(n) &= \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \log n) \\ -f(n) &= \Omega(n^{\log_b a + e}) \Rightarrow T(n) = \Theta(f(n)) \end{aligned}$$

- Matrix multiplication
- Merge sort
- Search in sorted array
- Power
- Fibonacci numbers
 - top-down vs bottom-up

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

Strassen's algorithm

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

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

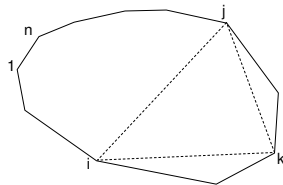
September 14, 2005 Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson 1.2.37

Dynamic Programming

- Structure of opt
 - optimal substructure
- Recursion
- Solve

Min-Weight (Convex) Polygon Triangulation

- Weight of triangulation
 - sum of weights of triangles
 - weight of Δabc is $w(abc)$
 - w is arbitrary function
 - e.g., perimeter of abc
- Find (the weight of) triangulation with min weight
- For $i < j$ define
 - $T(i, j)$ = min-weight triangulation of $i, i+1, \dots, j, i$
- Then $T(i, j) = 0$ if $j = i+1$,
 $T(i, j) = \min_{i < k < j} \{T(i, k) + T(k, j) + w(ikj)\}$ o.w.
- Compute $T(i, j)$ in order of increasing $j-i$
- Solution = $T(1, n)$



Rod cutting

- $p(i)$ – price of length- i (sub)rod
 - integer i
- Decompose to max revenue
- $r(n) = \max_{i=1 \dots n} \{p(i) + r(n-i)\}$

Procedure (bad choice)	Array (good choice)
$ROD(n)$ $\bullet ROD(0)=0$ $\bullet ROD(n)=\max_{i=1 \dots n} \{p(i) + ROD(n-i)\}$	$ROD(n)$ $\bullet Rod(0)=0$ $\bullet \text{For } j = 1 \dots n$ $\quad \bullet Rod(j)=\max_{i=1 \dots j} \{p(i) + Rod(j-i)\}$ $\bullet \text{Return } Rod(n)$
Déjà vu?	

All Pairs Shortest Paths

- No neg cycles (but allow neg weights)
 - w.l.o.g., complete graph
- Structure of opt
- Recursion
- Solve

Algorithmic “techniques”

- D&C
- DP
- No strict definition
- Learn by example
- Practice

DP

DP/OC in MDP

- Structure of opt
- Recursion
- Solve

- Reconstruct the solution
-- backtrack

Dynamic-programming hallmark #1

Optimal substructure
An optimal solution to a problem (instance) contains optimal solutions to subproblems.

Dynamic-programming hallmark #2

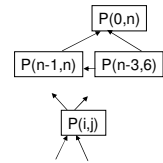
Overlapping subproblems
A recursive solution contains a "small" number of distinct subproblems repeated many times.

In mathematics and computer science, **dynamic programming** is a method of solving complex problems by breaking them down into simpler steps. It is applicable to problems that exhibit the properties of **overlapping subproblems** which are only slightly smaller¹ and **optimal substructure** (described below). When applicable, the method takes much less time than naive methods.

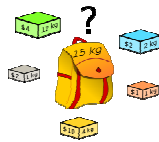
¹It may seem strange that dynamic programming relies on subproblems being both independent and overlapping. Although these requirements may sound contradictory, they describe two different notions, rather than two points on the same axis. Two subproblems of the same problem are independent if they do not share resources. Two subproblems are overlapping if they are really the same subproblem that occurs as a subproblem of different problems.

DP in a nutshell

- Subproblems
 - better polynomial number of!
 - e.g., indexed by $O(1)$ indices
- Subproblems graph
 - better be acyclic!
 - but not necessarily a tree
- Recursively solve subproblems
 - better have all predecessor subproblems solved before solving a subproblem!
 - e.g., topological sort



Knapsack



- n items
 - weight $w(i)$
 - value $v(i)$
- Capacity W
- Maximize value
- Tabulate $O(nW)$ subproblems
 - $\text{val}(i,0) = \text{val}(0,w) = 0$
 - $\text{val}(i,w) = \text{val}(i-1,w)$ if $w(i) > w$
 - $\text{val}(i,w) = \max\{\text{val}(i-1,w), \text{val}(i-1,w-w(i))+v(i)\}$ o.w.

Traveling salesman problem

- Given $G=(V,E,w)$
- Find (the length of) shortest cycle visiting each vertex exactly once
 - $O(n!)$ time -- straightforward
- DP: S subset of V
 - $1 \in S$
 - $j \in S$
- $T(S,j)$ – shortest tour from 1 to j visiting each vertex in S exactly once
 - $T(S,j) = |1j|$ if $|S|=2$
 - $T(S,j) = \min_{l \in S \setminus \{j\}} \{T(S \setminus \{j\}, l) + |lj|\}$
- $O(2^n n^2)$

NP-completeness

Running time

- Poly(input size) of constant degree
 - $O(n^c), O(n^c + m^c)$
- TSP is not
 - no polynomial alg known
 - no exponential lower bound
- More problems like that
 - NP-complete

Decision versions

- Given
- Does there exist ...
 - not optimization
 - more convenient

NP-complete

- “Def 1”: Π is NP-complete if
 - given instance I of Π
 - “no polytime alg distinguishes if I is a Yes or a No”
- “Def 2”: Π is NP-complete (more formally) if
 - solution verifiable in poly time
 - some known NP-complete problem Π' can be reduced to Π
- Reduction
 - a poly-time transformation
 - from any instance I' of Π' to an instance I of Π
 - I is Yes iff I' is Yes
- Π is “no easier” than Π'
- Proof of definitions equivalence: By contradiction

Thousands of problems and growing

- If **any one** NP-complete problem can be solved
 - in polytime
- Then **all the others** too!



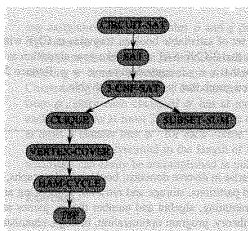
SAT: NP-complete problem “0”

- Variables: $X = \{x_1, \dots, x_n\}$
 - Literal: x_i or $\neg x_i$
- Clause: set of literals

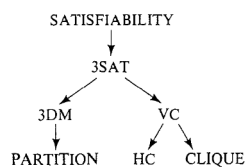
SAT

- Given
 - variables $X = \{x_1, \dots, x_n\}$
 - clauses c_1, \dots, c_m
- Does there exist truth assignment $T: X \rightarrow \{0,1\}$ s.t. every clause has a true literal?

Basic NP-complete problems



CLRS



Garey, Michael R.; Johnson, David S. (1979),
Computers and Intractability:
A Guide to the Theory of NP-Completeness

3-SAT

SAT

- Given
 - variables $X = \{x_1, \dots, x_n\}$
 - clauses c_1, \dots, c_m
- Does there exist truth assignment $T: X \rightarrow \{0,1\}$ s.t. every clause has a true literal?

3-SAT

- Given
 - variables $X = \{x_1, \dots, x_n\}$
 - clauses c_1, \dots, c_m
 - $|c_i| = 3$
- Does there exist truth assignment $T: X \rightarrow \{0,1\}$ s.t. every clause has a true literal?

Reduction: SAT -> 3-SAT

- From instance I of SAT
 - variables $X = \{x_1 \dots x_n\}$, clauses $c_1 \dots c_m$
- Form (in **poly(n,m) time!**) instance I' of 3-SAT
 - variables $X' = \{x'_1 \dots x'_n\}$, clauses $c'_1 \dots c'_m$
 - $|c'_j| = 3$
- Exists $T: X \rightarrow \{0,1\}$ s.t. each of c_j has a true literal
- Iff exists $T': X' \rightarrow \{0,1\}$ s.t. each of c'_j has a true literal

X', C'

- $X' = X \cup X_1 \cup \dots \cup X_m$; $C' = C_1 \cup \dots \cup C_m$
- $c_j = \{a\} \Rightarrow X_j = \{q_j, r_j\}$,
 $C_j = \{\{a, q_j, r_j\}, \{a, q_j, \neg r_j\}, \{a, \neg q_j, r_j\}, \{a, \neg q_j, \neg r_j\}\}$
- $c_j = \{a, b\} \Rightarrow X_j = \{s_j\}$, $C_j = \{\{a, b, s_j\}, \{a, b, \neg s_j\}\}$
- $c_j = \{a, b, d\}$
- $c_j = \{a_1, a_2, \dots, a_k\} \Rightarrow X_j = \{t_1, t_2, \dots, t_{k-3}\}$,
 $C_j = \{\{a_1, a_2, t_1\}, \{\neg t_1, a_2, t_2\}, \{\neg t_2, a_3, t_3\}, \dots, \{\neg t_{k-3}, a_{k-1}, a_k\}\}$
 – min I s.t. $a_i = 1$

General vs Restricted

- A polytime alg for general problem \Rightarrow
- A polytime alg for a restricted one
- Restricted problem is NP-complete \Rightarrow
- General problem is NP-complete
- Reductions can show
 - NP-completeness
 - polynomial-time solvability (a “design” technique)
 - nothing

2-SAT

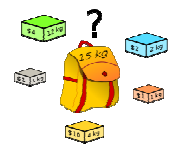
- Given
 - variables $X = \{x_1 \dots x_n\}$
 - clauses $c_1 \dots c_m$
 - $|c_i| = 2$
- Does there exist truth assignment $T: X \rightarrow \{0,1\}$ s.t. every clause has a true literal?
- L/R labeling
 - more in Wiki

Subset Sum

	x_1	x_2	x_3	C_1	C_2	C_3	C_4
$x_1 =$	1	0	0	1	0	0	1
$x_2 =$	1	0	0	0	1	1	0
$x_3 =$	0	1	0	0	0	0	1
$x_4 =$	0	1	0	1	1	1	0
$x_5 =$	0	0	1	1	1	0	0
$x_6 =$	0	0	0	1	0	0	0
$x_7 =$	0	0	0	2	0	0	0
$x_8 =$	0	0	0	0	1	0	0
$x_9 =$	0	0	0	0	2	0	0
$x_{10} =$	0	0	0	0	0	1	0
$x_{11} =$	0	0	0	0	0	2	0
$x_{12} =$	0	0	0	0	0	0	1
$x_{13} =$	0	0	0	0	0	0	2
$T =$	1	1	1	4	4	4	4

Figure 34.19 The reduction of 3-CNF-SAT to SUBSET-SUM. The formula in 3-CNF is $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$, where $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$, $C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, $C_3 = (\neg x_1 \vee \neg x_2 \vee x_3)$, and $C_4 = (x_1 \vee x_2 \vee x_3)$. A satisfying assignment of ϕ is $(x_1 = 0, x_2 = 0, x_3 = 1)$. The set S consists of the 10 numbers consisting of the binary 10 numbers 0000000001, 0000000010, 0000000011, 0000000100, 0000000101, 0000000110, 0000000111, 0000001000, 0000001001, 0000001010.

Knapsack



- NP-complete
- $O(nW)$
 - NOT polynomial
 - exponential
 - pseudopolynomial
 - poly in max integer
- Weakly NP-complete

Amortized analysis

Stack

- Push(S,x)
- Pop(S)
- Multipop(S,k)
 - while S is not empty and $k \neq 0$
 - pop(S)
 - $k = k - 1$

Empty stack, n Push, Pop, Multipop

- n $O(n)$?
- Pop (even inside Multipop) \rightarrow Push
- Plates and \$
 - Push \$2
 - Pop, Multipop \$0
 - at any time, total_paid \geq total_cost

Potential method

- Most powerful
- Potential = $|S|$
 - amortized_cost(i) = cost(i) + $|S_i| - |S_{i-1}|$
 - total_amortized_cost = total_cost + P(S_i)

operation	actual cost	$\Delta\Phi$	amortized cost
PUSH	1	$(s+1) - s = 1$	$1 + 1 = 2$
		where $s = \#$ of objects initially	
POP	1	$(s-1) - s = -1$	$1 - 1 = 0$
MULTIPOP	$k' = \min(k, s)$	$(s - k') - s = -k'$	$k' - k' = 0$

Cost

	Worst-case	Amortized
Push	$O(1)$	$O(1)$
Pop	$O(1)$	$O(1)$
Multipop	$O(n)$	$O(1)$

Graham's scan

Algorithm CONVEXHULL(P)

Input. A set P of points in the plane.

Output. A list containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

1. Sort the points by x -coordinate, resulting in a sequence p_1, \dots, p_n .
2. Put the points p_1 and p_2 in a list $\mathcal{L}_{\text{upper}}$, with p_1 as the first point.
3. **for** $i \leftarrow 3$ **to** n
4. **do** Append p_i to $\mathcal{L}_{\text{upper}}$.
5. **while** $\mathcal{L}_{\text{upper}}$ contains more than two points **and** the last three point in $\mathcal{L}_{\text{upper}}$ do not make a right turn
6. **do** Delete the middle of the last three points from $\mathcal{L}_{\text{upper}}$.

Probability

(Discrete) probability space

- Sample space S
- Events F : subsets of S
- $P: F \rightarrow [0,1]$
 - $P(S) = 1$
 - $P(A \cup B) = P(A) + P(B)$
 - A disjoint from B

(Discrete) random variable

- $X: S \rightarrow N$
- Event X in B subset of N
 - preimage of B
 - $X^{-1}(B)$
 - $\{s \text{ in } S : X(s) \text{ in } B\}$

New sample space!

- $N \cup \emptyset$
 - $P(B) = P(X \text{ in } B) = P(X^{-1}(B)) = P(\{s \text{ in } S : X(s) \text{ in } B\})$
- Probability theory
 - in terms of r.v.'s
 - X
 - **forget “original” sample spaces !**
 - S
 - but not forget forever

Distribution

- Probability mass function pdf
 - $P(B) = P(X \text{ in } B) = P(X^{-1}(B)) = P(\{s \text{ in } S : X(s) \text{ in } B\})$
 - $P(x) = P(X=x) = P(X^{-1}(x)) = P(\{s \text{ in } S : X(s) = x\})$
 - probability density function pdf
- (Cumulative) distribution function cdf
 - $P(X \leq x) = P((-\infty, x]) = \sum_{i \leq x} P(i)$

Joint distribution

- r.v.'s $X, Y: S \rightarrow R$
- Joint pmf
 - $f(x, y) = P(X = x, Y = y)$
- Marginals
 - $P_x(x) = \sum_y P(X=x, Y=y)$
 - $P_y(y) = \sum_x P(X=x, Y=y)$

Independence

- $P(X=x, Y=y) = P(X=x)P(Y=y)$ – r.v.
- $P(AB) = P(A)P(B)$ – evnets
- 2 coins
 - $X=0, Y=1$
 - $X=0, Y \neq X$
 - $X=0, Y=X$
- 2 dice
 - $X=3, X+Y=7$
 - $X=3, X+Y=6$

Expectation

- $E[X] = \sum_x x \Pr(X=x)$
 - $E[aX+bY] = aE[X] + bE[Y]$
 - $E[XY] = E[X]E[Y]$ for independent X, Y

Conditional probability

- $P(X=x | Y=y) = P(X=x, Y=y) / P(Y=y)$ – r.v.
- $P(A|B) = P(AB) / P(B)$ – events
- Restricted sample space
 - 2 coins, one H
 - 2 dice
 - $X=3, X+Y=7$
 - $X=3, X+Y=6$

Conditional expectation

- $E[X | Y]$
 - function of Y
 - $f(y) = E[X | Y=y]$
- $E[X] = E[E[X | Y]]$

Indicator r.v.

- (back to sample space)
- $I_A(s) = 1$ if s in A , $= 0$ o.w.
 - Preimages of 1 and 0?
- $I_A = I(A)$
- $I_{X=x} = I(X=x)$
 - Examples: $I(X>2), I(X>Y), \dots$
- $E[I(A)] = P(A)$

Examples

	Bernoulli	Geometric	Binomial
pmf			
E			

- 2 dice, $P(2 \text{ before } 5)$?
- 1 is before 2 in (uniformly) random permutation
 - counting permutations
- $P(i\text{-permutation of } n \text{ numbers})$

Random permutation

- Permute $A[1...n]$ uniformly at random
 - each permutation equally likely
- For $i = 1$ to n
 - swap $A[i]$ with $A(\text{rand}(i,n))$
- Before iteration $i+1$
 - $A[1...i]$ equally likely to be any i -permutation

Average-case analysis

Selection sort

For $j = 1...n$
 smallest = IndexOfMin($A[j...n]$)
 swap $A[j]$ with $A[\text{smallest}]$

SELECTION-SORT(A)

```
 $n \leftarrow \text{length}[A]$ 
for  $j \leftarrow 1$  to  $n - 1$ 
  do  $\text{smallest} \leftarrow j$ 
    for  $i \leftarrow j + 1$  to  $n$ 
      do if  $A[i] < A[\text{smallest}]$ 
        then  $\text{smallest} \leftarrow i$ 
    exchange  $A[j] \leftrightarrow A[\text{smallest}]$ 
```

SELECTION-SORT(A)

```
 $n = A.\text{length}$ 
for  $j = 1$  to  $n - 1$ 
   $\text{smallest} = j$ 
  for  $i = j + 1$  to  $n$ 
    if  $A[i] < A[\text{smallest}]$ 
       $\text{smallest} = i$ 
  exchange  $A[j]$  with  $A[\text{smallest}]$ 
```

Insertion sort

INSERTION-SORT(A)

```
for  $j \leftarrow 2$  to  $n$ 
  do  $\text{key} \leftarrow A[j]$ 
     $\triangleright$  Insert  $A[j]$  into the sorted sequence  $A[1 \dots j - 1]$ .
     $i \leftarrow j - 1$ 
    while  $i > 0$  and  $A[i] > \text{key}$ 
      do  $A[i + 1] \leftarrow A[i]$ 
       $i \leftarrow i - 1$ 
     $A[i + 1] \leftarrow \text{key}$ 
```

INSERTION-SORT(A, n)

```
for  $j = 2$  to  $n$ 
   $\text{key} = A[j]$ 
  // Insert  $A[j]$  into the sorted sequence  $A[1 \dots j - 1]$ .
   $i = j - 1$ 
  while  $i > 0$  and  $A[i] > \text{key}$ 
     $A[i + 1] = A[i]$ 
     $i = i - 1$ 
   $A[i + 1] = \text{key}$ 
```

Quicksort

- $A[1...n]$
- $q = \text{rnd}(1,n)$
- $A(q) = \text{pivot}$
- For $i = 1...n$
 - if $A(i) < \text{pivot}$, move $A(i)$ left of pivot
 - o.w., move $A(i)$ right of pivot
 - sort arrays left and right of pivot recursively

Quicksort run time

- Run time = $O(\# \text{ of comparisons})$
 - + $O(n)$, of course
- Observation: a comparison is only against the pivot
- i and j are compared \Leftrightarrow
 - i or j is a pivot
 - because o.w. no comparison against i or j
 - none of $[i+1, i+2, \dots, j-1]$ has been a pivot before
 - because o.w. i, j are in different subproblems

P(i,j are compared)

- i and j are compared \Leftrightarrow
 - i is a pivot or j is a pivot
 - none of $[i+1, i+2, \dots, j-1]$ has been a pivot before
- Same as:
 - one of i, j is the *first* from $[i, \dots, j]$ chosen as pivot
- $P(i, j \text{ compared}) =$
 $= P(\text{one of } i, j \text{ is the first from } [i, \dots, j] \text{ chosen as pivot}) =$
 $= 2/(j-i+1)$
 - assume $j > i$

Quicksort expected run time

- $X(i, j) = \text{Ind}(i \text{ and } j \text{ are compared})$
- $E[\text{runtime}] = E(\sum_{i,j} X(i, j)) = \sum_i \sum_{j>i} 2/(j-i+1) =$
 $= \sum_i \sum_k 2/k = O(\sum_i H(k)) = O(\sum_i H(n)) = O(n \log n)$
 - $H(k) = 1 + 1/2 + 1/3 + \dots + 1/k = O(\log k)$
 - k th harmonic number

Algorithm MINIDISC(P)

Input. A set P of n points in the plane.

Output. The smallest enclosing disc for P .

1. Compute a random permutation p_1, \dots, p_n of P .
2. Let D_2 be the smallest enclosing disc for $\{p_1, p_2\}$.
3. **for** $i \leftarrow 3$ **to** n
4. **do if** $p_i \in D_{i-1}$
5. **then** $D_i \leftarrow D_{i-1}$
6. **else** $D_i \leftarrow \text{MINIDISCWITHPOINT}(\{p_1, \dots, p_{i-1}\}, p_i)$
7. **return** D_n

MINIDISCWITHPOINT(P, q)

Input. A set P of n points in the plane, and a point q such that there exists an enclosing disc for P with q on its boundary.

Output. The smallest enclosing disc for P with q on its boundary.

1. Compute a random permutation p_1, \dots, p_n of P .
2. Let D_1 be the smallest disc with q and p_1 on its boundary.
3. **for** $j \leftarrow 2$ **to** n
4. **do if** $p_j \in D_{j-1}$
5. **then** $D_j \leftarrow D_{j-1}$
6. **else** $D_j \leftarrow \text{MINIDISCWITH2POINTS}(\{p_1, \dots, p_{j-1}\}, p_j, q)$
7. **return** D_n

MINIDISCWITH2POINTS(P, q_1, q_2)

Input. A set P of n points in the plane, and two points q_1 and q_2 such that there exists an enclosing disc for P with q_1 and q_2 on its boundary.

Output. The smallest enclosing disc for P with q_1 and q_2 on its boundary.

1. Let D_k be the smallest disc with q_1 and q_2 on its boundary.
2. **for** $k \leftarrow 1$ **to** n
3. **do if** $p_k \in D_{k-1}$
4. **then** $D_k \leftarrow D_{k-1}$
5. **else** $D_k \leftarrow$ the disc with q_1, q_2 , and p_k on its boundary
6. **return** D_n

Max3SAT

- 3SAT
- Maximize # of clauses with a true literal
- No clause has x and $\neg x$

a-Approximation algorithm

- poly-time
- $APX \leq a \cdot OPT$
 - minimization
- $APX \geq a \cdot OPT$
 - maximization

TSP (Min Covering Cycle)

- Given:
 - $G=(V, E, w)$
- Find:
 - min-length cycle visiting each vertex at least once
 - TSP (standard, formal): exactly once
- Metric completion \rightarrow complete graph
 - 1-to-1 correspondence between walks
 - w is metric
 - triangle inequality
- NP-complete
- 2- apx by doubling the MST

Matchings

Matching

- $G=(V,E)$
- M subset of E
- Any vertex v
 - incident to at most one edge in M
- v is matched

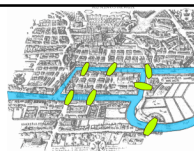
Perfect Matching

- Given: $G=(V,E)$
 - $|V|=2k$
- Is there matching M s.t.
 - each vertex is matched
- $O(n^3)$

Min-Weight Perfect Matching

- Given: $G=(V,E,w)$
 - $|V|=2k$
 - complete
 - o.w. add ∞ -weight edges
- Find: matching M of min weight s.t.
 - each vertex is matched
- $O(n^3)$

Euler cycle



- Given $G=(V,E)$
- Is there a cycle traversing each edge 1! time ?
- Chinese postman
 - covering version of Euler cycle
- Find a shortest cycle traversing each edge

Christofides algorithm

- TSP
- $MST < TSP$
- $MWPM(\text{odd-degree nodes}(MST)) < TSP/2$
- $MST + MWPM = \text{Eulerian graph}$
- $\text{cost} < 3/2 \text{ TSP}$

Covering vs. Packing

- Min vs. Max
- Vertices vs. Edges
 - VC, DS, IS, EDS, EC, M
- Matching is packing
 - max weight (cardinality)
 - unlike min-weight perfect matching

$\text{MaxM} < \text{MaxWM}$

V

$\text{PM} < \text{MinWPM}$

Maximum-Weight Matching

- Given $G=(V,E,w)$
- Find: matching M maximizing $w(M)$
- $O(n^3)$

Maximum(-Cardinality) Matching

- Given $G=(V,E)$
- Find: matching M maximizing $|M|$
- Symmetric difference
- Augmenting paths
 - longer than $2k+1 \Rightarrow (1+1/k)$ -apx maximum

Maximal

- Maximal matching
 - 2-apx for maximum matching
 - 2-apx for VC
 - edge dominating set
- Minimum maximal matching – NP-complete
- al vs. um

Bipartite graph

- $G=(V,E)$
 - $V = M \cup W$ or $B \cup W$
- No m-m or w-w edge
 - no monochromatic edge
- E – subset of $M \times W$
- Matching $E' =$ marriages
 - perfect matching
 - $|M|=|W|$
 - $|E'|=n$
 - each m,w is in exactly one e' from E'
 - no single person

3DM

- Given:
 - sets M, W, D
 - $|M|=|W|=|D|=n$
 - E subset of $M \times W \times D$
- Is there a subset E' of E s.t.
 - $|E'|=n$
 - perfect matching
 - each m,w,d is in exactly one e' from e'
 - no single participant

Maximum Bipartite Matching: Hopcroft—Karp algorithm

- Given $G=(B \cup W, E)$;
 - $|B \cup W| = n$, $|E| = m$
- Find: matching M maximizing $|M|$
- Augmenting path:
 - BFS from unmatched B + DFS from unmatched W
- k phases
 - mk time
 - $> k$ paths
- $< n/k$ phases remain
 - $M \Delta M^*$ has augmenting paths of length $> k \Rightarrow$ has $< n/k$ paths $\Rightarrow < n/k$ edges in the difference; > 1 edge is added to M at every iteration
 - $< m^* n/k$ time remains
- total = $mk + mn/k$ for any k
 - $k = n^{1/2}$
 - $m n^{1/2} + m n/n^{1/2} = O(m n^{1/2}) = O(n^2 n^{1/2}) = O(n^{5/2})$
- VC, IS – poly in 2-part G

Hall's Theorem

- Given $G=(B \cup W, E)$
 - $|B|=|W|$
- $N(X) = \{y: x \text{ in } X, xy \text{ in } E\}$
- If G has perfect matching, then
- $|N(X)| \geq |X|$ for any X

Max-Weight Bipartite Matching

- $G = (B \cup W, E, w)$
- Cardinality iteration

(Some) Uncovered topics

- Exam questions:
 - how many questions, what kind of questions, how much time, how difficult will the questions be, how detailed explanations are expected; no devices
- Dir/undir graphs
- Stable matching, blossom (max-card matching in general graphs), edge cover vs MM
- 3SAT \rightarrow dirHamPath (Arora--Barak),
- DP for Euclidean TSP
- Median (including rnd alg)
- Lower bounds: EU \rightarrow sorting, 3SUM.
- Cook's Thm
- Radix sort
- Inapproximability
- Longest common subseq