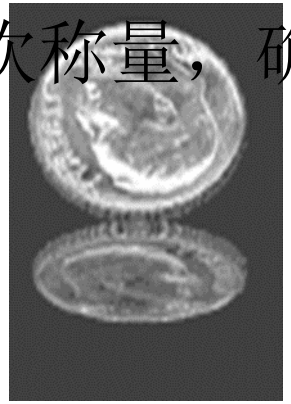
计算机问题求解 – 论题2–03 - 算法的效率

2014年03月04日

找假币

- 给你70个外观完全一样的金币,但是你知道其中有一个是假币,其重量比真币轻。给你一架没有砝码的天平，你可以在天平两边摆任意多个金币，比较他们的轻重。
- 请设计一种方法，通过若干次称量，确定哪一个是假币。



第一个解法稍稍改进

- 先将70个金币平均分两份，放到天平两边。

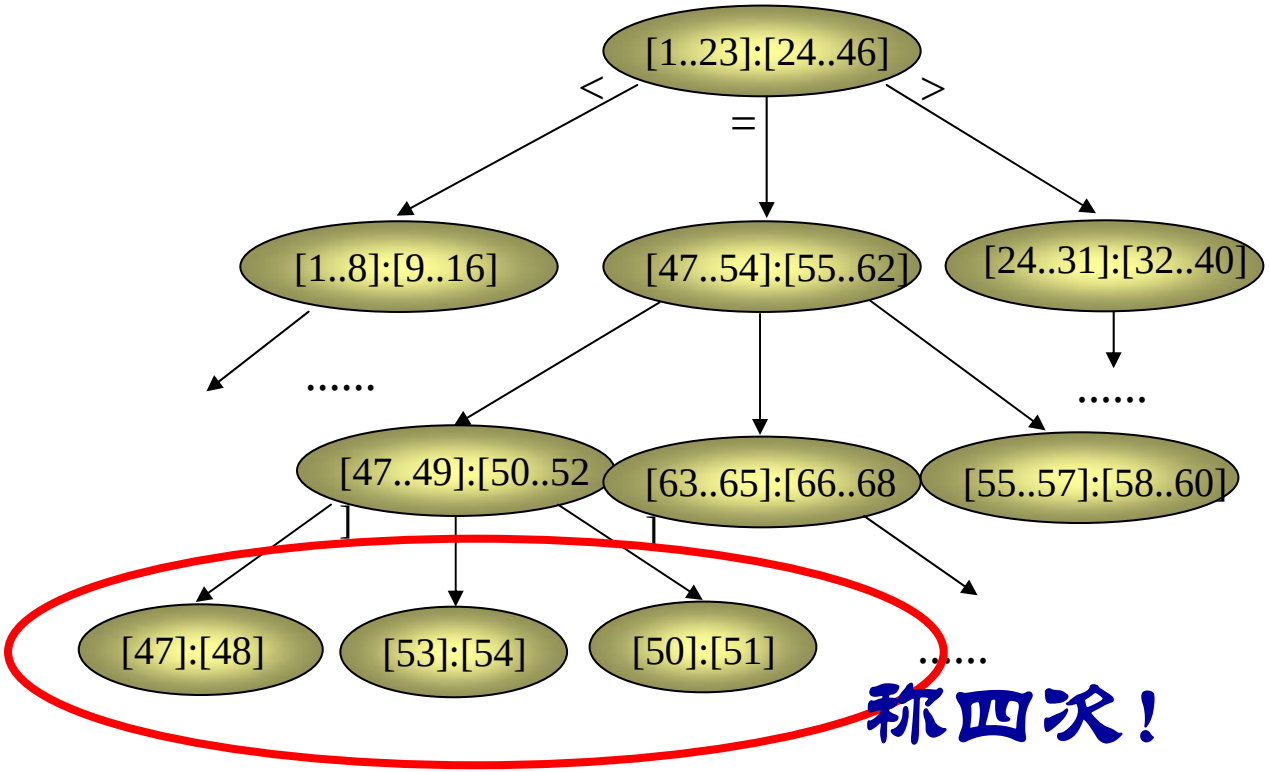
假币必在轻的那一侧的35个中。

- 将那35个中任意一个放一旁，其余再平分称量，每边17个。
- 若相等则旁边的是假币，否则将轻的一侧的17个中任意一个放旁边，其余的再平分再称量，每边8个。
- 若相等则旁边的是假币，否则将轻的一侧的8个平分再称量，每边4个。
- 将轻的一侧的4个再平分称量，每边2个。
- 将轻的一侧的2个再平分称量，每边个。轻的是假币。完成

总共称6次

但是，你还应该问自己：还能更好吗？

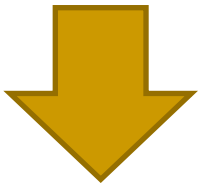
一个更好的办法


[1..23]:[24..46]
<
=
>
[1..8]:[9..16]
[47..54]:[55..62]
[24..31]:[32..40]
.....
.....
[47..49]:[50..52
[63..65]:[66..68
[55..57]:[58..60]
[47]:[48]
[53]:[54]
[50]:[51]
.....
称四次！

问题1：你如何理解这里的“优化”？算法级？程序级？

(2) **for** I **from** 1 **to** N **do:**

(2.1) $L(I) \leftarrow L(I) \times 100/MAX$



(1) compute the maximum score in MAX;

(2) $FACTOR \leftarrow 100/MAX$;

(3) **for** I **from** 1 **to** N **do:**

(3.1) $L(I) \leftarrow L(I) \times FACTOR$.

问题2：

你能说出如何用Linear Search算法搜索一个未排序的序列吗？书中给出的优化方法是什么？这种优化是量上的优化还是质上的优化？？

给定一个算法，什么样的优化算是质上的优化？

问题3：

“算法分析”主要是干什么？

问题4：

为什么我们不能用一个数值表示算法的“代价”？

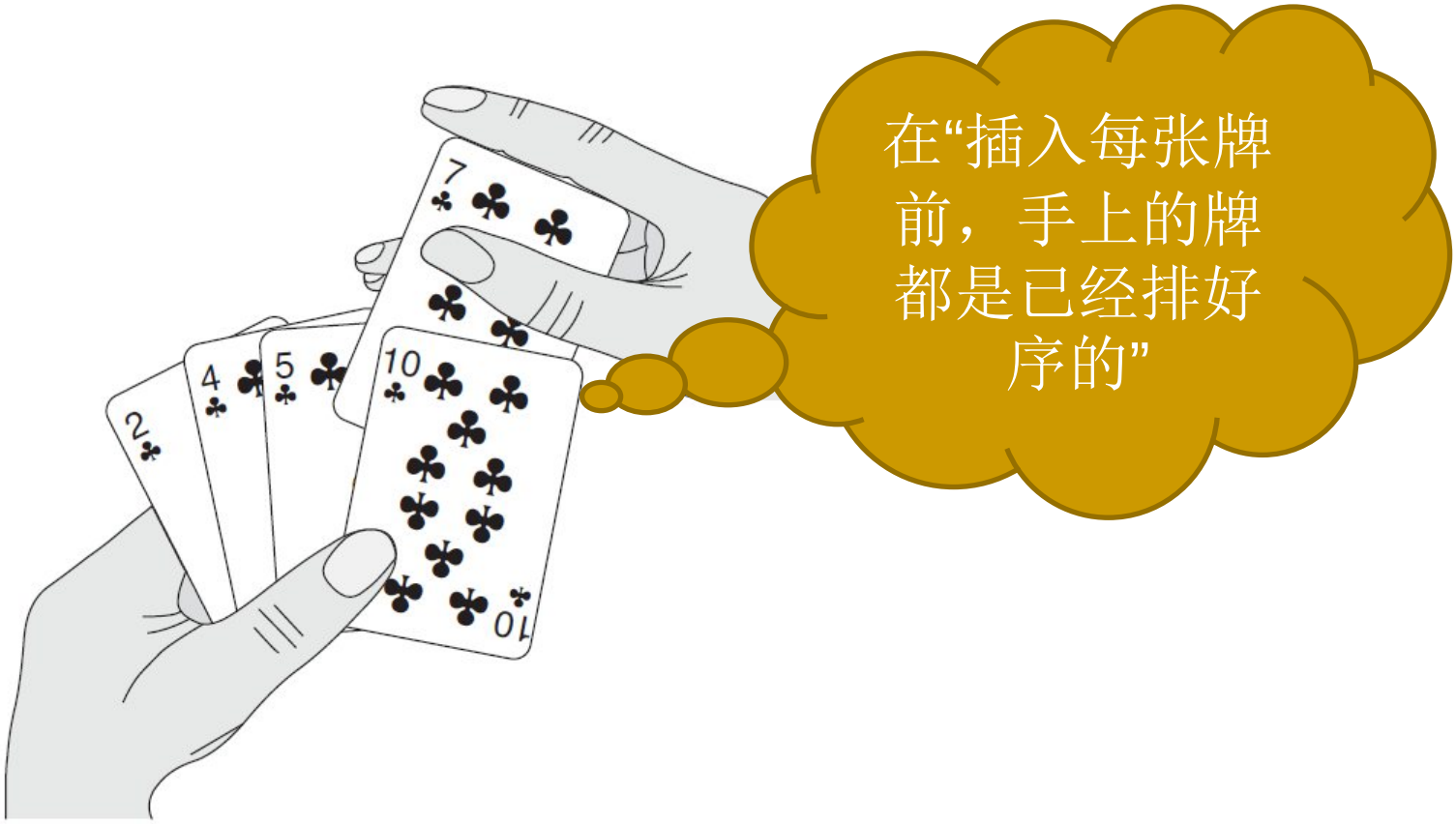
一个插入排序方法

Our first algorithm, insertion sort, solves the ***sorting problem*** introduced in Chapter 1:

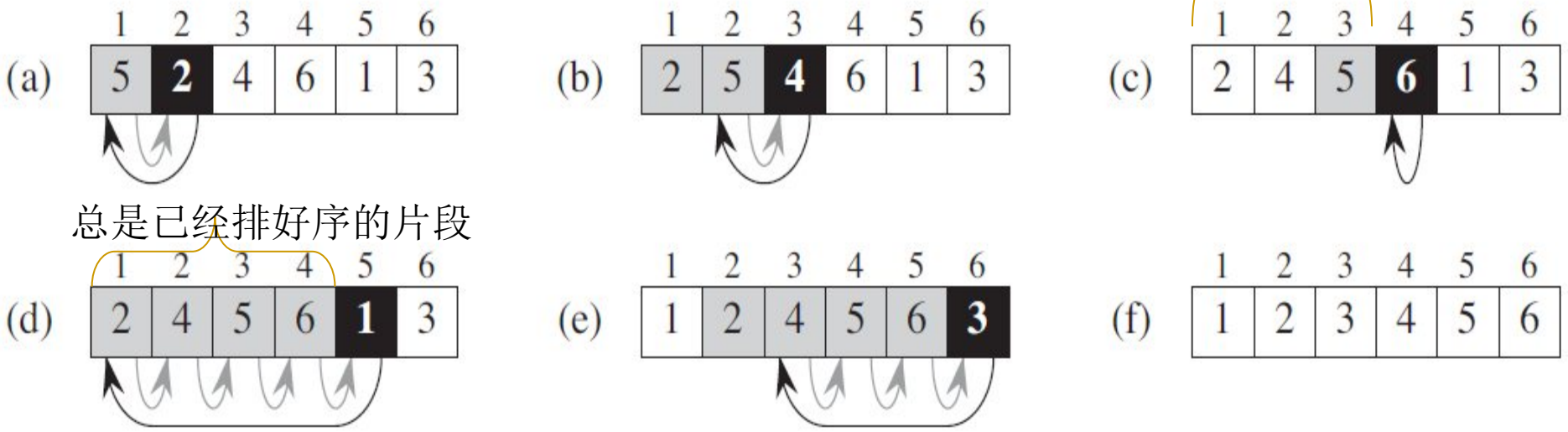
Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

插入法：



范例


总是已经排好序的片段
(a)
1 2 3 4 5 6
5 2 4 6 1 3
(b)
1 2 3 4 5 6
2 5 4 6 1 3
总是已经排好序的片段
(c)
1 2 3 4 5 6
2 4 5 6 1 3
总是已经排好序的片段
(d)
1 2 3 4 5 6
2 4 5 6 1 3
(e)
1 2 3 4 5 6
1 2 4 5 6 3
(f)
1 2 3 4 5 6
1 2 3 4 5 6

伪代码

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

循环不变量

now in sorted order. We state these properties of $A[1 \,..\, j-1]$ formally as a ***loop invariant***:

> At the start of each iteration of the **for** loop of lines 1–8, the subarray $A[1 \,..\, j-1]$ consists of the elements originally in $A[1 \,..\, j-1]$, but in sorted order.

用哪些数值来标定算法的性能？

- 时间性能（时间复杂度）
  - 算法在给定一个输入的情况下，要执行多少条指令

没错，数数字！

数数字！

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$

性能评估函数

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1)$$
$$+ c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) \; .$$

Best case

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \; .
\end{aligned}
$$

Worst case

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

and

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

(see Appendix A for a review of how to solve these summations), we find that in the worst case, the running time of INSERTION-SORT is

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8) \,. \end{aligned}$$

Average case

The "average case" is often roughly as bad as the worst case. Suppose that we randomly choose n numbers and apply insertion sort. How long does it take to determine where in subarray $A[1 \,..\, j-1]$ to insert element $A[j]$? On average, half the elements in $A[1 \,..\, j-1]$ are less than $A[j]$, and half the elements are greater. On average, therefore, we check half of the subarray $A[1 \,..\, j-1]$, and so t_j is about $j/2$. The resulting average-case running time turns out to be a quadratic function of the input size, just like the worst-case running time.

两个不同算法的优劣关键是”增长率”

Algorithm	1	2	3	4	
Time function(ms)	$33n$	$46n \lg n$	$13n^2$	$3.4n^3$	2^n
Input size(n)	Solution time				
10	0.00033 sec.	0.0015 sec.	0.0013 sec.	0.0034 sec.	0.001 sec.
100	0.0033 sec.	0.03 sec.	0.13 sec.	3.4 sec.	4×10^{16} yr.
1,000	0.033 sec.	0.45 sec.	13 sec.	0.94 hr.	
10,000	0.33 sec.	6.1 sec.	22 min.	39 days	
100,000	3.3 sec.	1.3 min.	1.5 days	108 yr.	
Time allowed	Maximum solvable input size (approx.)				
1 second	30,000	2,000	280	67	20
1 minute	1,800,000	82,000	2,200	260	26

通常情况下：

算法	1	2	3	4	5
Big-O	O(n)	O(nlgn)	$O(n^2)$	$O(n^3)$	$O(2^n)$

问题5：

什么是“Big-O”？

集合 “Big Oh”

- Definition
  - Giving $g: N \rightarrow R^+$, then $O(g)$ is the set of $f: \mathrm{N} \rightarrow \mathrm{R}^+$, such that for some $c \in \mathrm{R}^+$ and some $n_0 \in \mathrm{N}$, $f(n) \leq c g(n)$ for all $n \geq n_0$.
- A function $f \in O(g)$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c < \infty$
  - Note: c may be zero. In that case, $f \in o(g)$, “little Oh”

一个例子

- Let $f(n)=n^2$, $g(n)=n\lg n$, then:
  - $f \notin O(g)$, since $\lim_{n\to\infty} \frac{n^2}{n \lg n} = \lim_{n\to\infty} \frac{n}{\lg n} = \infty$
  - $g \in O(f)$, since $\lim_{n\to\infty} \frac{n \log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} = 0$
  - $n\lg n \in O(n^2)$

对数函数与幂函数

Which grows faster?

$\log_2 n$ or $\sqrt{n}$?

So, $\log_2 n \in O(\sqrt{n})$

$$\lim_{n\to\infty} \frac{\log_2 n}{\sqrt{n}} = \lim_{n\to\infty} \frac{\frac{\log_2 e}{n}}{\frac{1}{2\sqrt{n}}} = (2\log_2 e) \lim_{n\to\infty} \frac{\sqrt{n}}{n} = 0$$

一般性结论

- The log function grows more slowly than ***any*** positive power of n

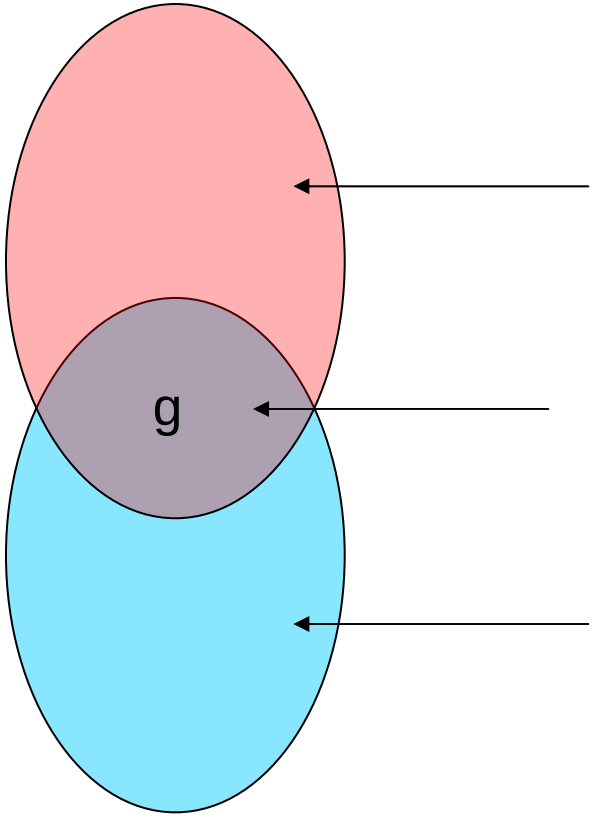
$$\lg n \in o(n^{\alpha}) \text{ for any } \alpha>0$$

By the way:

The power of n grows more slowly than any exponential function with base greater than 1

$$n^{k} \in o(c^{n}) \text{ for any } c>1$$

函数增长率的比较



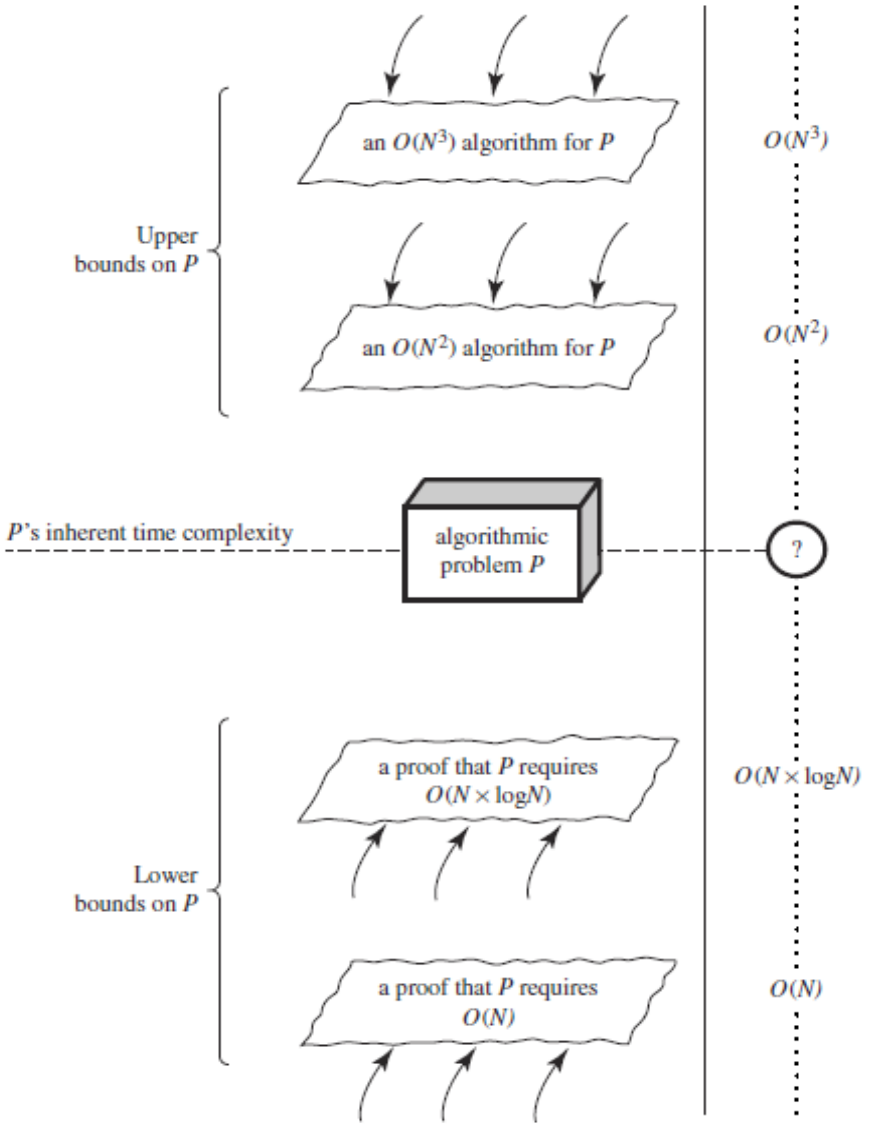
Ω(g)：该集合中任一函数的增长率不低于*g*的增长率（至少与*g*增长得一样快！）

Θ(g)：这里的函数与g具有“相同”的增长率。

O(g)：该集合中任一函数的增长率不高于*g*的增长率（最多与*g*增长得一样快！）

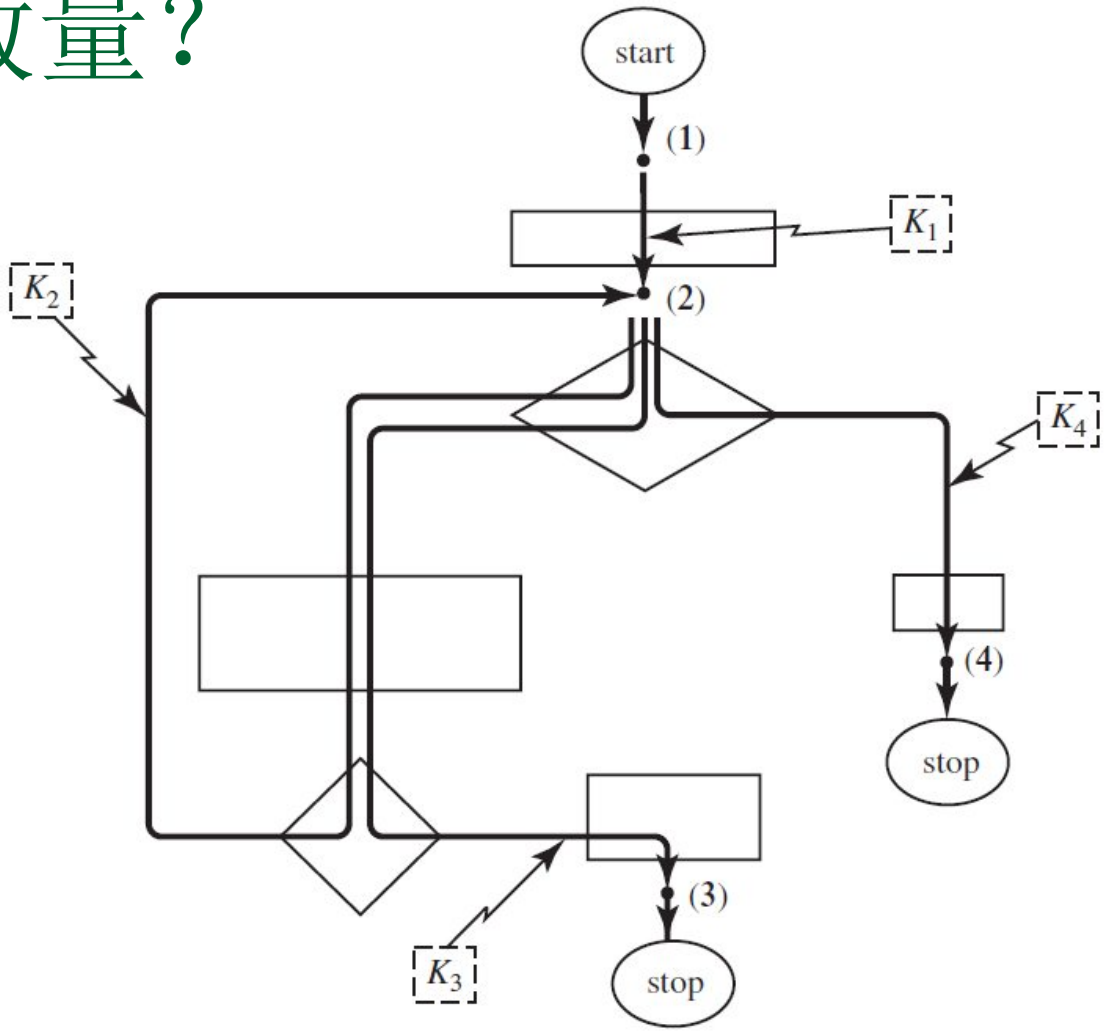
问题6：

给定一个算法，什么样的优化算是质上的优化？



问题7：什么是“Algorithmic Gap”？

问题8：在二分搜索算法的分析中，为什么我们可以只观察“比较”操作的数量？



问题8：

Big-O 的“Robustness”是什么意思？

In other words, as long as the basic set of allowed elementary instructions is agreed on, and as long as any shortcuts taken in high-level descriptions (such as that of Figure 6.1) do not hide unbounded iterations of such instructions, but merely represent finite clusters of them, big-*O* time estimates are *robust*.

问题9：

为什么有时候Big-O可能误导人？

We have known that : $\log n \in o(n^{0.0001})$

(since $\lim_{n\to\infty} \frac{\log n}{n^{\varepsilon}} = 0$ for any $\varepsilon > 0$)

However, which is larger : $\log n$ and n^{ε}, if $n = 10^{100}$?

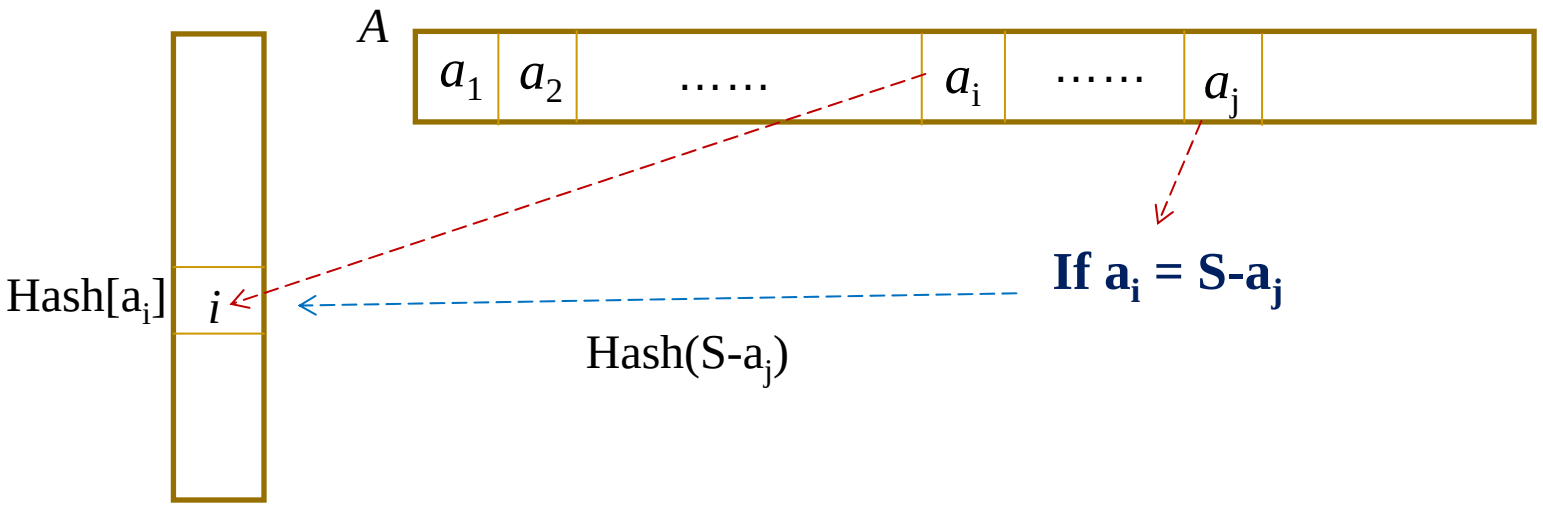
问题10：

从Linear Search到Binary Search, 收益是什么？需要付出什么代价？

考你一下：

Let *A* be an array of integers and *S* a target integer. Design an efficient algorithm for determining if there exist a pair of indices *i*,*j* such that *A*[*i*]+*A*[*j*]=*S*.

如果这里“efficient”是指“线性的”，你的答案满足要求吗？



Sleeping Tigers

(1) find the "lowest" point P_1;

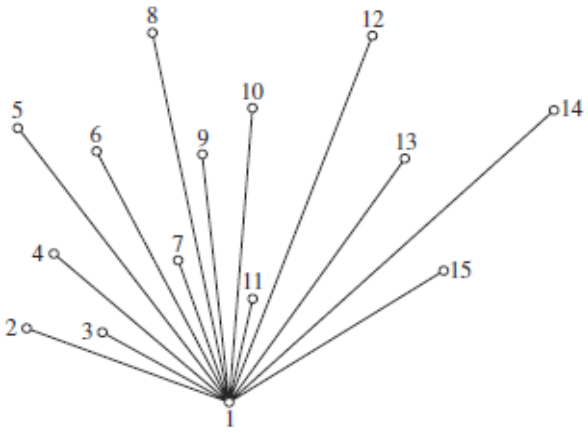
(2) sort the remaining points by the magnitude of the angle they form with the horizontal axis when connected with P_1, and let the resulting list be $P_2, \ldots, P_N$;

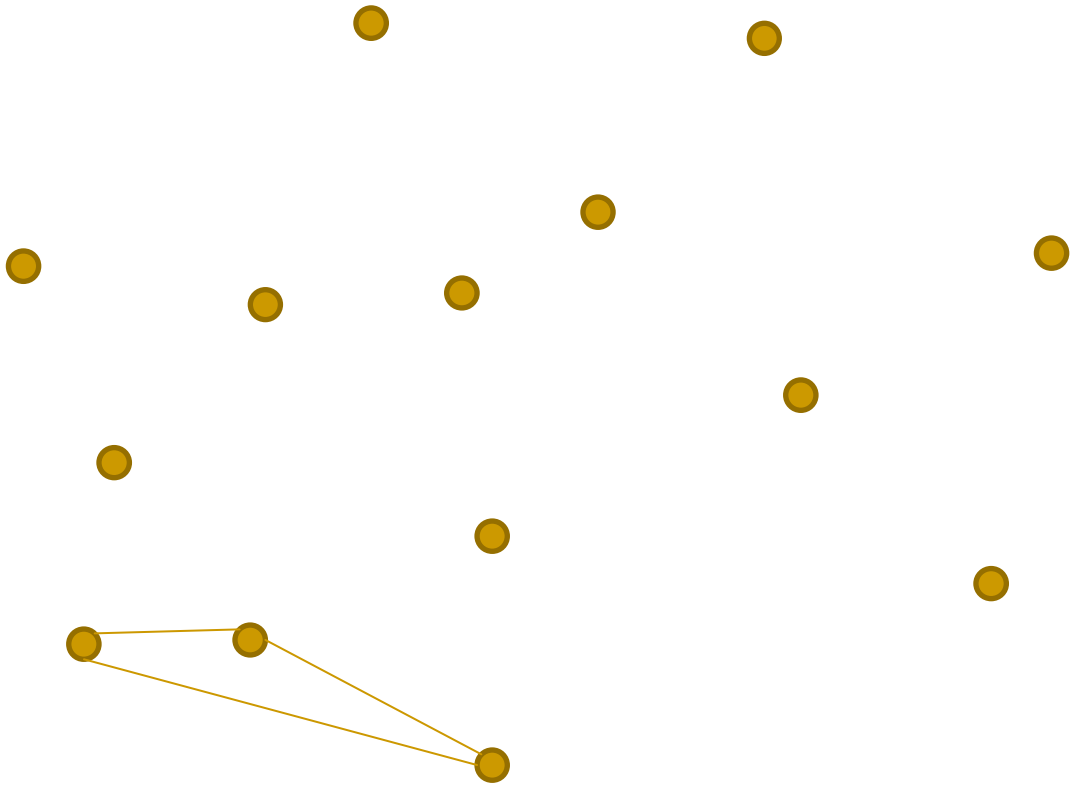
(3) start out with P_1 and P_2 in the current hull;

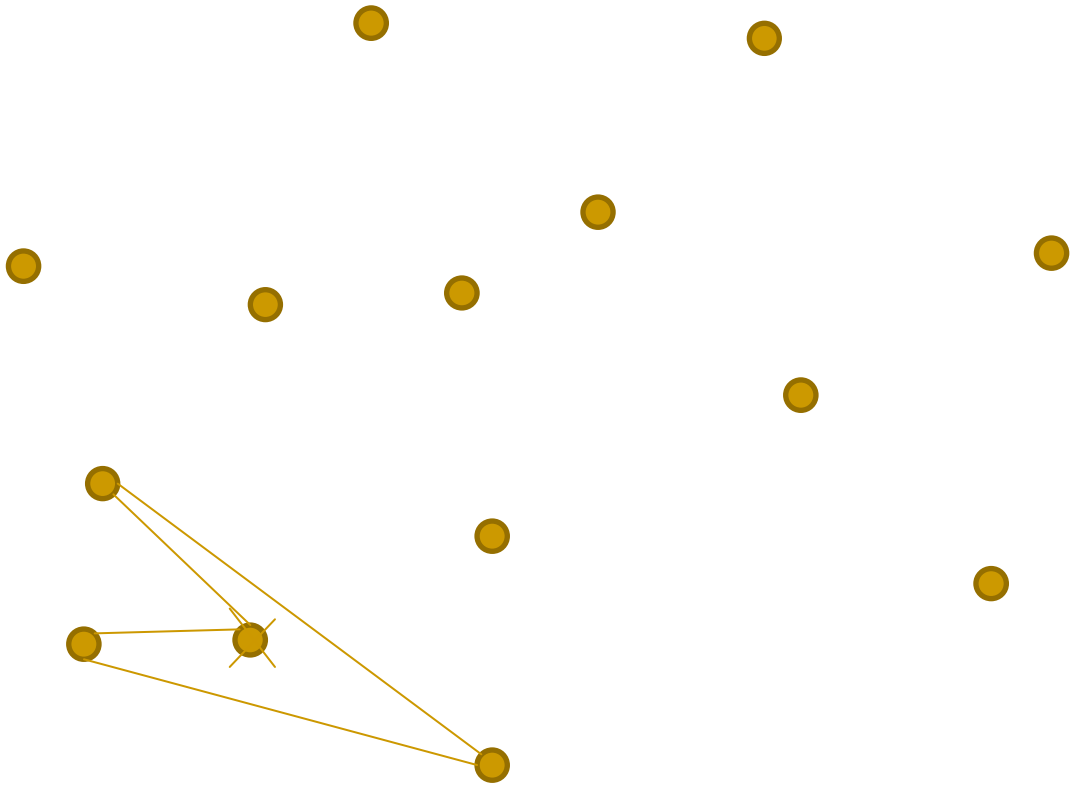
(4) for I from 3 to N do the following:

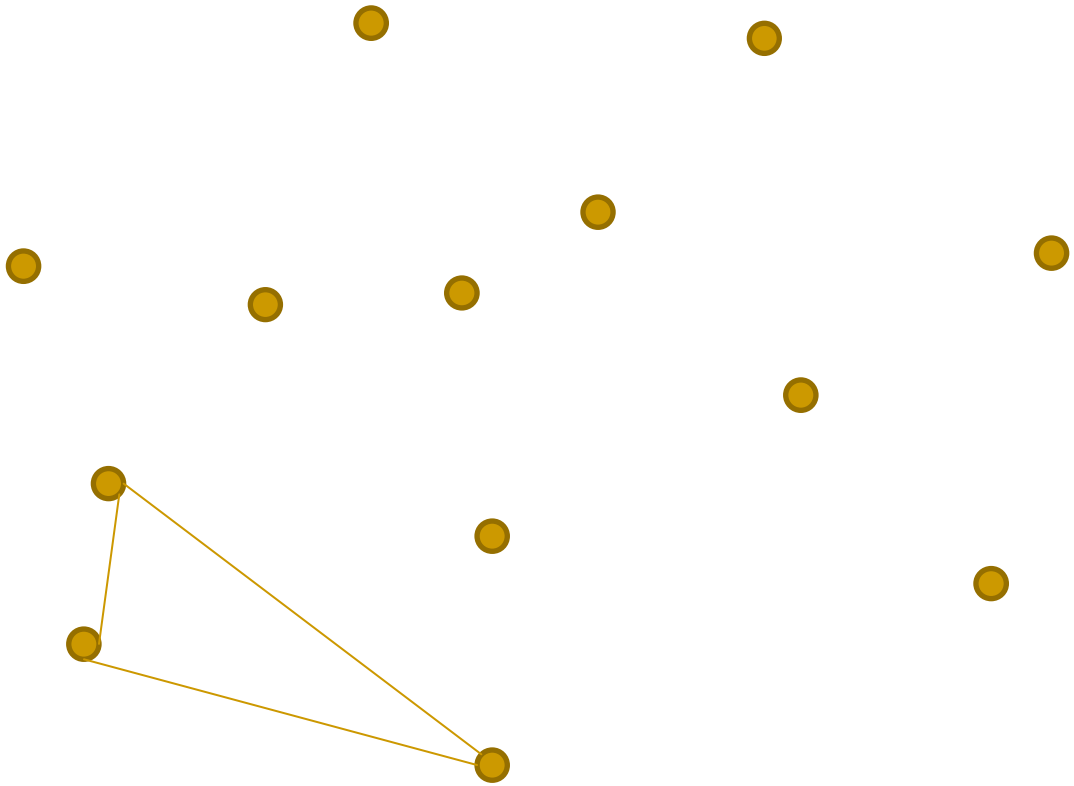
(4.1) add P_I tentatively to the current hull;

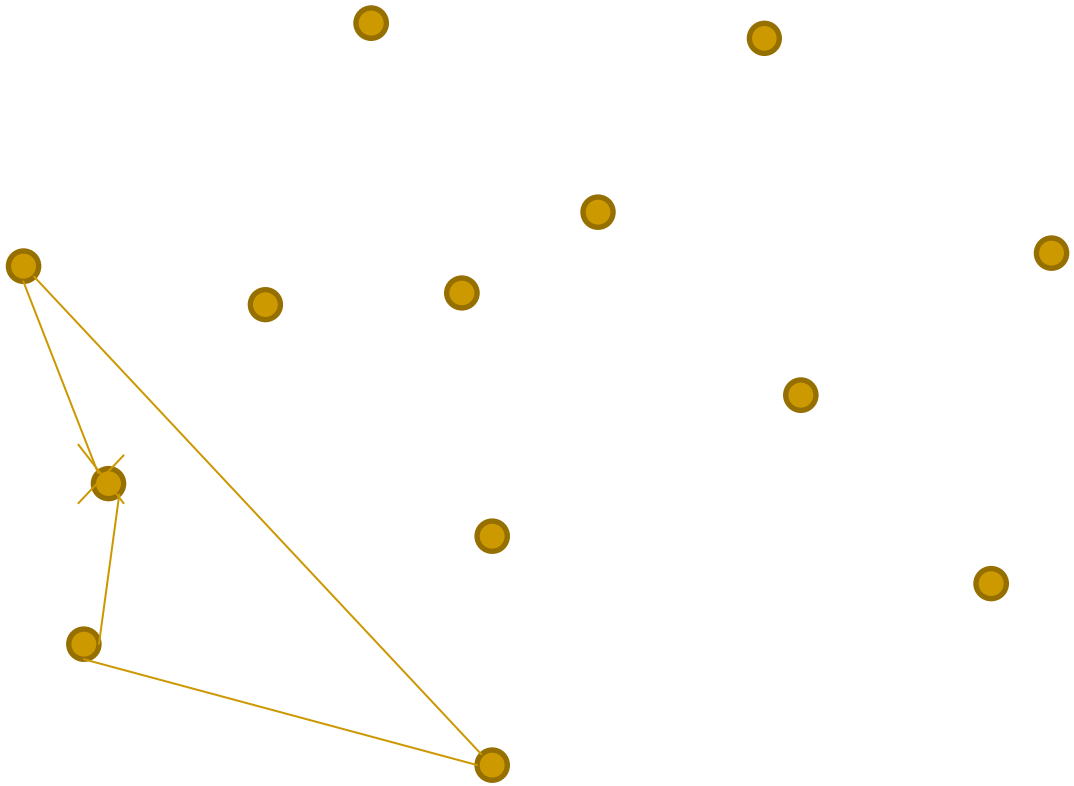
(4.2) work backwards through the current hull, eliminating a point P_J if the two points P_1 and P_I are on different sides of the line between P_{J-1} and P_J, and terminating this backwards scan when a P_J that does not need to be eliminated is encountered.

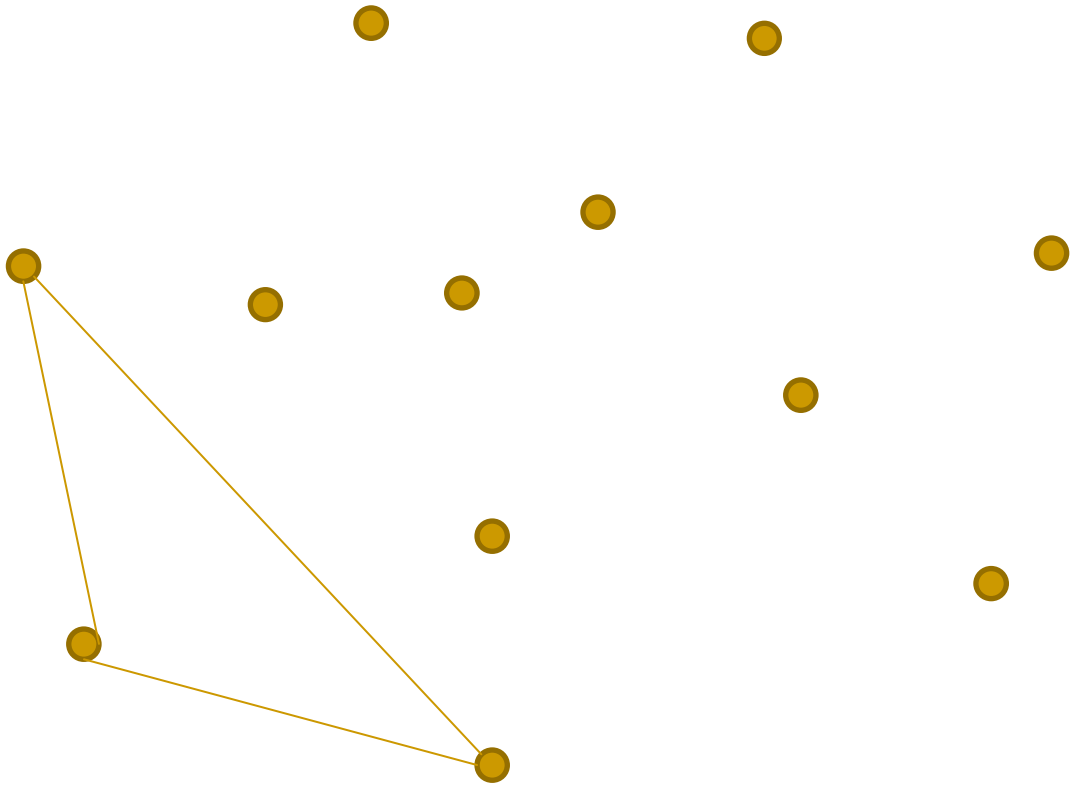


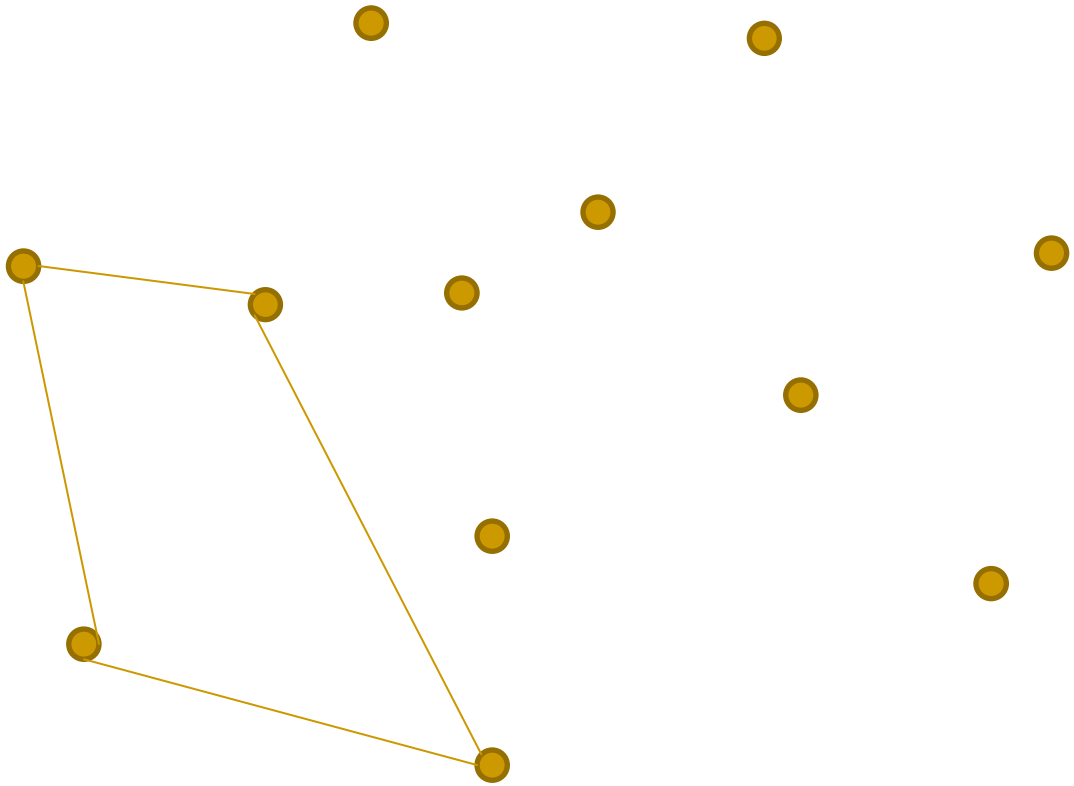


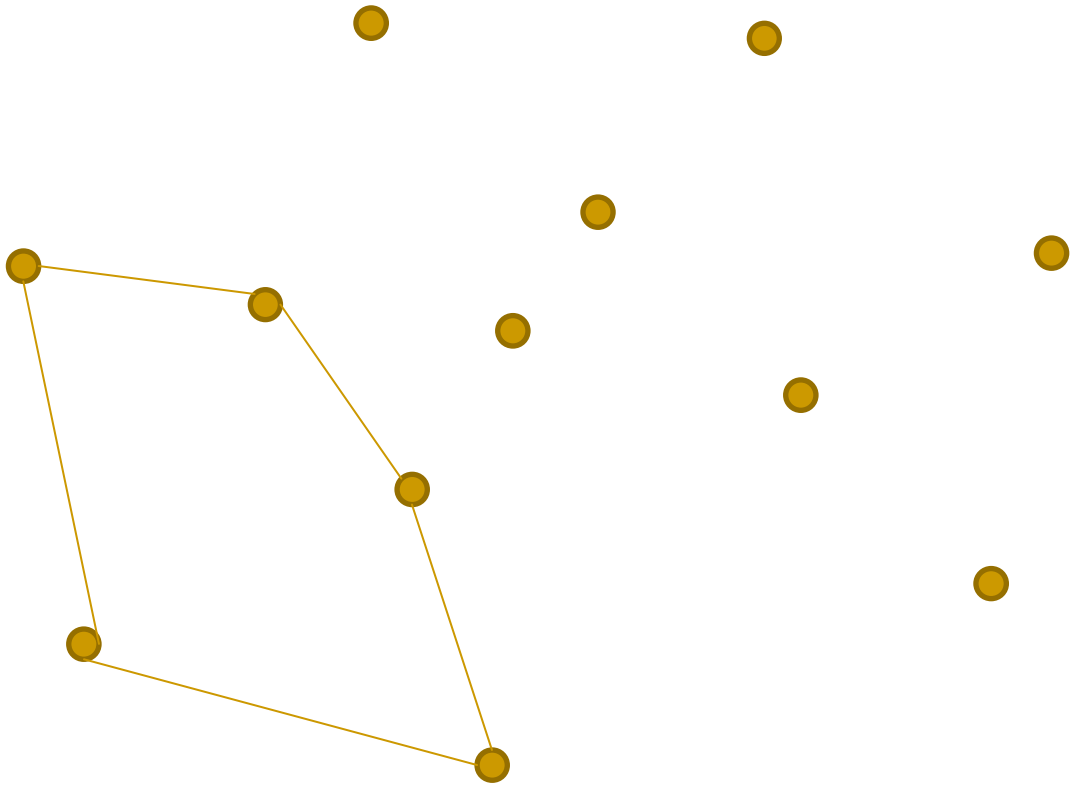


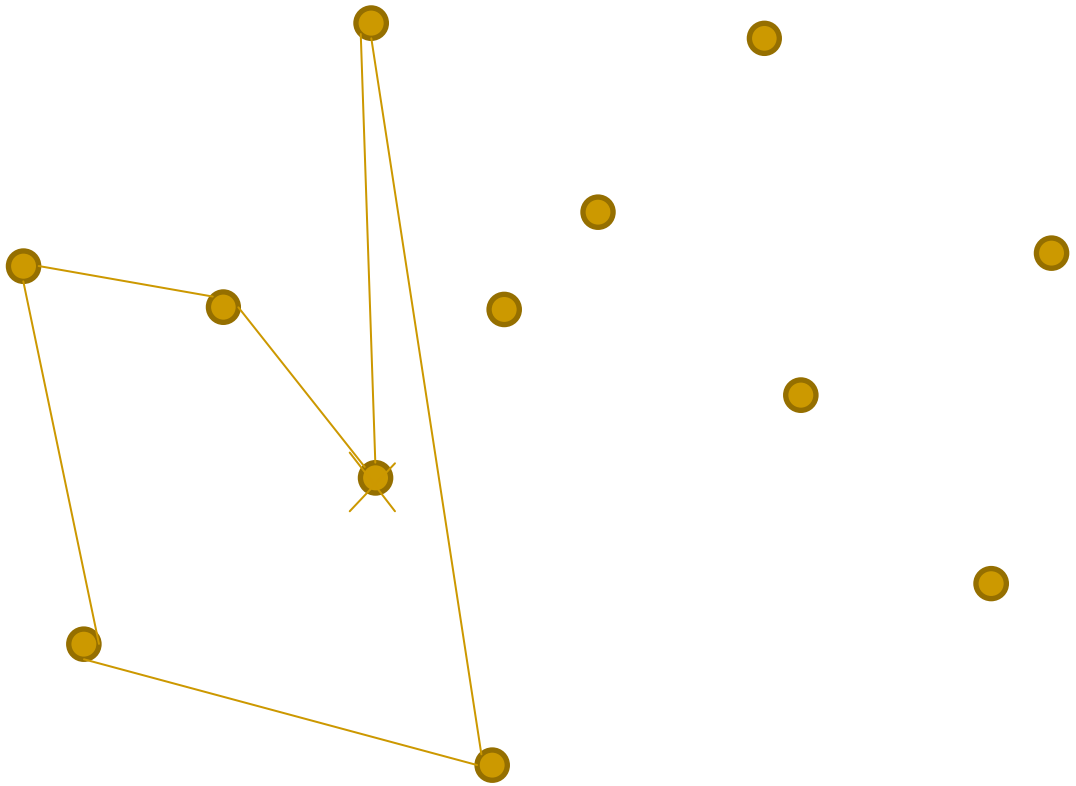


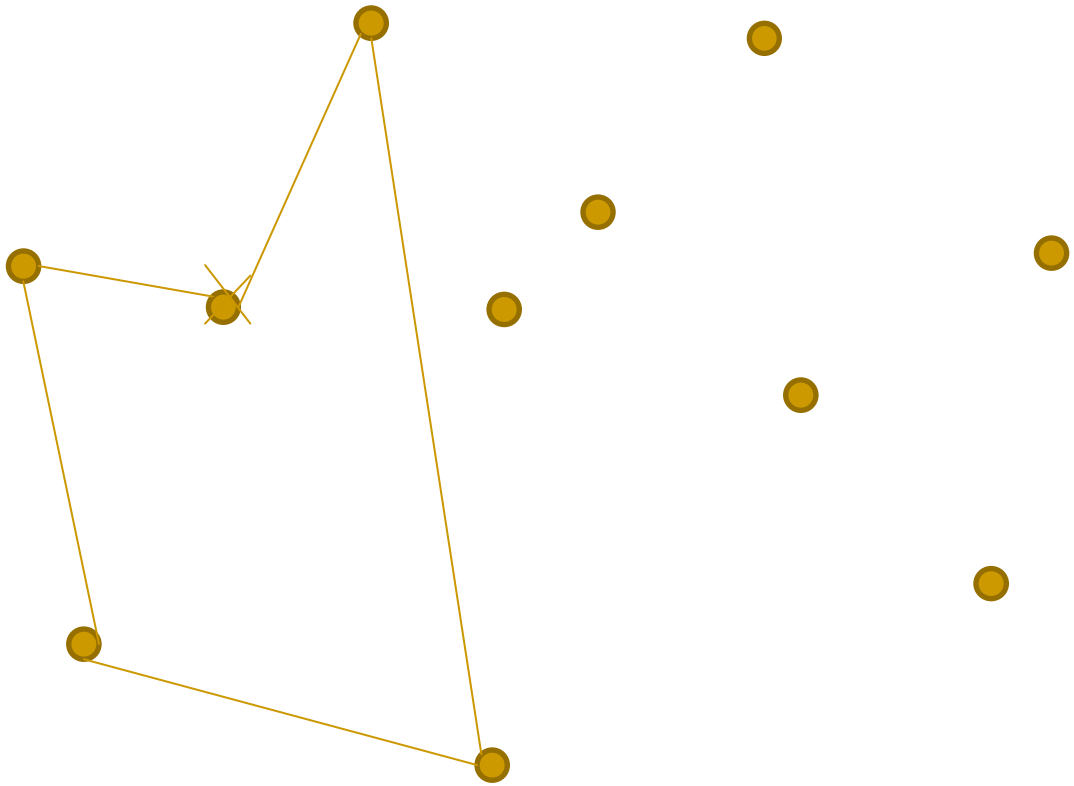












问题11：

你是否还记得有什么非排序算法，其主要代价就是排序的呢？

Step (1)	$O(N)$
Step (2)	$O(N \times \log N)$
Step (3)	$O(1)$
Step (4)	$O(N)$
Total	$O(N \times \log N)$

课外作业

- DH: pp.153-
  - 6.1 – 6.3
  - 6.10 , 6.11
  - 6.15, 6.16