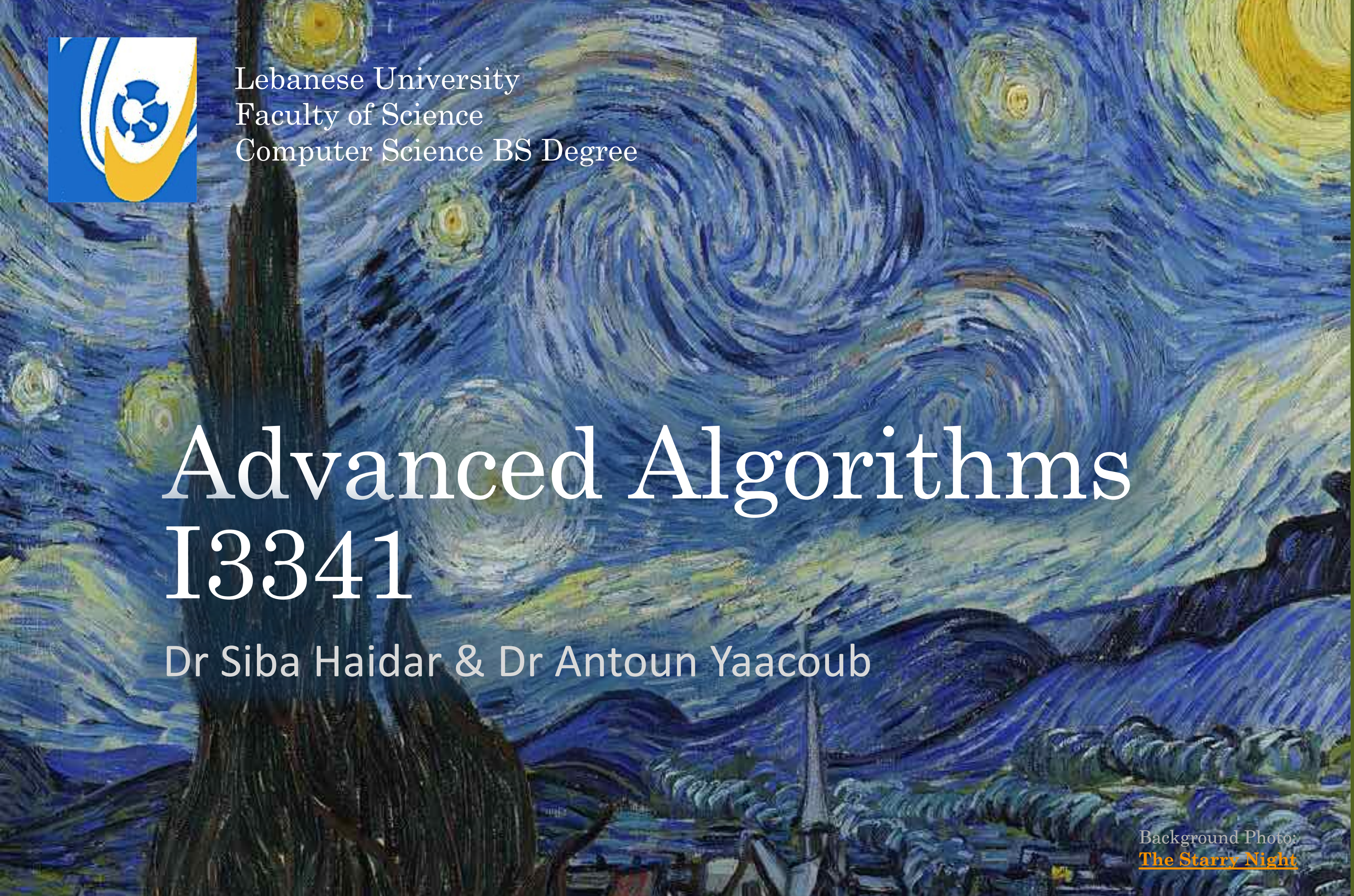


Lebanese University
Faculty of Science
Computer Science BS Degree

Advanced Algorithms I3341

Dr Siba Haidar & Dr Antoun Yaacoub

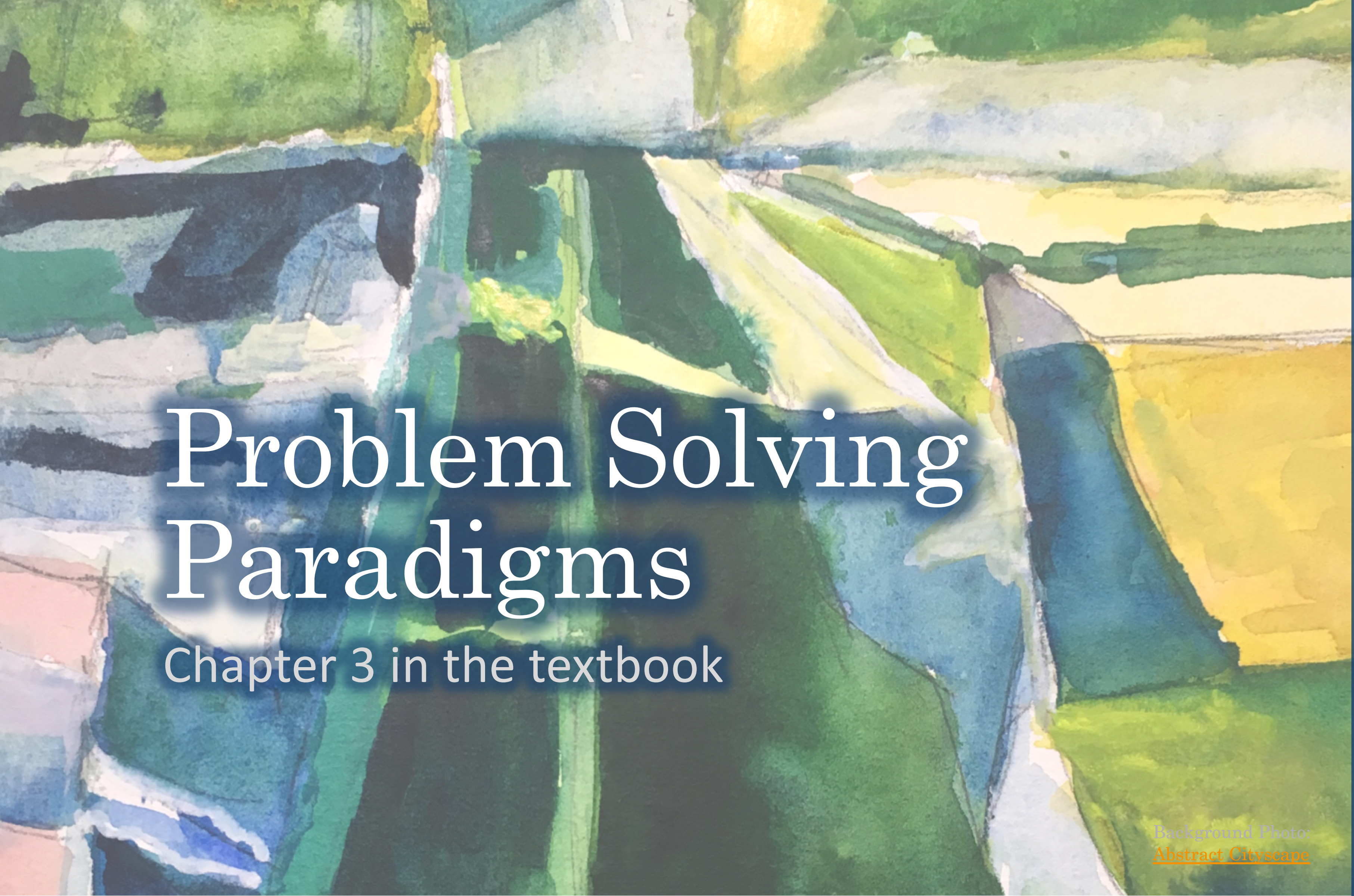
Background Photo: The Starry Night

Course Chapters

as per the textbook

Competitive Programming 3
The New Lower Bound of Programming Contests.
Steven Halim
Felix Halim
HANDBOOK FOR ACM ICPC AND IOI CONTESTANTS
2013

1. Introduction
2. Data Structures and Libraries
3. Problem Solving Paradigms
4. Graph
5. Mathematics
6. String Processing
7. Computational Geometry



Problem Solving Paradigms

Chapter 3 in the textbook

Background Photo: Abstract Cityscape

Dr Siba Haidar & Dr Antoun Yaacoub

Chapter Three Outline

in the textbook

1. Complete Search = Brute Force
    a. Iterative Complete Search
    b. Recursive Complete Search
    c. Tips
2. Divide and Conquer
    a. Interesting Usages of Binary Search
3. Greedy
    a. Examples
4. Dynamic Programming
    a. DP Illustration
    b. Classical Examples
    c. Non-Classical Examples

Greedy

- *having an excessive desire or appetite for food*
- *having or showing an intense and selfish desire for wealth or power*

Background Photo:
La Vague

Greedy

- an algorithm is said to be greedy if
  - it makes the locally optimal choice at each step
  - with the hope of eventually reaching the globally optimal solution
- in some cases
  - greedy works
  - the solution is short and runs efficiently
- for many others
  - it does not
- a problem must exhibit these two properties in order for a greedy algorithm to work:
- 1. optimal sub-structures
  - optimal solution to the problem contains optimal solutions to the sub-problems
- 2. greedy property
  - difficult to prove in time-critical contest environment!
  - if we make a choice that seems like the best at the moment and proceed to solve the remaining sub-problem → we reach the optimal solution
  - we will never have to reconsider our previous choices

Examples

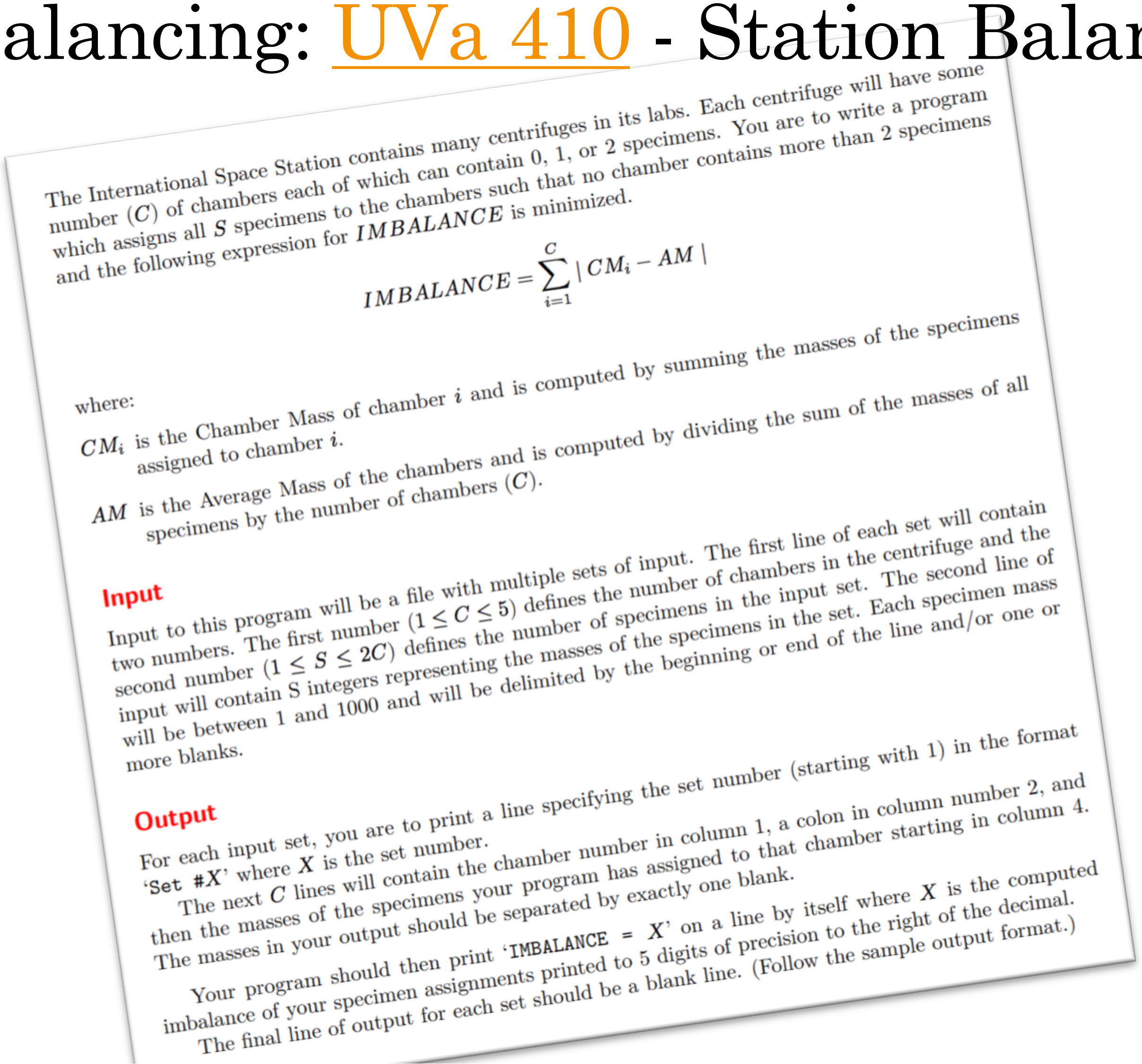
Background Photo: Bridge

Coin Change - The Greedy Version

- problem description
  - given target amount V cents, and
  - list of denominations of n coins
    - coinValue[i] for i ∈ [0..n-1]
  - what is the minimum number of coins that we must use to represent amount V?
  - assume unlimited supply
- example
  - n = 4; coinValue = {25, 10, 5, 1} cents
    - 1-cent coin → can make every value
  - V = 42 cents
  - → greedy: select largest coin denomination <= remaining amount
  - 42 - 25 : 17 - 10 : 7 - 5 : 2 - 1 : 1 - 1 : 0
  - → 5 coins → optimal

- 2 ingredients :
- 1. optimal sub-structures
  - optimal solutions to sub-problem contained within solution
  - 17 cents = 10 + 5 + 1 + 1
  - part of solution for 42 cents
- 2. greedy property
  - using any other strategies will not lead to an optimal solution
- however does not work for all
  - list: {4, 3, 1} cents; V = 6;
  - greedy → 3 coins: 4, 1, 1
  - optimal → 2 coins: 3, 3!
  - general version revisited in DP

Load Balancing: UVa 410 - Station Balance



The International Space Station contains many centrifuges in its labs. Each centrifuge will have some number (C) of chambers each of which can contain 0, 1, or 2 specimens. You are to write a program which assigns all S specimens to the chambers such that no chamber contains more than 2 specimens and the following expression for $IMBALANCE$ is minimized.

$$IMBALANCE = \sum_{i=1}^{C} | CM_i - AM |$$

where:

CM_i is the Chamber Mass of chamber i and is computed by summing the masses of the specimens assigned to chamber i.

AM is the Average Mass of the chambers and is computed by dividing the sum of the masses of all specimens by the number of chambers (C).

Input

Input to this program will be a file with multiple sets of input. The first line of each set will contain two numbers. The first number ($1 \le C \le 5$) defines the number of chambers in the centrifuge and the second number ($1 \le S \le 2C$) defines the number of specimens in the input set. The second line of input will contain S integers representing the masses of the specimens in the set. Each specimen mass will be between 1 and 1000 and will be delimited by the beginning or end of the line and/or one or more blanks.

Output

For each input set, you are to print a line specifying the set number (starting with 1) in the format '`Set #`X' where X is the set number.

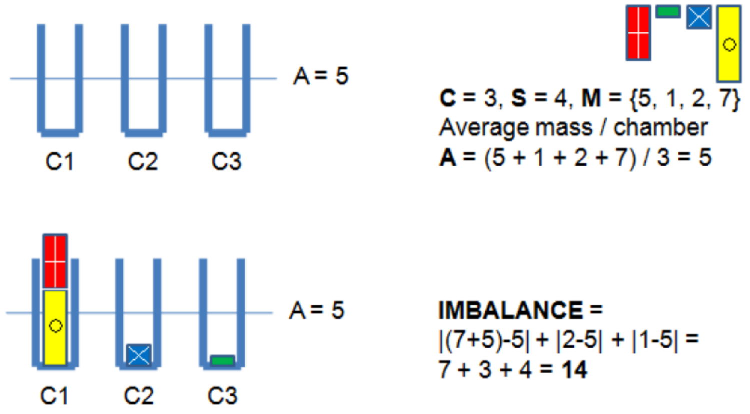
The next C lines will contain the chamber number in column 1, a colon in column number 2, and then the masses of the specimens your program has assigned to that chamber starting in column 4. The masses in your output should be separated by exactly one blank.

Your program should then print '`IMBALANCE = `X' on a line by itself where X is the computed imbalance of your specimen assignments printed to 5 digits of precision to the right of the decimal.

The final line of output for each set should be a blank line. (Follow the sample output format.)

Load Balancing: UVa 410 - Station Balance

- given
  - 1 ≤ C ≤ 5 chambers which can store 0, 1, or 2 specimens,
  - 1 ≤ S ≤ 2C specimens, and
  - list M of the masses of the S specimens
- determine which chamber should store each specimen in order to minimize $Imbalance$
  - $A = {(\sum_{j=1}^{S} M_j)}/{C}$
    - average of total mass in each of the C chambers
  - $Imbalance = \sum_{i=1}^{C} |X_i - A|$
    - sum of differences between the total mass in each chamber w.r.t. A
    - X_i total mass in chamber i

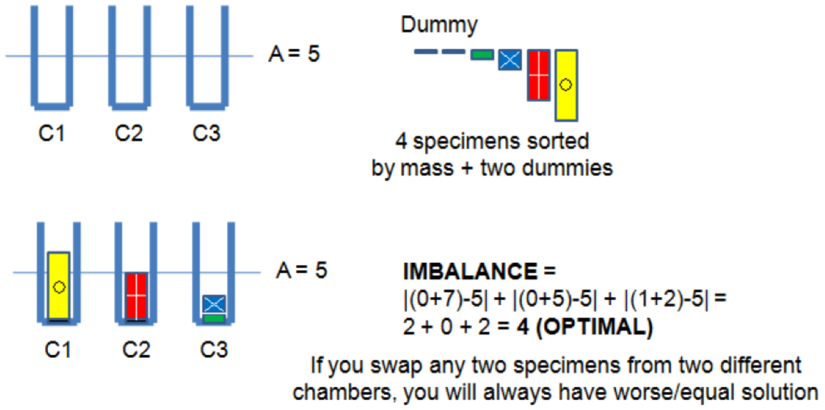


C = 3, **S** = 4, **M** = {5, 1, 2, 7}
Average mass / chamber
A = (5 + 1 + 2 + 7) / 3 = 5

IMBALANCE =
|(7+5)-5| + |2-5| + |1-5| =
7 + 3 + 4 = **14**

Load Balancing: UVa 410 - Station Balance

- observation #1:
  - if ∃ empty chamber → move one specimen from a chamber with two specimens to the empty chamber!
- observation #2:
  - If S > C, then S – C specimens must be paired with a chamber already containing other specimens
  - Pigeonhole principle! (next slide)
- key insight
  - solution simplified with sorting:
  - if S < 2C, add 2C – S dummy specimens with mass 0
- greedy strategy
  - pair specimens from two extremities and put them in chambers
- example, C = 3, S = 4,
  - M = {5, 1, 2, 7} → C = 3, S = 6
  - $2 \times C - S = 2$ → add 2 zeros: M={5, 1, 2, 7, 0, 0}
  - sort specimens → {0, 0, 1, 2, 5, 7}



A = 5
Dummy
C1
C2
C3
4 specimens sorted by mass + two dummies
A = 5
C1
C2
C3
IMBALANCE =
|(0+7)-5| + |(0+5)-5| + |(1+2)-5| =
2 + 0 + 2 = 4 (OPTIMAL)
If you swap any two specimens from two different chambers, you will always have worse/equal solution

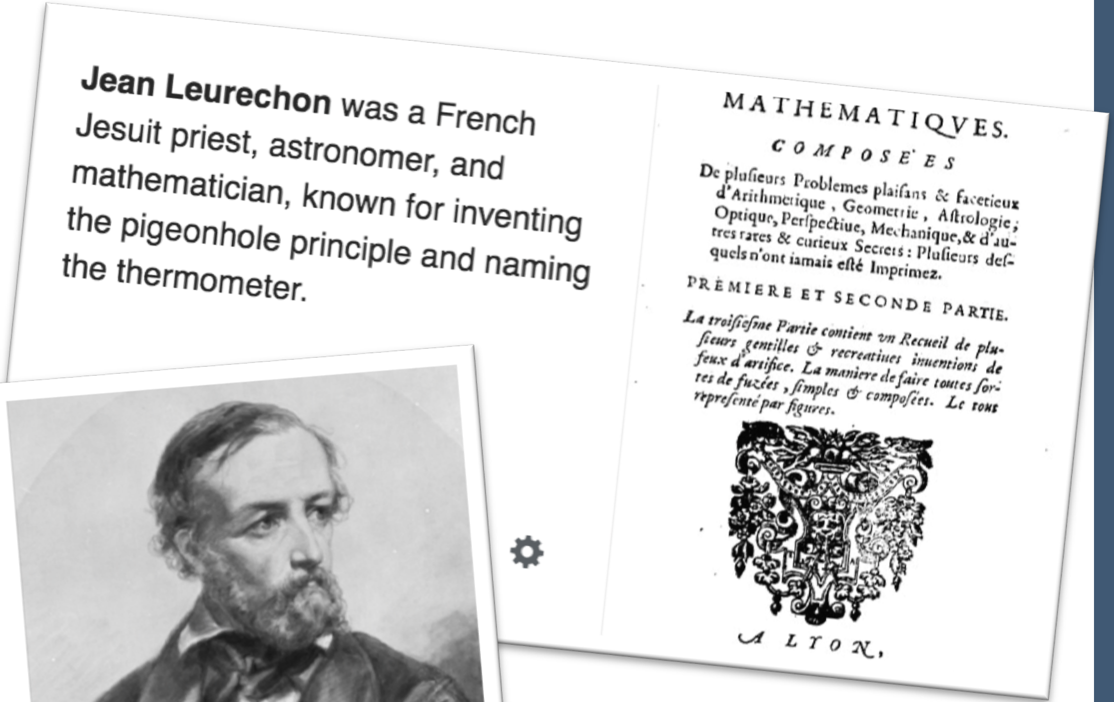
Figure 3.6: UVa 410 - Greedy Solution

Pigeonhole principle

- in mathematics
  - if n items are put into m containers
    - n > m
  - then at least one container must contain more than one item
- 1624 by Jean Leurechon
- 1834 treated by Dirichlet
  - Schubfachprinzip
  - "drawer|shelf principle"

- pigeons in holes
  - here there are n = 10 pigeons in m = 9 holes
  - since 10 is greater than 9, the pigeonhole principle says that at least one hole has more than one pigeon

Who are they?



Jean Leurechon was a French Jesuit priest, astronomer, and mathematician, known for inventing the pigeonhole principle and naming the thermometer.

MATHEMATIQVES.

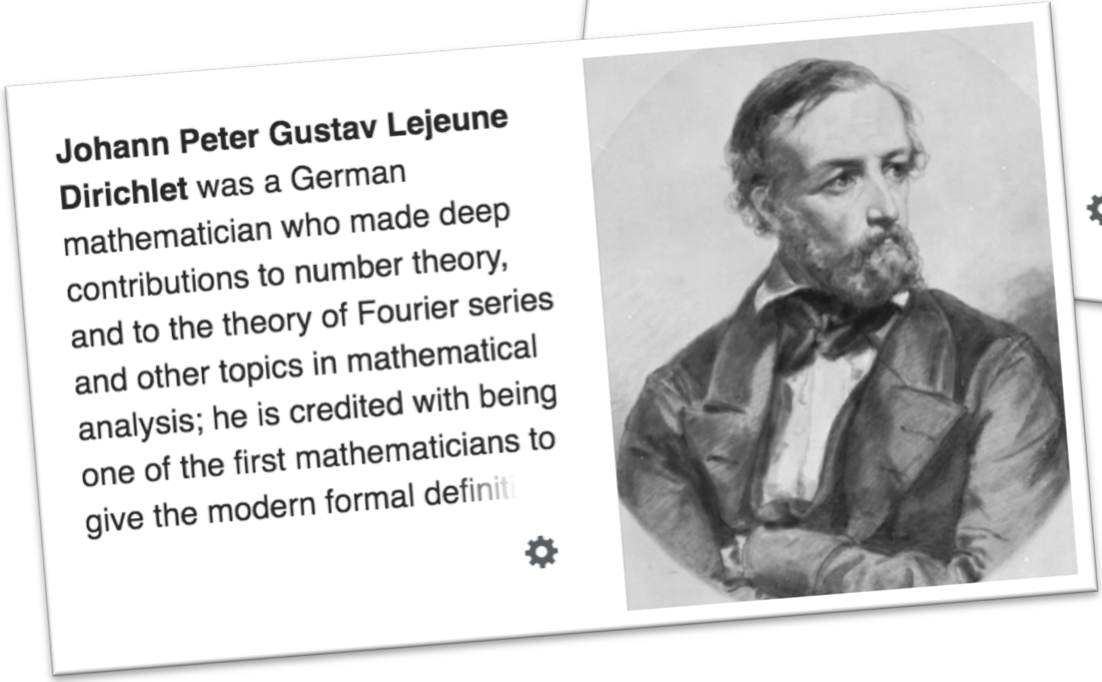
COMPOSE'ES

De pluſieurs Problemes plaiſans & faceticux d'Arithmetique, Geometrie, Aſtrologie; Optique, Perſpectiue, Mechanique, & d'autres rares & curieux Secrets: Pluſieurs deſquels n'ont iamais eſté Imprimez.

PREMIERE ET SECONDE PARTIE.

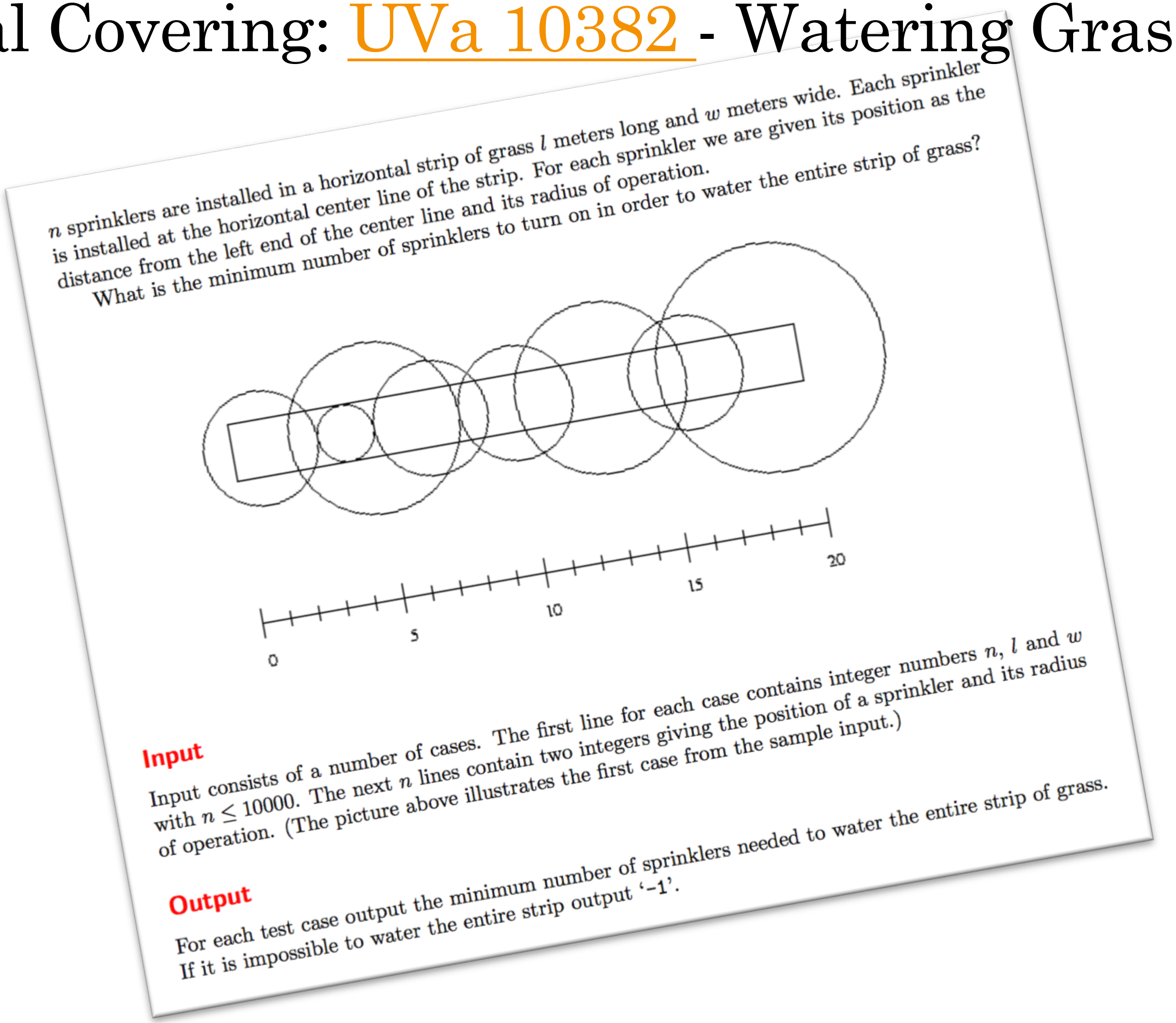
La troiſieſme Partie contient vn Recueil de pluſieurs gentilles & recreatiues inuentions de feux d'artifice. La maniere de faire toutes sortes de fuzées, ſimples & compoſées. Le tout repreſenté par figures.

A LYON.



Johann Peter Gustav Lejeune Dirichlet was a German mathematician who made deep contributions to number theory, and to the theory of Fourier series and other topics in mathematical analysis; he is credited with being one of the first mathematicians to give the modern formal definiti

Interval Covering: UVa 10382 - Watering Grass



n sprinklers are installed in a horizontal strip of grass l meters long and w meters wide. Each sprinkler is installed at the horizontal center line of the strip. For each sprinkler we are given its position as the distance from the left end of the center line and its radius of operation.

What is the minimum number of sprinklers to turn on in order to water the entire strip of grass?

Input

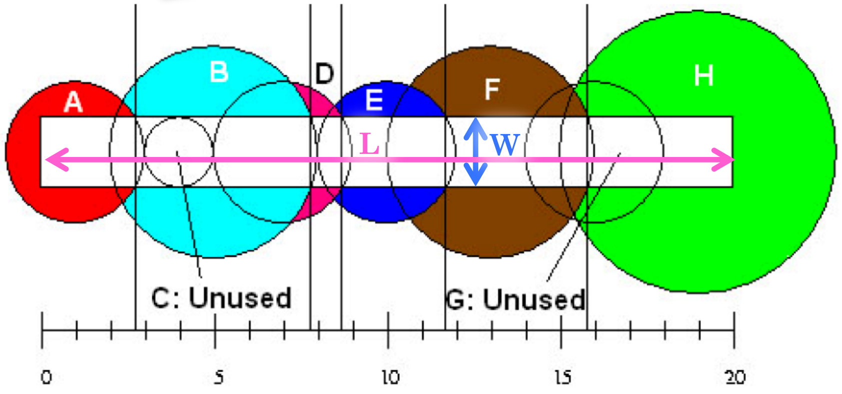
Input consists of a number of cases. The first line for each case contains integer numbers n, l and w with $n \leq 10000$. The next n lines contain two integers giving the position of a sprinkler and its radius of operation. (The picture above illustrates the first case from the sample input.)

Output

For each test case output the minimum number of sprinklers needed to water the entire strip of grass. If it is impossible to water the entire strip output '−1'.

Interval Covering: UVa 10382 - Watering Grass

- problem description
  - n sprinklers in a strip of grass L x W
  - n ≤ 10000
  - each sprinkler centered vertically in the strip + given its radius & position: distance from left end of center line
- question:
  - what is the minimum number of sprinklers that should be turned ON in order to water the entire strip of grass?
- example :
  - answer = 6
  - → {A, B, D, E, F, H} + 2 unused: {C, G}
- brute force strategy?
  - try all possible subsets of sprinklers to be turned on?
  - 2^{10000}→ infeasible ⊗
  - because $2^{30} = 1{,}073{,}741{,}824 \cong giga$

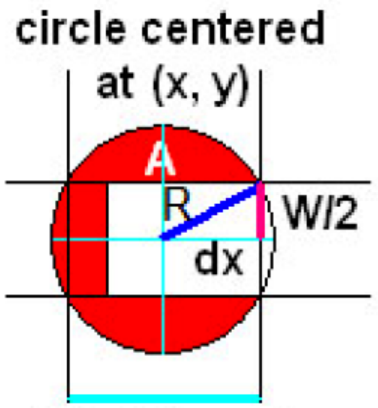


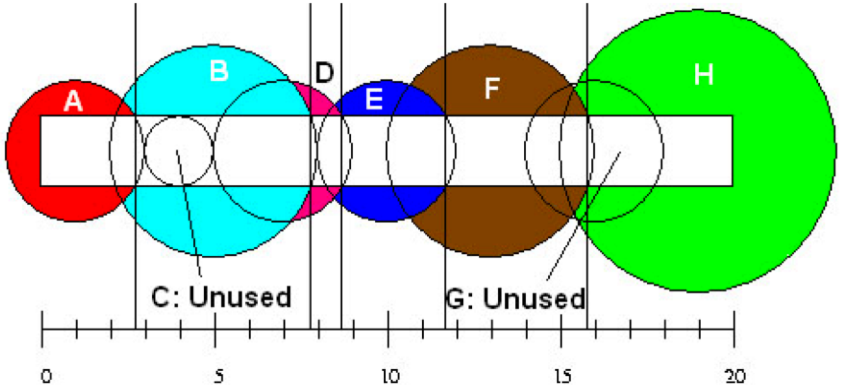
Interval Covering: UVa 10382 - Watering Grass

- interval covering problem + simple geometric twist
- greedy:
  - sort the intervals by
    - increasing left endpoint
    - decreasing right endpoint
  - take interval that
    - covers 'as far right as possible'
    - uninterrupted coverage
  - ignore already covered
- ex:
  - take A, B
  - ignore C ($C \subset B$)
  - take D ($B \cap E = \emptyset$), E, F,
  - ignore G & take H
    - G ! as far & $H \cap F \neq \emptyset$
  - → min = 6

$$dx = \sqrt{R^2 - (W/2)^2}$$

$$[x - dx \,..\, x + dx]$$





Interval Covering: UVa 10382 - Watering Grass

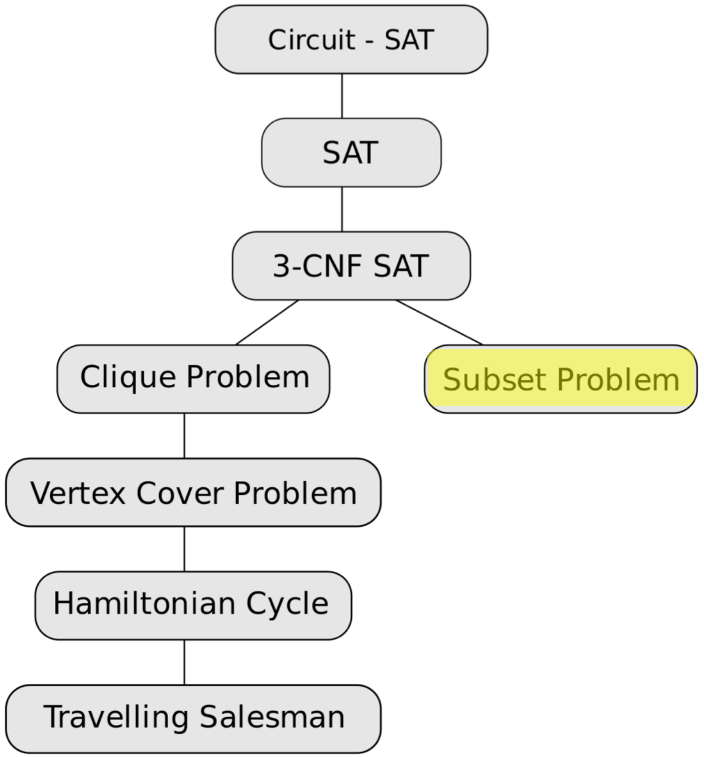
```
#include <bits/stdc++.h>
using namespace std;
const double EPS = 1e-9;
struct sp {
  int x, r;
  double x_l, x_r;
};
sp sprinkler[10010];
bool cmp(sp a, sp b) {
  if (fabs(a.x_l - b.x_l) > EPS)
    return a.x_l < b.x_l;
  else
    return a.x_r > b.x_r;
}
int main() {
  int n, l, w;
  while (scanf("%d %d %d", &n, &l, &w) != EOF) {
    for (int i = 0; i < n; ++i) {
      scanf("%d %d", &sprinkler[i].x, &sprinkler[i].r);
      if (2*sprinkler[i].r >= w) {
        double d_x = sqrt((double)
             sprinkler[i].r*sprinkler[i].r
             - (w/2.0) * (w/2.0));
        sprinkler[i].x_l = sprinkler[i].x-d_x;
        sprinkler[i].x_r = sprinkler[i].x+d_x;
      }
      else
        sprinkler[i].x_l = sprinkler[i].x_r
                         = sprinkler[i].x;
    }
```

```
sort(sprinkler, sprinkler+n, cmp);
  bool possible = true;
  double covered = 0.0;
  int ans = 0;
  for (int i = 0; (i < n) && possible; ++i) {
    if (covered > l) break;
    if (sprinkler[i].x_r < covered+EPS)
      continue;
    if (sprinkler[i].x_l < covered+EPS) {
      double max_r = -1.0;
      int max_id;
      for (int j = i; (j < n)
       && (sprinkler[j].x_l < covered + EPS); j++)
        if (sprinkler[j].x_r > max_r) {
          max_r = sprinkler[j].x_r;
          max_id = j;
        }
      ++ans;
      covered = max_r;
      i = max_id;
    }
    else  possible = false;
  }
  if (!possible || (covered < l))
        printf("-1\n");
  else  printf("%d\n", ans);
}
return 0;
}
```

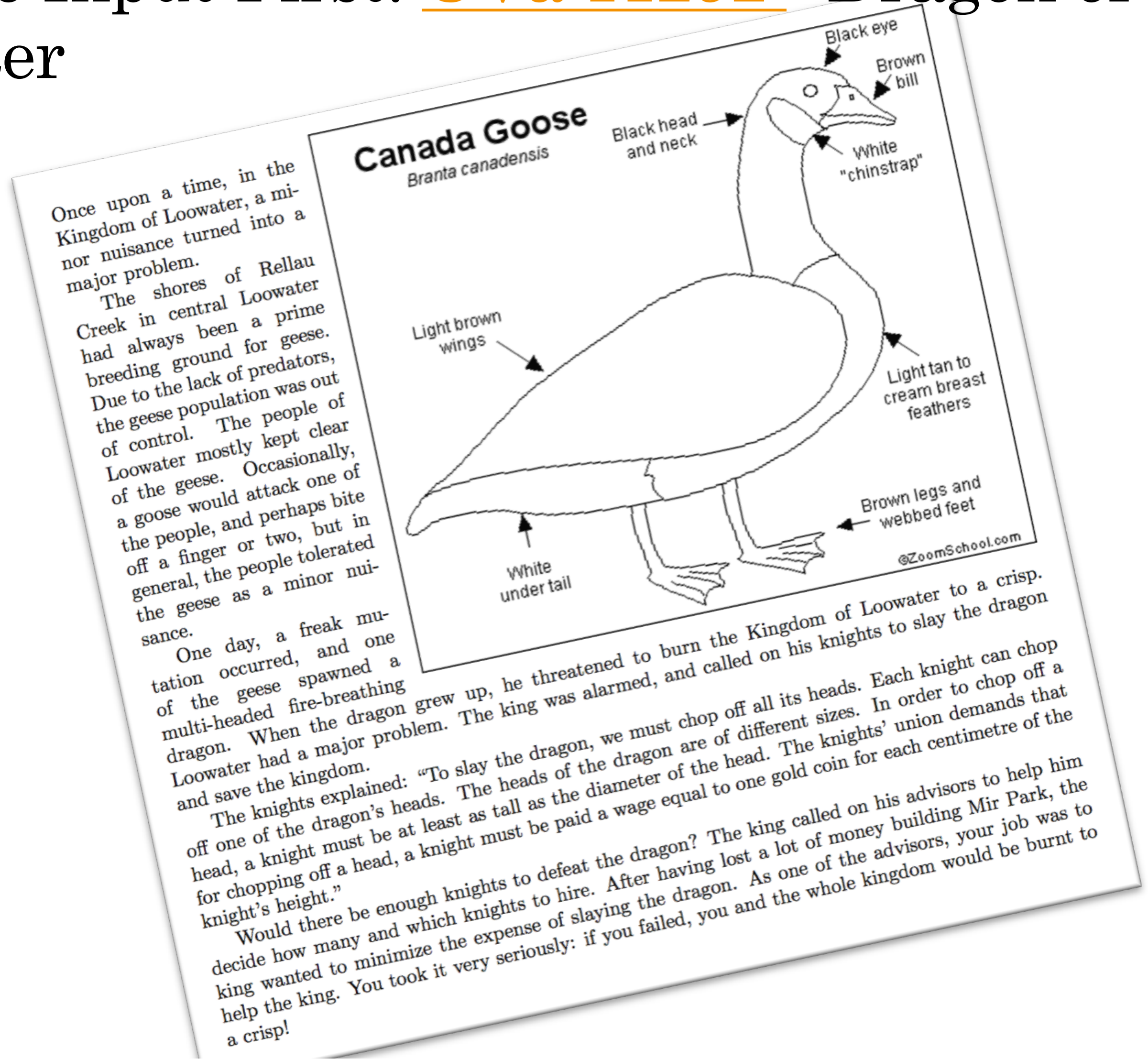
Set cover problem

- classical question
  - in combinatorics, computer science, operations research, and complexity theory
- one of Karp's 21 NP-complete problems
- "whose study has led to the development of fundamental techniques for the entire field" of approximation algorithms
- given a universe {1,2,...,n}
  - and a collection S of m sets whose union equals the universe,
  - identify the smallest sub-collection of S whose union equals the universe
- example
  - universe U={1,2,3,4,5} and S={{1,2,3},{2,4},{3,4},{4,5}}
  - can cover with smaller number of sets: {{1,2,3},{4,5}}

- some NP-complete problems



Sort the Input First: UVa 11292 - Dragon of Loowater



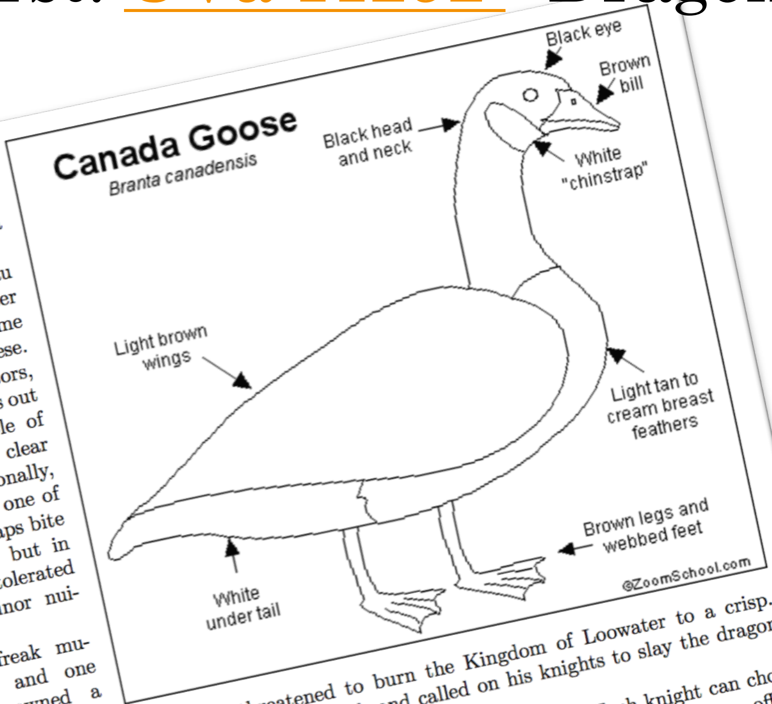
Once upon a time, in the Kingdom of Loowater, a minor nuisance turned into a major problem.

The shores of Rellau Creek in central Loowater had always been a prime breeding ground for geese. Due to the lack of predators, the geese population was out of control. The people of Loowater mostly kept clear of the geese. Occasionally, a goose would attack one of the people, and perhaps bite off a finger or two, but in general, the people tolerated the geese as a minor nuisance.

One day, a freak mutation occurred, and one of the geese spawned a multi-headed fire-breathing dragon. When the dragon grew up, he threatened to burn the Kingdom of Loowater to a crisp. Loowater had a major problem. The king was alarmed, and called on his knights to slay the dragon and save the kingdom.

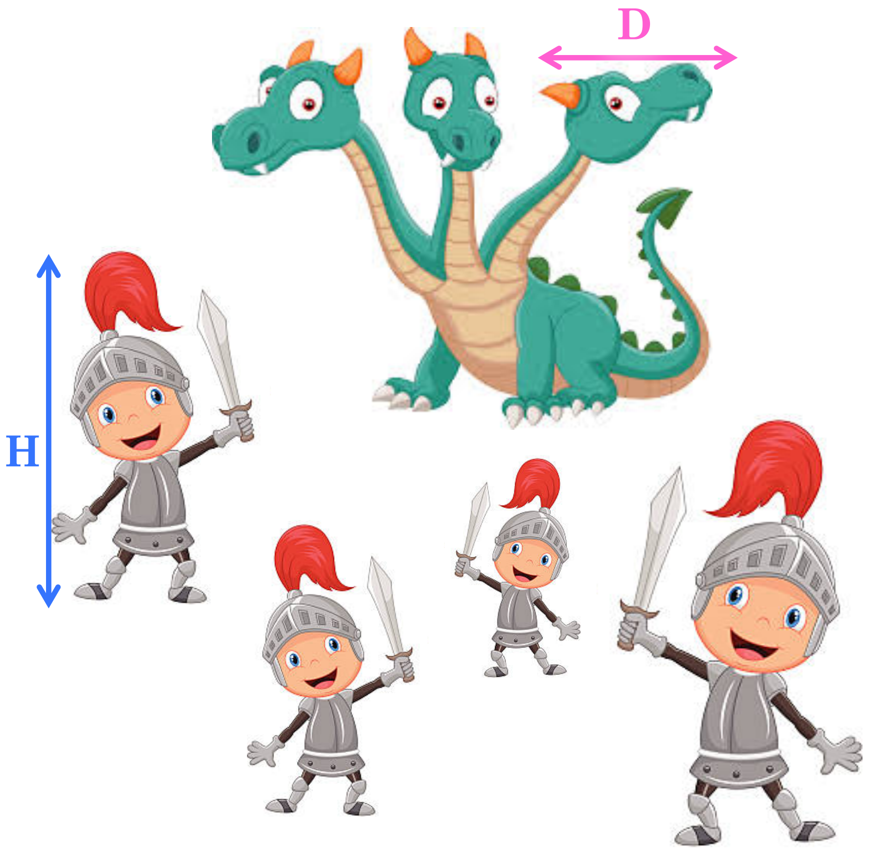
The knights explained: "To slay the dragon, we must chop off all its heads. Each knight can chop off one of the dragon's heads. The heads of the dragon are of different sizes. In order to chop off a head, a knight must be at least as tall as the diameter of the head. The knights' union demands that for chopping off a head, a knight must be paid a wage equal to one gold coin for each centimetre of the knight's height."

Would there be enough knights to defeat the dragon? The king called on his advisors to help him decide how many and which knights to hire. After having lost a lot of money building Mir Park, the king wanted to minimize the expense of slaying the dragon. As one of the advisors, your job was to help the king. You took it very seriously: if you failed, you and the whole kingdom would be burnt to a crisp!



Sort the Input First: UVa 11292 - Dragon of Loowater

- problem description:
  - n dragon heads & m knights
  - 1 ≤ n,m ≤ 20000
  - dragon head → diameter D
  - knight → height H
  - if D ≤ H
    - knight can chop dragon head
  - 1 knight → 1 dragon head
- given
  - list of n diameters
  - list of m heights
- question
  - is it possible to chop off all the dragon heads?
  - if yes, what is the minimum total height of the knights used to chop off the dragons' heads?



Sort the Input First: UVa 11292 - Dragon of Loowater

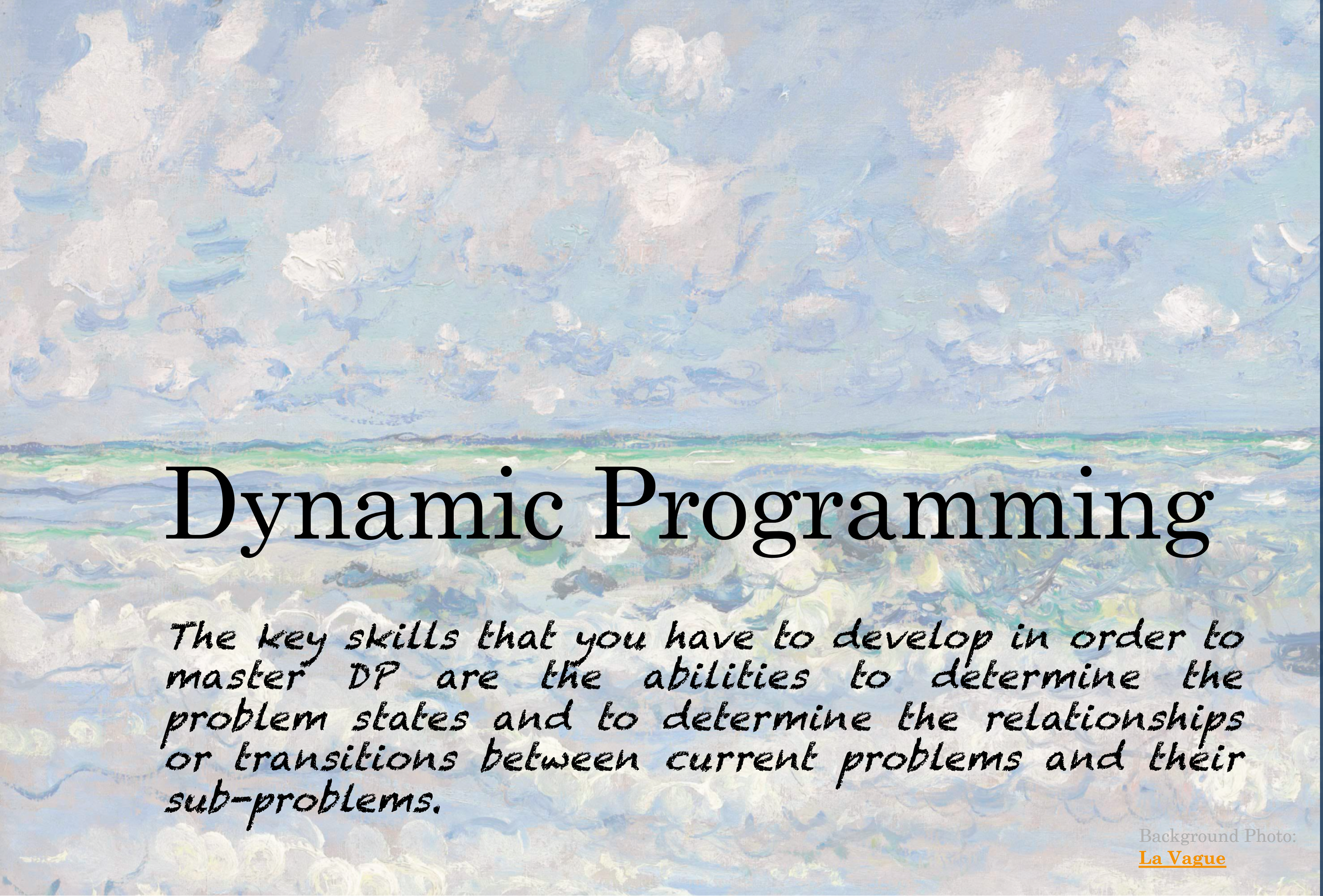
- bipartite matching problem
  - pair in a maximal fashion
- greedy
  - dragon head should be chopped by a knight with the shortest height that is at least as tall as the diameter of the dragon's head
  - **sort** both lists: $O(nlogn + mlogm)$
  - O(min(n,m)) scan:

```
typedef vector<int> vi;
int main() {
  int n, m;
  while (scanf("%d %d", &n, &m), (n || m)) {
    vi D(n);
    vi H(m);
    for (int d = 0; d < n; ++d)
      scanf("%d", &D[d]);
    for (int k = 0; k < m; ++k)
      scanf("%d", &H[k]);
    sort(D.begin(), D.end());
    sort(H.begin(), H.end());

    int gold = 0, d = 0, k = 0;
    while ((d < n) && (k < m)) {
      while ((k < m) && (D[d] > H[k])) ++k;
      if (k == m) break;
      gold += H[k];
      ++d; ++k;
    }
    if (d == n)
      printf("%d\n", gold);
    else
      printf("Loowater is doomed!\n");
  }
  return 0;
}
```

Remarks About Greedy Algorithm in Programming Contests

- seen 3 classical problems solvable with Greedy algs → memorize solutions ☺
- important strategy
  - sort input data to elucidate hidden greedy strategies
- 2 other classical examples:
  - Kruskal's & Prim's for MST
  - Dijkstra's for SSSP
  - + others
- greedy is risky → no TL but maybe WA
- proof is time consuming
- rule of thumb:
  - if input size 'small enough' for CS or DP use them
  - else → greedy
- however problem setters set input bounds ambiguous;
  - contestants cannot quickly determine required algorithm!
- luckily # novel Greedy problems is low



Dynamic Programming

The key skills that you have to develop in order to master DP are the abilities to determine the problem states and to determine the relationships or transitions between current problems and their sub-problems.

Background Photo: La Vague

Dr Siba Haidar & Dr Antoun Yaacoub