Chapter 1

Introduction

Why to study algorithms?

- It is commonly believed that in order to obtain high speed computation, it suffices to have a very high speed computer. <u>This is, however, not entirely true</u>.
- A good algorithm implemented on a slow computer may perform much better than a bad algorithm implemented on a fast computer.

Sorting problem:

- To sort a set of elements into increasing or decreasing order.

  11, 7, 14, 1, 5, 9, 10

  ↓sort

  1, 5, 7, 9, 10, 11, 14

- Sorting Algorithm:
  - Insertion sort
  - Quick sort
  - Etc.

Insertion Sort

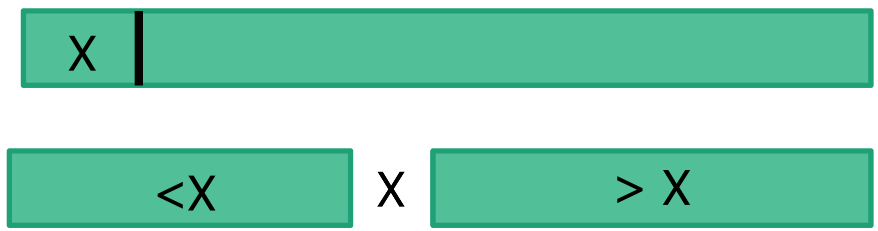
11, 7, 14, 1, 5, 9, 10.

Insertion sort works on the above sequence of data elements as follows:

Sorted Sequence	Unsorted Sequence
11,	7, 14, 1, 5, 9, 10
7, 11	14, 1, 5, 9, 10
7, 11, 14	1, 5, 9, 10
1, 7, 11, 14	5, 9, 10
√ 1, 5, 7, 11, 14	9, 10
1, 5, 7, 9, 11, 14	10
1, 5, 7, 9, 10, 11, 14	

QuickSort

- Quicksort would use the first data element , say x, to divide all data elements into three subsets:
  - those smaller than x,
  - those larger than x, and
  - those equal to x.



- Divide approach & recursive algorithm

(5, 1, 8, 7, 3) (10) (17, 14, 26, 21).

We now have to sort two sequences:

(5, 1, 8, 7, 3) → (1, 3) (5) (7, 8)

and (17, 14, 26, 21). → (14) (17) (26, 21)

Quicksort Example

Input: 10, 5, 1, 17, 14, 8, 7 , 26, 21, 3

Steps: **10, 5, 1, 17, 14, 8, 7 , 26, 21, 3**

(5, 1, 8, 7, 3) (10) (17, 14, 26, 21).

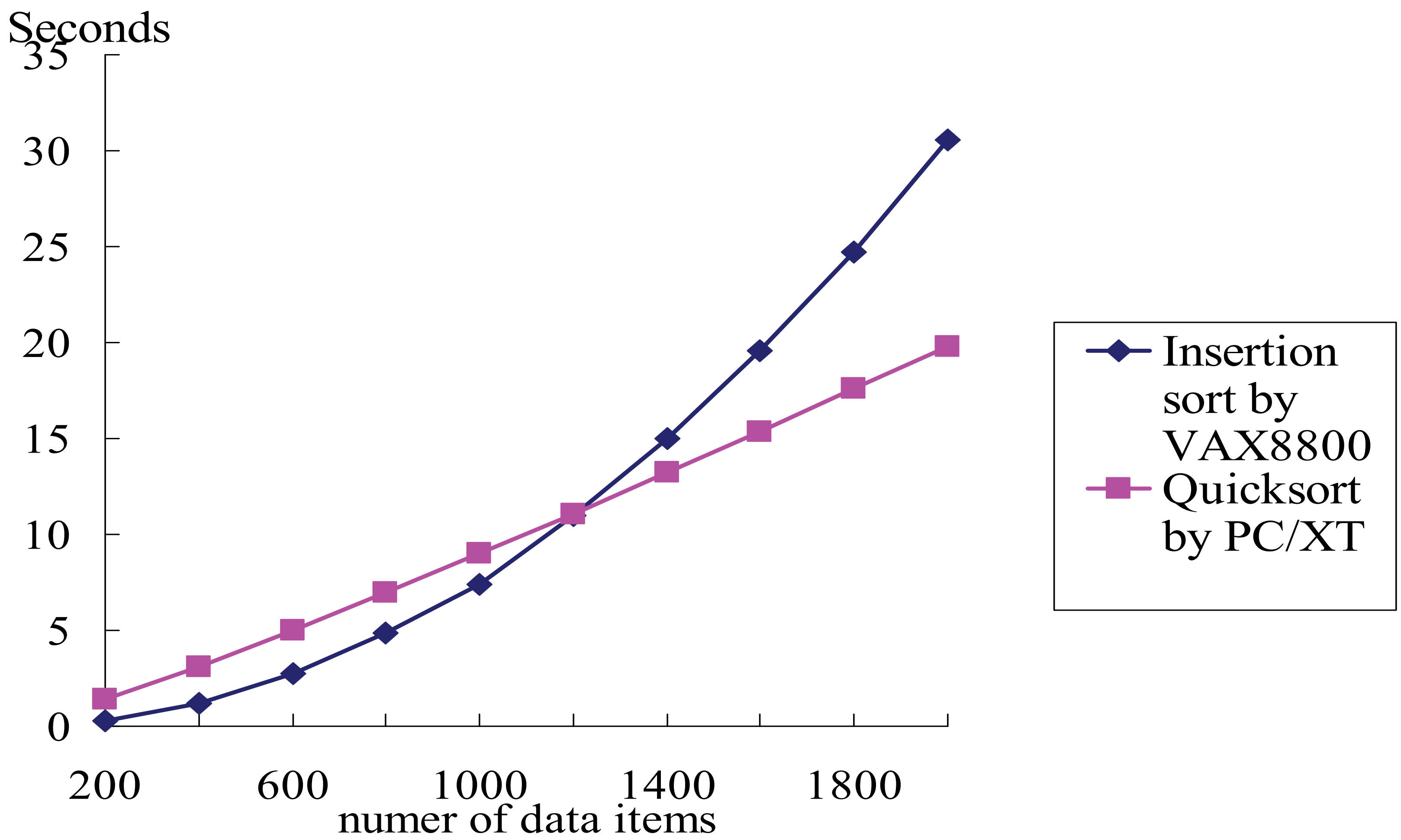
We now have to sort two sequences:

(5, 1, 8, 7, 3) $\Rightarrow$ (1, 3) (5) (7, 8)

and (17, 14, 26, 21). $\Rightarrow$ (14) (17) (26, 21)

Comparison of algorithms

- Comparison of two algorithms implemented on two computers (average of ten times)



Analysis of algorithms

- Measure the goodness of algorithms
  - efficiency
  - asymptotic notations: e.g. $O(n^2)$
  - worst case
  - average case
  - best case
  - amortized analysis (均攤分析法)
- Measure the difficulty of problems
  - NP-complete ($\geq O(2^n)$) or polynomial solvable
  - Undecidable
  - lower bound
- Is the algorithm optimal?

Asymptotic notations

- **Def: $f(n) = O(g(n))$** "at most"

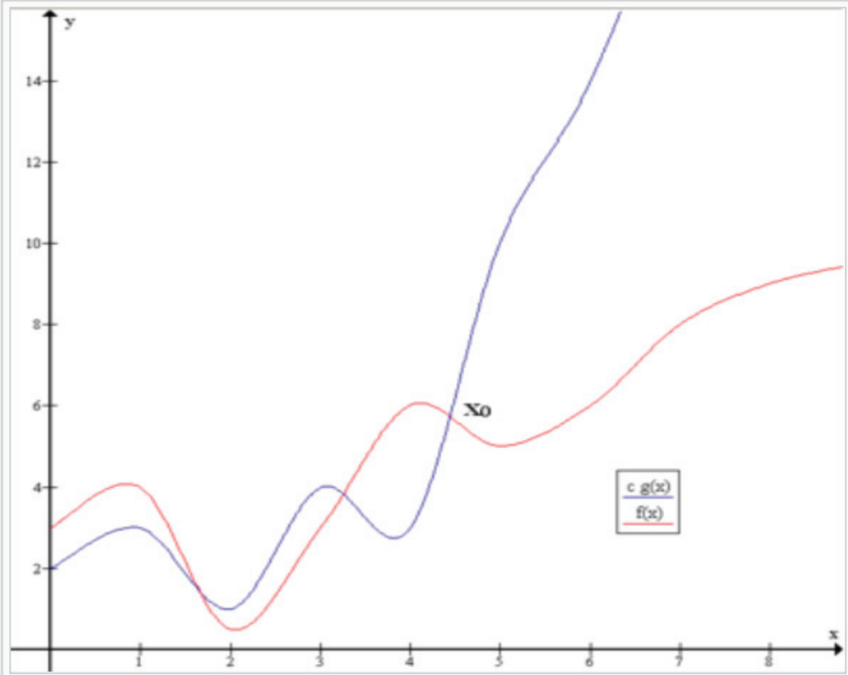
  $\exists\ c, n_0 \rightarrow |f(n)| \leq c|g(n)| \quad \forall\ n \geq n_0$

- e.g. $f(n) = 3n^2 + 2$

  $g(n) = n^2$

  $\Rightarrow n_0=2, c=4$

  $\therefore\ f(n) = O(n^2)$



Example of Big O notation: $f(x) \in O(g(x))$ as there exists $c > 0$ (e.g., $c = 1$) and x_0 (e.g., $x_0 = 5$) such that $f(x) < cg(x)$ whenever $x > x_0$.

- e.g. $f(n) = n^3 + n = O(n^3)$
- e. g. $f(n) = 3n^2 + 2 = O(n^3)$ or $O(n^{100})$

Asymptotic notations

- **Def : $f(n) = \Omega(g(n))$** "at least", "lower bound"

  $\exists$ c, and n_0, $\ni |f(n)| \geq c|g(n)|\ \forall\ n \geq n_0$

  e. g. $f(n) = 3n^2 + 2 = \Omega(n^2)$ or $\Omega(n)$

- **Def : $f(n) = \Theta(g(n))$**

  $\exists\ c_1, c_2$, and n_0, $\ni c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|\ \forall\ n \geq n_0$

  e. g. $f(n) = 3n^2 + 2 = \Theta(n^2)$

- **Def : $f(n) \sim o(g(n))$**

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \to 1$$

e.g. $f(n) = 3n^2+n = o(3n^2)$

Problem size n and function

	10	10^2	10^3	10^4
$\log_2 n$	3.3	6.6	10	13.3
n	10	10^2	10^3	10^4
$n\log_2 n$	0.33×10^2	0.7×10^3	10^4	1.3×10^5
n^2	10^2	10^4	10^6	10^8
2^n	1024	1.3×10^{30}	$>10^{100}$	$>10^{100}$
$n!$	3×10^6	$>10^{100}$	$>10^{100}$	$>10^{100}$

Time Complexity Functions

Common computing time functions

- Time complexity classes:
  - $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$
  - Exponential algorithm: $O(2^n)$
  - polynomial algorithm: e.g. $O(n^2)$, $O(n\log n)$, …

- Algorithm A : $O(n^3)$, algorithm B : $O(n)$
  - Should Algorithm B run faster than A?

    NO !
  - It is true only when **n is large enough**!

Introduction

- How do we measure the goodness of an algorithm?
- How do we measure the difficulty of a problem?
- How do we know that an algorithm is optimal for a problem?
- How can we know that there does not exist any other better algorithm to solve the same problem?

The goodness of an algorithm

- Time complexity (more important)
- Space complexity (memory size)
- For a parallel algorithm :
  - time-processor product
  - O(log n) time, O(n) processors $\rightarrow$ O(n log n)
- For a VLSI circuit :
  - area-time (AT, AT^2), A is the area of the VLSI

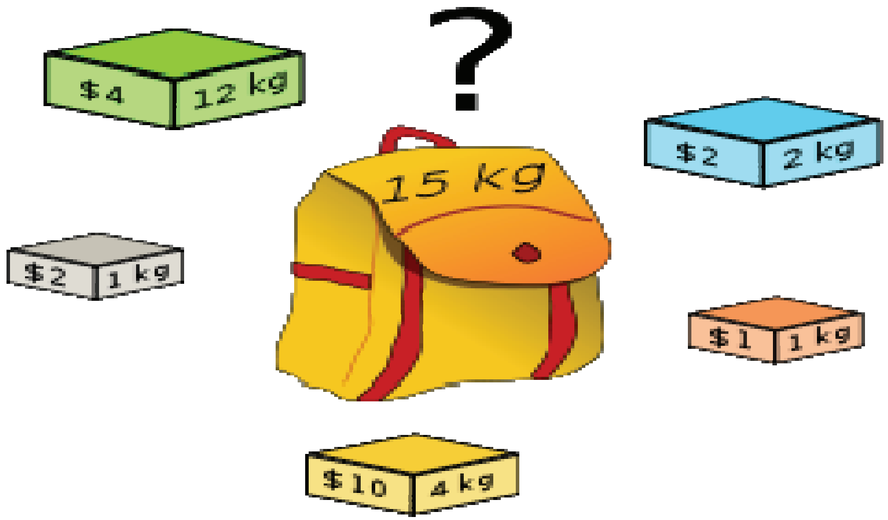
Undecidable problems

- An undecidable problem is a decision problem (output T or F) for which it is known to be impossible to construct a single algorithm that always leads to a correct yes-or-no answer.
- Example:
  - **Halting problem**: "Given a description of an arbitrary computer program, decide whether the program finishes running or continues to run forever".
  - This is equivalent to the problem of deciding, given a program and an input, whether the program will eventually halt when run with that input, or will run forever.
  - Alan Turing proved in 1936.

http://en.wikipedia.org/wiki/Halting_problem

0/1 Knapsack problem 背包問題

	P_1	P_2	P_3	P_4	P_5
Value	4	2	10	1	2
Weight	12	1	4	1	2

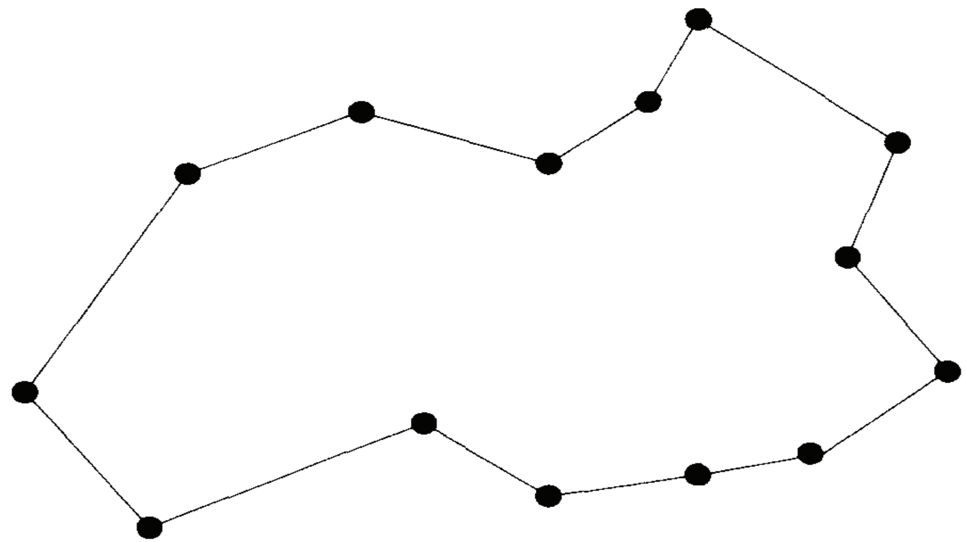


- **M** (total weight limitation)=15;
- 0/1 constraint;
- best solution (maximal sum of weight)?
- This problem is NP-complete.
- As the number of items becomes very large, it is very hard to find an optimal solution.

Traveling salesperson problem (TSP)

- Given: A set of n planar points

  Find: A closed tour which includes all points exactly once such that its total length is minimized.
- This problem is NP-complete.



http://www.tsp.gatech.edu/

http://www.math.uwaterloo.ca/tsp/index.html

Partition problem

- Given: A set of positive integers S

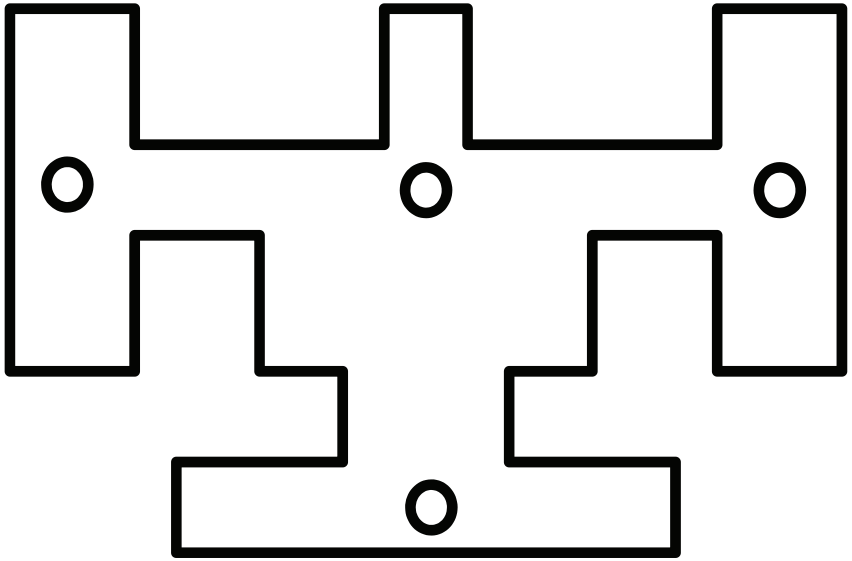
  Find: S_1 and S_2 such that $S_1 \cap S_2 = \varnothing$, $S_1 \cup S_2 = S$,

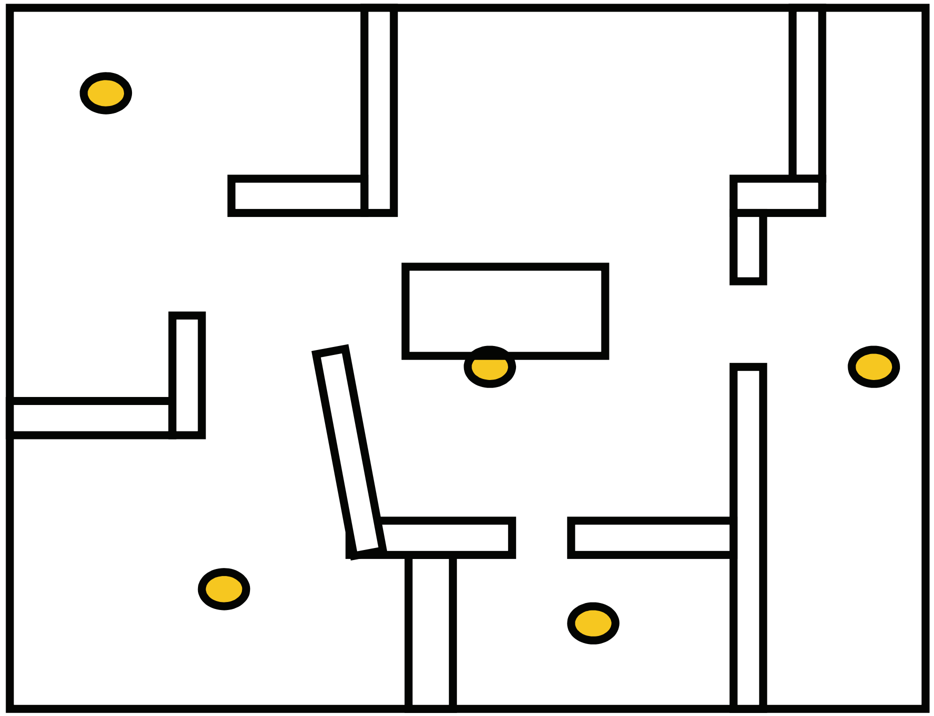
$$\sum_{i \in S_1} i = \sum_{i \in S_2} i$$

(partition into S_1 and S_2 such that the sum of S_1 is equal to that of S_2)

- e.g. S={1, 7, 10, 4, 6, 3, 8, 13}
  - S_1={1, 10, 4, 8, 3}
  - S_2={7, 6, 13}
- This problem is NP-complete.

Art gallery problem



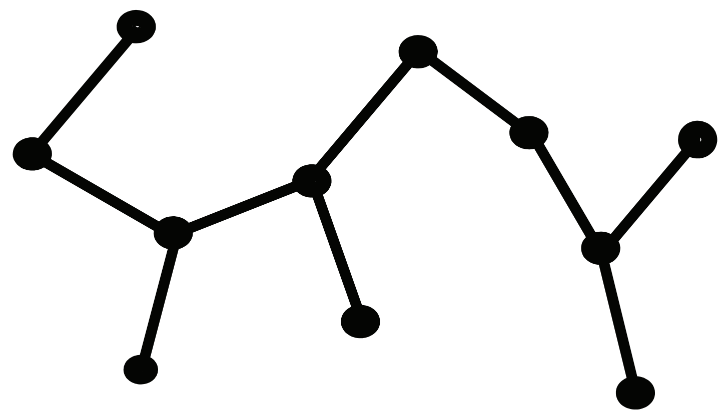


- Given: an art gallery

  Determine: min # of guards and their placements such that the entire art gallery can be monitored.
- NP-complete

Minimum spanning tree

- graph: greedy method
- geometry(on a plane): divide-and-conquer
- # of possible spanning trees for n points: n^{n-2} **(Cayley's formula)**
- $n=10 \rightarrow 10^8$, $n=100 \rightarrow 10^{196}$

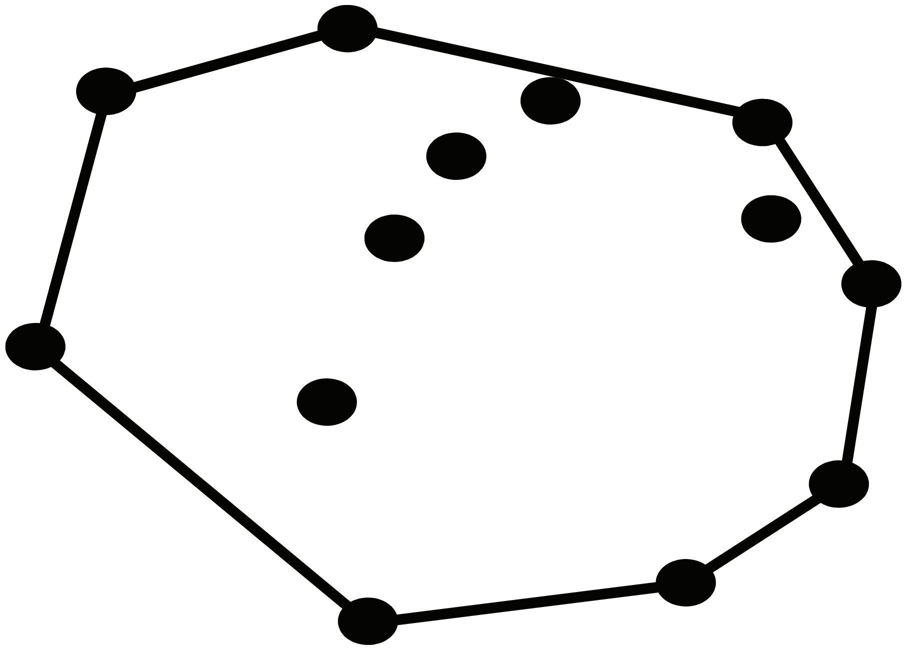


Cayley's Formula

https://www.youtube.com/watch?v=Ve447EOW8ww

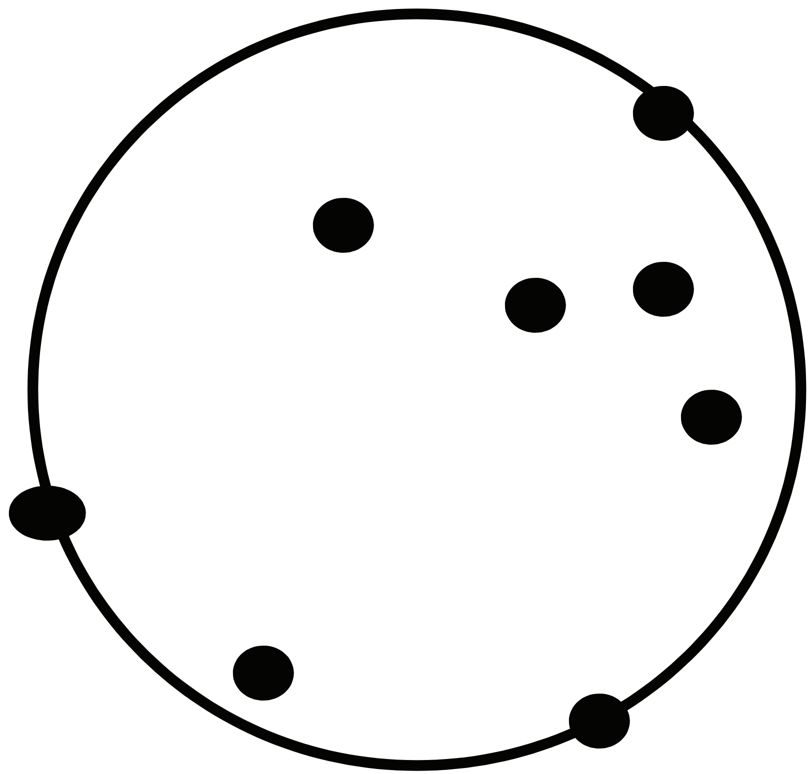
http://www.csie.ntu.edu.tw/~kmchao/tree07spr/counting.pdf

Convex hull



- Given a set of planar points, find a smallest convex polygon which contains all points.
- It is not obvious to find a convex hull by examining all possible solutions
- divide-and-conquer

One-center problem



- Given a set of planar points, find a smallest circle which contains all points.
- Prune-and-search