



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis (H)

CS216

Prof. Shiqi Yu (于仕琪)

yusq@sustech.edu.cn

<http://faculty.sustech.edu.cn/yusq/>



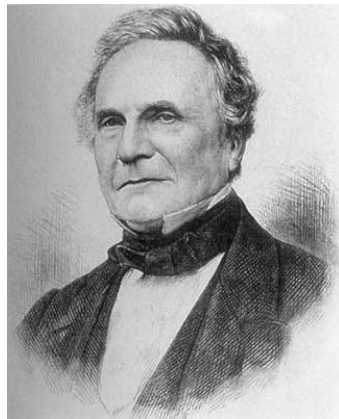
1. Computational Tractability

"For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing." - *Francis Sullivan*

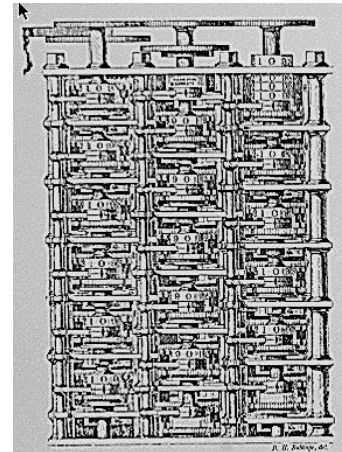


Computational Tractability

As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time? - *Charles Babbage*



Charles Babbage
(1864)

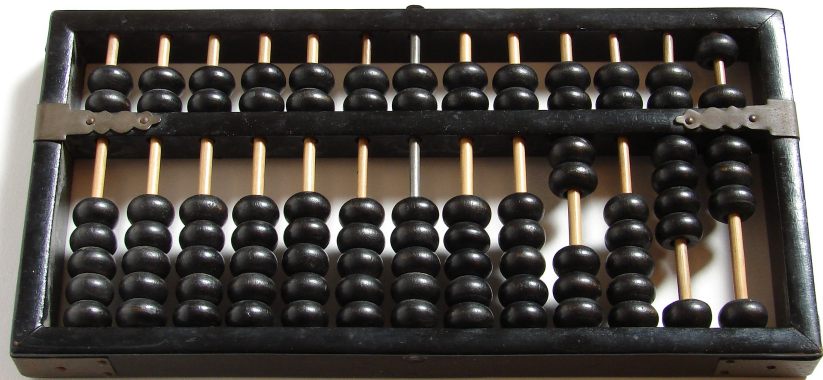


Analytic Engine
(schematic)

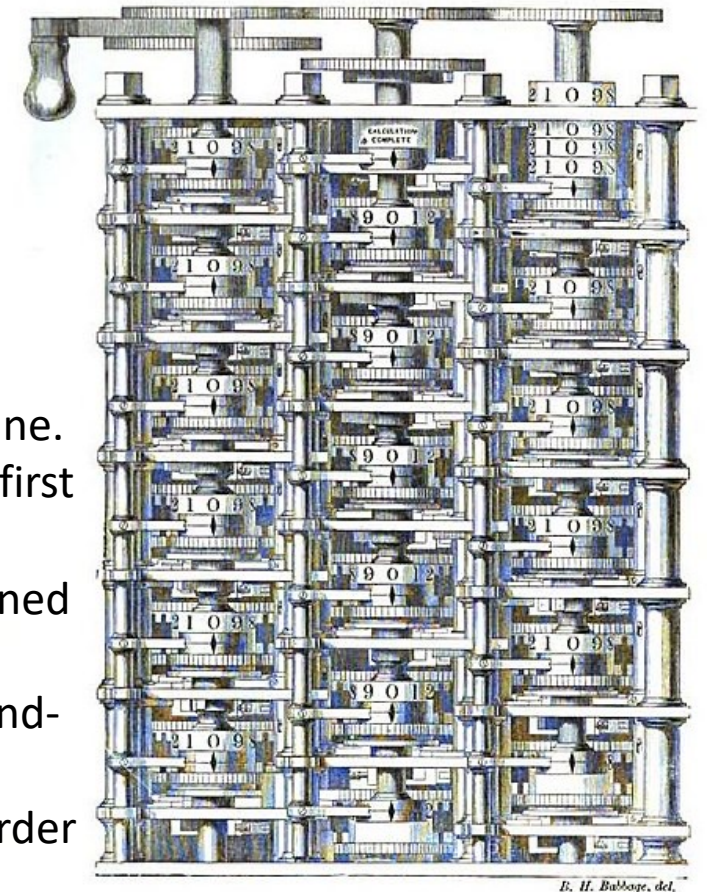


Computers

- A computer is a ~~digital-electronic~~ machine that can be programmed to carry out sequences of arithmetic or logical operations (computation) automatically.
- Components
 - Calculation capability
 - Storage
 - Instructions



A portion of Babbage's Difference engine. It was designed in the 1820s, and was first created by Charles Babbage, is an automatic mechanical calculator designed to tabulate polynomial functions. It operated on 6-digit numbers and second-order differences, and was tended to operate on 20-digit numbers and six-order differences.





Computers

- Boolean algebra by George Boole
- A master's thesis by Claude Elwood Shannon: Electrical applications of Boolean algebra could construct any logical numerical relationship.
 - Mechanical-> Electrical

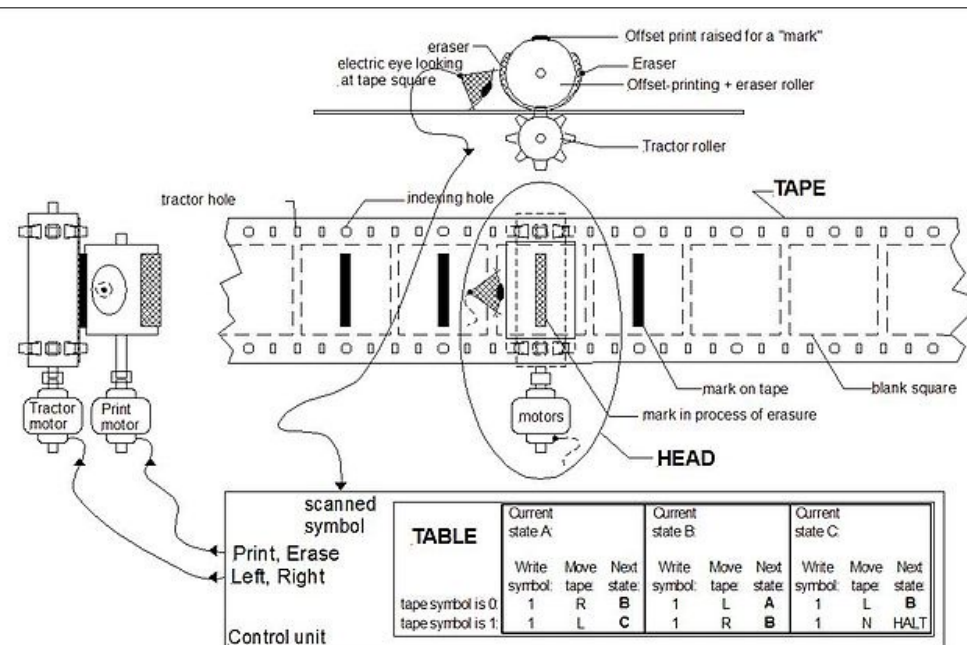


图灵机

- 艾伦图灵早期思考的三个根本问题

- 数学问题是否都有明确的答案。
- 如果有明确的答案，是否可以通过有限步的计算得到答案。
- 对于那些有可能在有限步计算出来的数学问题，能否有一种假想的机器，他们不断地运动，最后当它停下来时，那个数学问题就解决了。

专注于

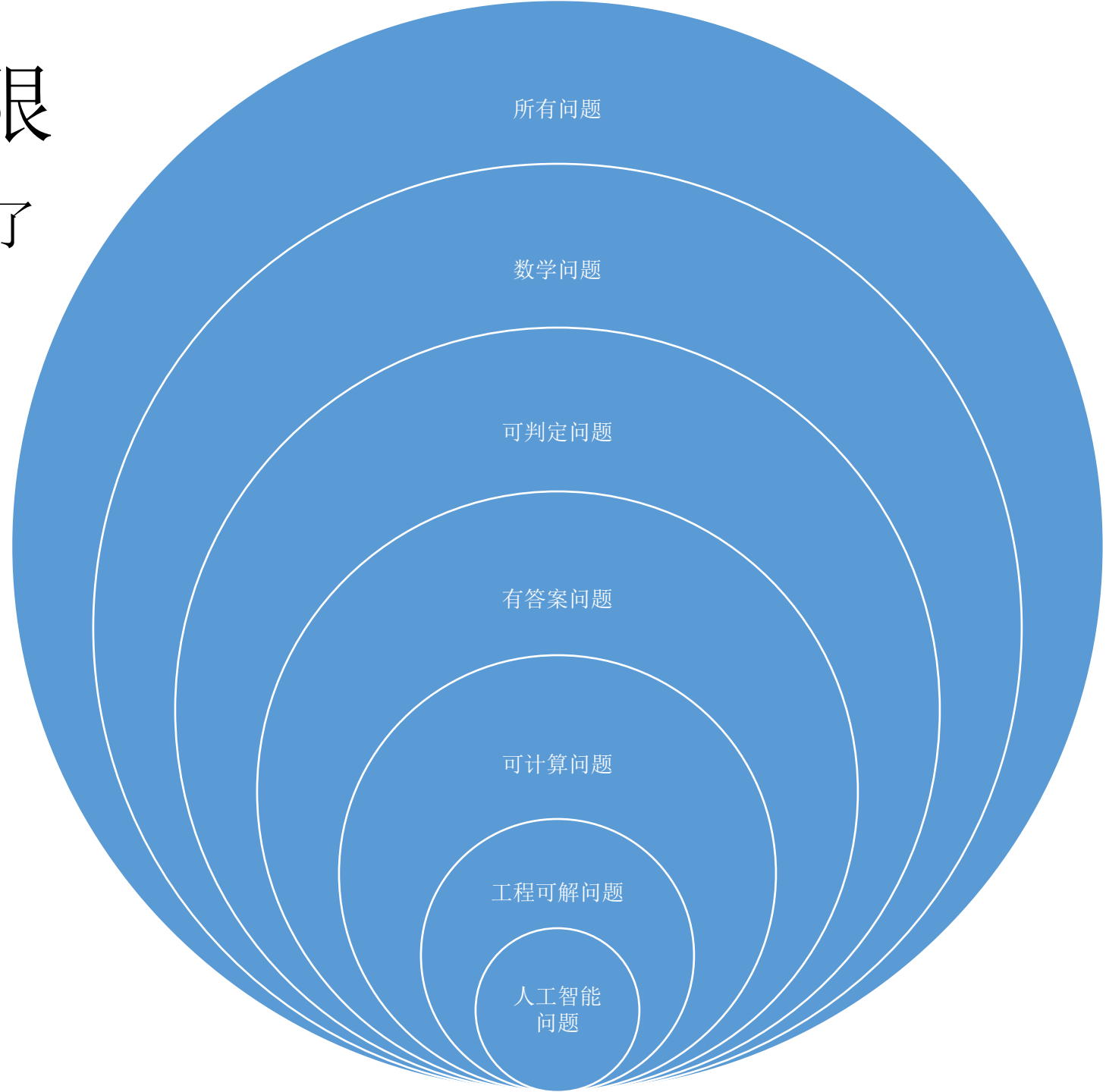


A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.



人工智能的极限

- David Hilbert & Kurt Gödel证明了世间很多问题不是数学问题
 - 完备性、一致性和有效公理化
- 数学问题
 - 不能判断是否存在答案
 - 能判断是否存在答案
- 可判定问题
 - 无答案问题
 - 有答案问题
- 有答案问题
- 可计算问题：图灵机可解
- 工程可解问题：计算复杂度度
- 人工智能问题





We human beings

- We are NOT good at perception on numbers
- Time
 - 1ms VS 1nm
 - 1 year
 - 40 years
 - 2.6×10^3 years: Confucius
 - 5.0×10^9 years: The earth
- Distance:
 - 1 meter
 - 1 light-year = 9.4607×10^{15} meters
 - The universe size = 93×10^9 light-years
 $= 8.798 \times 10^{26}$ meters




- All particles of an object on your desk? of the earth?
- All particles in the universe: 10^{79} - 10^{83}
- Combinations of a 20-word English sentence
 - Number of words: 10^5
 - $(10^5)^{20} = 10^{100}$

We need a ruler



Polynomial-Time

- **Brute force.** For many non-trivial problems, there is a natural brute force search algorithm that checks every possible solution.
 - Typically takes 2^N time or worse for inputs of size N .  $n!$ for stable matching with n men and n women
 - Unacceptable in practice.
- **Desirable scaling property.** When the input size doubles, the algorithm should only slow down by some constant factor C .

There exists constants $c > 0$ and $d > 0$ such that on every input of size N , its running time is bounded by $c N^d$ steps.

- **Def.** An algorithm is **poly-time** if the above scaling property holds.



choose $C = 2^d$



Worst-Case Analysis

- **Worst case running time.** Obtain bound on **largest possible** running time of algorithm on input of a given size N .
 - Generally captures efficiency in practice.
 - Draconian view, but hard to find effective alternative.
- **Average case running time.** Obtain bound on running time of algorithm on **random** input as a function of input size N .
 - Hard (or impossible) to accurately model real instances by random distributions.
 - Algorithm tuned for a certain distribution may perform poorly on other inputs.



Worst-Case Polynomial-Time

- **Def.** An algorithm is **efficient** if its running time is polynomial.
- **Justification:** **It really works in practice!**
 - Although $6.02 \times 10^{23} \times N^{20}$ is technically poly-time, it would be useless in practice.
 - In practice, the poly-time algorithms that people develop almost always have low constants and low exponents.
 - Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.
- **Exceptions.**
 - Some poly-time algorithms do have high constants and/or exponents, and are useless in practice.
 - Some exponential-time (or worse) algorithms are widely used because the worst-case instances seem to be rare.

simplex method
Unix grep



Why It Matters

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

2. Asymptotic Order of Growth



Asymptotic Order of Growth

- **$T(n)$** is a function, the worse-case running time of an algorithm on an input of size n .
- **Upper bounds** 上界. $T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \leq c \cdot f(n)$.
- **Lower bounds** 下界. $T(n)$ is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \geq c \cdot f(n)$.
- **Tight bounds** 紧确界. $T(n)$ is $\Theta(f(n))$ if $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$.
- **Ex:** $T(n) = 32n^2 + 17n + 32$.
 - $T(n)$ is $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, and $\Theta(n^2)$.
 - $T(n)$ is not $O(n)$, $\Omega(n^3)$, $\Theta(n)$, or $\Theta(n^3)$.



Notation

- **Slight abuse of notation.** $T(n) = O(f(n))$.
 - Not transitive:
 - ✓ $f(n) = 5n^3$; $g(n) = 3n^2$
 - ✓ $f(n) = O(n^3) = g(n)$
 - ✓ but $f(n) \neq g(n)$.
 - Better notation: $T(n) \in O(f(n))$.
- **Meaningless statement.** Any comparison-based sorting algorithm requires at least $O(n \log n)$ comparisons.
 - Statement doesn't "type-check."
 - Use Ω for lower bounds.



Properties

- **Transitivity.**

- If $f = O(g)$ and $g = O(h)$ then $f = O(h)$.
- If $f = \Omega(g)$ and $g = \Omega(h)$ then $f = \Omega(h)$.
- If $f = \Theta(g)$ and $g = \Theta(h)$ then $f = \Theta(h)$.

- **Additivity.**

- If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$.
- If $f = \Omega(h)$ and $g = \Omega(h)$ then $f + g = \Omega(h)$.
- If $f = \Theta(h)$ and $g = O(h)$ then $f + g = \Theta(h)$.



Asymptotic Bounds for Some Common Functions

- **Polynomials.** $a_0 + a_1n + \dots + a_dn^d$ is $\Theta(n^d)$ if $a_d > 0$.
- **Polynomial time.** Running time is $O(n^d)$ for some constant d independent of the input size n .
- **Logarithms.** $O(\log_a n) = O(\log_b n)$ for any constants $a, b > 0$.

can avoid specifying the base

- **Logarithms.** For every $x > 0$, $\log n = O(n^x)$.

log grows slower than every polynomial

- **Exponentials.** For every $r > 1$ and every $d > 0$, $n^d = O(r^n)$.

every exponential grows faster than every polynomial



3. A Survey of Common Running Times



Linear Time: $O(n)$

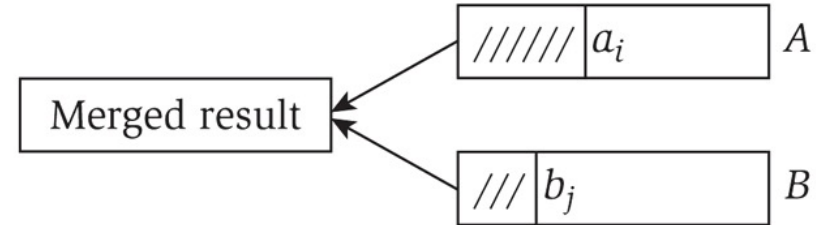
- **Linear time.** Running time is proportional to input size.
- **Computing the maximum.** Compute maximum of n numbers a_1, \dots, a_n .

```
max ← a1
for i = 2 to n {
    if (ai > max)
        max ← ai
}
```




Linear Time: $O(n)$

- **Merge.** Combine two sorted lists $\mathbf{A} = a_1, a_2, \dots, a_n$ with $\mathbf{B} = b_1, b_2, \dots, b_n$ into sorted whole.



```
i = 1, j = 1
while (both lists are nonempty) {
    if (a_i ≤ b_j) append a_i to output list and increment i
    else (a_i ≤ b_j) append b_j to output list and increment j
}
append remainder of nonempty list to output list
```

- **Claim.** Merging two lists of size n takes $O(n)$ time.
- **Pf.** After each comparison, the length of output list increases by 1.



$O(n \log n)$ Time

- $O(n \log n)$ time. Arises in divide-and-conquer algorithms.



also referred to as linearithmic time

- Sorting. Mergesort and heapsort are sorting algorithms that perform $O(n \log n)$ comparisons.
- Largest empty interval. Given n time-stamps x_1, \dots, x_n on which copies of a file arrive at a server, what is largest interval of time when no copies of the file arrive?
- $O(n \log n)$ solution. Sort the time-stamps. Scan the sorted list in order, identifying the maximum gap between successive time-stamps.



Quadratic Time: $O(n^2)$

- **Quadratic time.** Enumerate all pairs of elements.
- **Closest pair of points.** Given a list of n points in the plane $(x_1, y_1), \dots, (x_n, y_n)$, find the pair that is closest.
- **$O(n^2)$ solution.** Try all pairs of points.

```
min ←  $(x_1 - x_2)^2 + (y_1 - y_2)^2$ 
for i = 1 to n {
  for j = i+1 to n {
    d ←  $(x_i - x_j)^2 + (y_i - y_j)^2$ 
    if (d < min)
      min ← d
  }
}
```

← don't need to
take square roots

- Remark. $\Omega(n^2)$ seems inevitable, but this is just an illusion.

← see chapter 5



Cubic Time: $O(n^3)$

- **Cubic time.** Enumerate all triples of elements.
- **Set disjointness.** Given n sets S_1, \dots, S_n each of which is a subset of $1, 2, \dots, n$, is there some pair of these which are disjoint?
- **$O(n^3)$ solution.** For each pairs of sets, determine if they are disjoint.

```
foreach set  $S_i$  {  
    foreach other set  $S_j$  {  
        foreach element  $p$  of  $S_i$  {  
            determine whether  $p$  also belongs to  $S_j$   
        }  
        if (no element of  $S_i$  belongs to  $S_j$ )  
            report that  $S_i$  and  $S_j$  are disjoint  
    }  
}
```



Polynomial Time: $O(n^k)$ Time

- **Independent set of size k .** Given a graph, are there k nodes such that no two are joined by an edge?
- **$O(n^k)$ solution.** Enumerate all subsets of k nodes.

k is a constant

```
foreach subset S of k nodes {  
    check whether S is an independent set  
    if (S is an independent set)  
        report S is an independent set  
}
```

- Check whether S is an independent set = $O(k^2)$.
- Number of k element subsets =
- $O(k^2 n^k / k!) = O(n^k)$.

poly-time for $k=17$,
but not practical



Exponential Time

- **Independent set.** Given a graph, what is maximum size of an independent set?
- **$O(n^2 2^n)$ solution.** Enumerate all subsets.

```
S* ←  $\phi$ 
foreach subset S of nodes {
    check whether S is an independent set
    if (S is largest independent set seen so far)
        update S* ← S
}
```




$n!$

- The function $n!$ grows even more rapidly than 2^n



Sublinear Time: $O(\log n)$

- Binary search algorithm: search a given number in a sorted array.
- Reach to the end at $k = \log_2 n$ steps