Data Structures and Algorithms (1)

Instructor: Ming Zhang

Textbook Authors: Ming Zhang, Tengjiao Wang and Haiyan Zhao

Higher Education Press, 2008.6 (the "Eleventh Five-Year" national planning textbook)

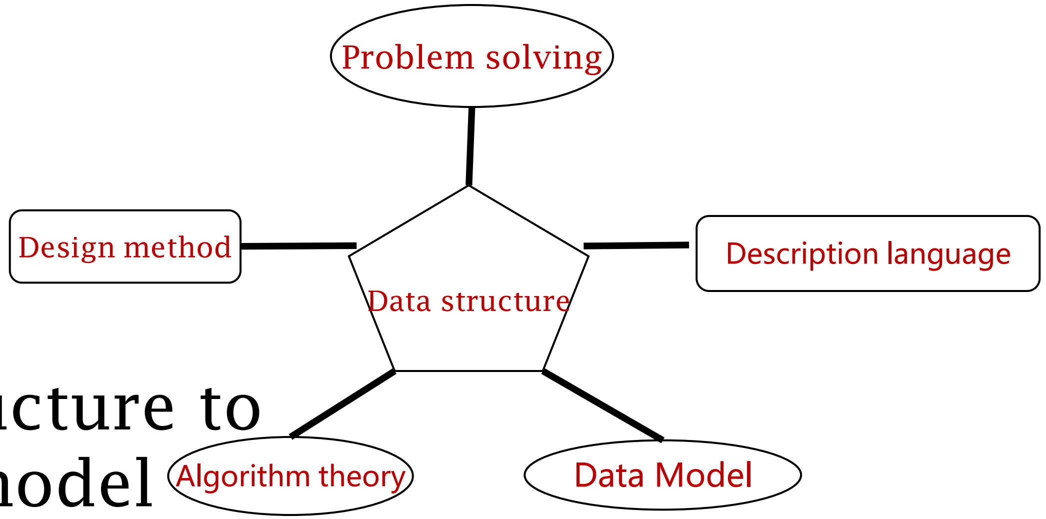
https://courses.edx.org/courses/PekingX/04830050x/2T2014/

Chapter 1 Overview

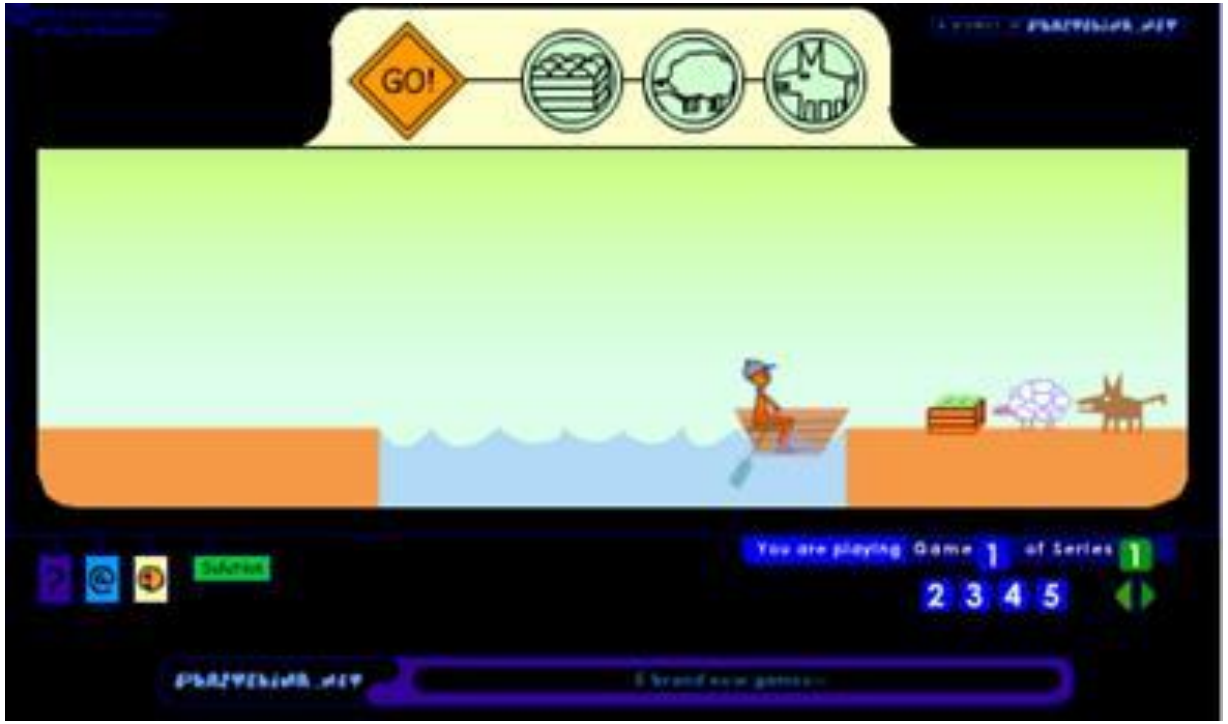
- Problem solving
- Data structures and abstract data types
- The properties and categories of algorithms
- Evaluating the efficiency of the algorithms

1.1 Problem solving

- Goal of writing computer programs ?
  - To solve practical problems
- Problem Abstraction
  - Analyze requirements and build a problem model
- Data Abstraction
  - Determine an appropriate data structure to represent a certain mathematical model
- Algorithm Abstraction
  - Design suitable algorithms for the data model
- Data structures + Algorithms => Programs
  - Simulate and solve practical problems



Farmer Crosses River Puzzle



cabbage

↓

sheep

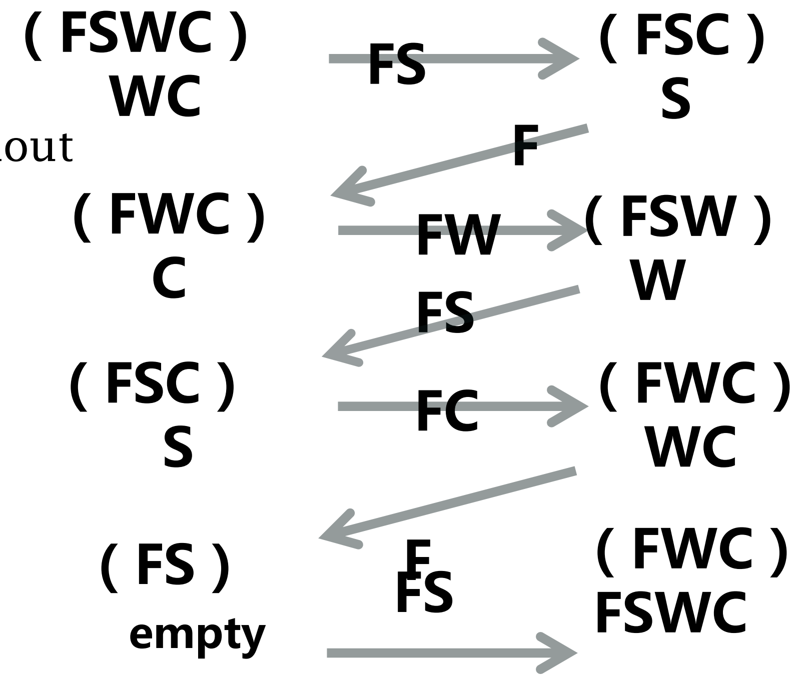
↓

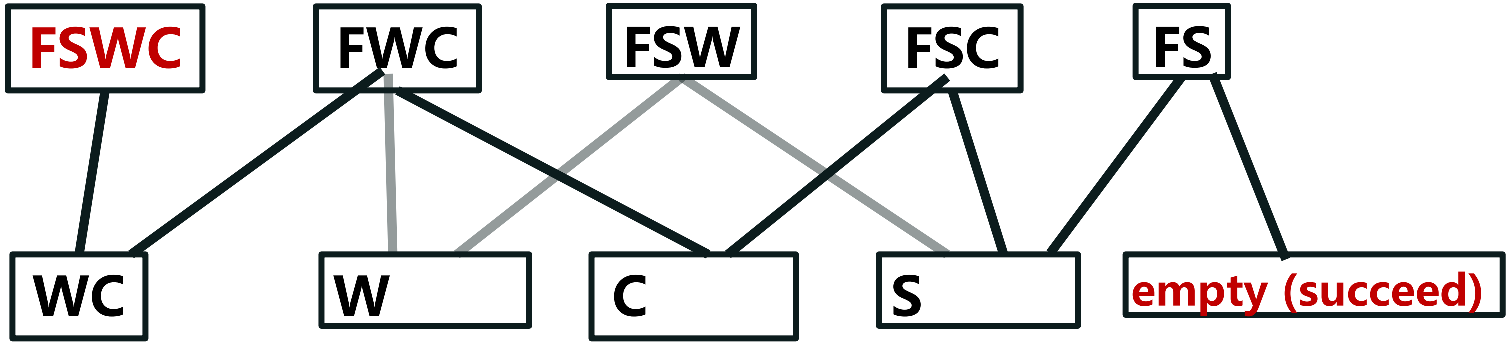
wolf

1.1 Problem solving

Farmer Crosses River Puzzle

- **Problem abstraction**: FSWC crossing over the river
  - Only the farmer can row the boat
  - There are only two seats on the boat including the farmer
  - "Wolf and sheep", "sheep and cabbages" can not stay along without the accompany of the farmer
- **Data abstraction**: graph model
  - Unreasonable state : WS、FC、SC、FW、WSC、F
  - The vertex represents the "original bank status"(10 states, including "empty")
  - edge : state transition as the result of a reasonable operation (cross over the river)





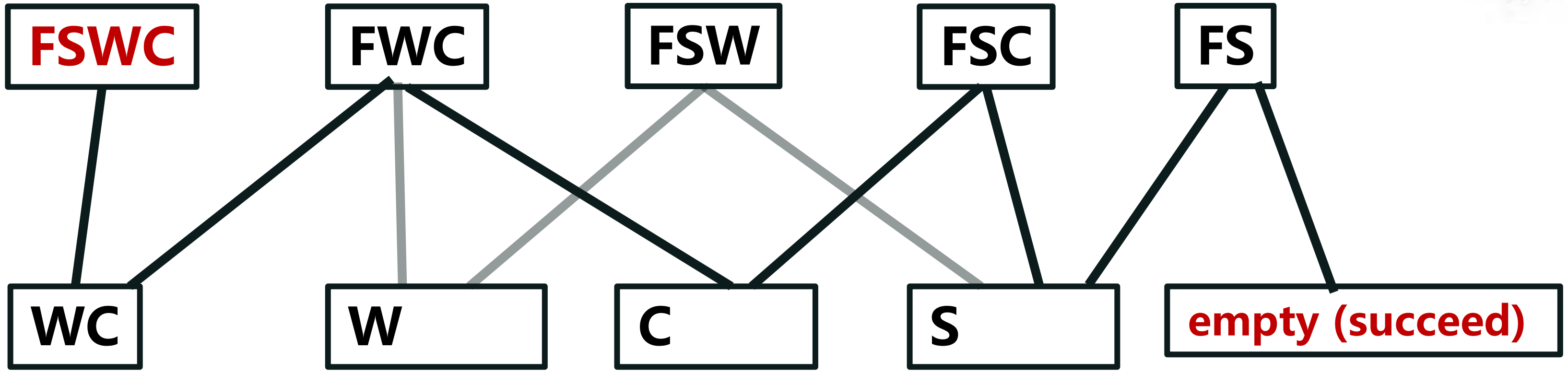
Farmer is abbreviated as F
Sheep is abbreviated as S
Wolf is abbreviated as W
cabbage is abbreviated as C

1.1 Problem solving

Farmer Crosses River Puzzle

- Data structure
  - Adjacency matrix
- Algorithm abstraction :
  - The shortest path

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$



Farmer is abbreviated as F
Sheep is abbreviated as S
Wolf is abbreviated as W
cabbage is abbreviated as C

Questions : process of problem solving

- Farmer Crosses River Puzzle —— The shortest path model
  - Problem abstraction ?
  - Data abstraction?
  - Algorithm abstraction ?
  - You may write programs to achieve it.

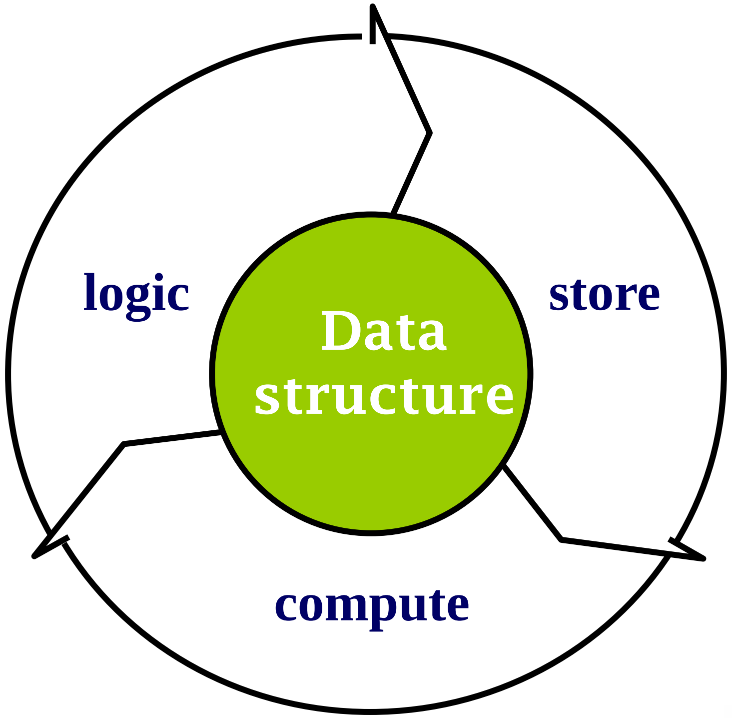
- Any other model ?

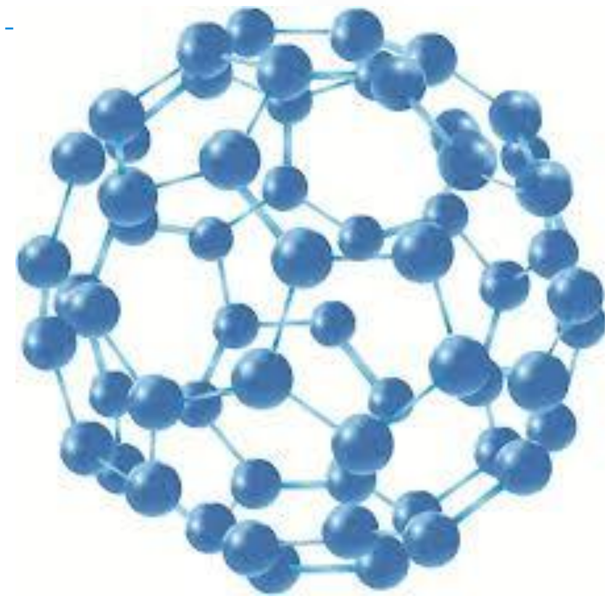
Chapter 1 Overview

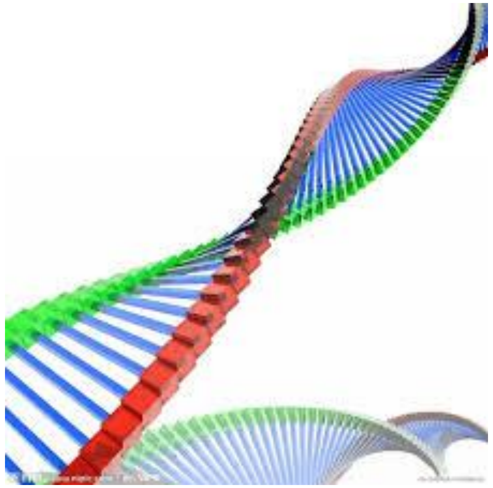
- Problem solving
- Data structures and abstract data types
- The properties and categories of algorithms
- Evaluating the efficiency of the algorithms

1.2 What is data structure

- **Structure: entity + relation**
- **Data structure :**
  - Data organized according to logical relationship
  - Stored in computer according to a certain storage method
  - A set of operations are defined on these data

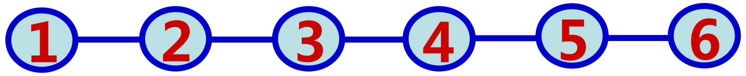




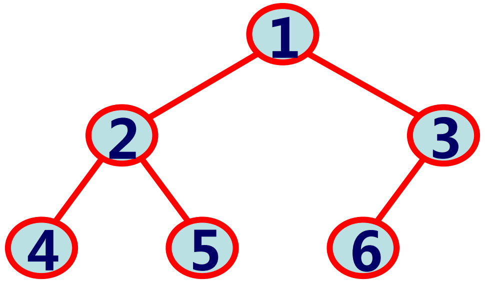


Logical organization of data structure

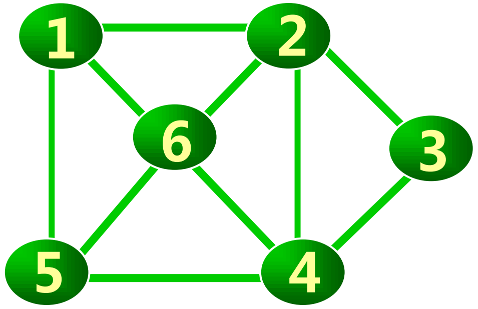
- Linear Structure



  - Linear lists (list , stack , queue , string, etc.)
- Nonlinear Structure



  - Trees (binary tree , Huffman tree , binary search tree etc)
  - Graphs (directed graph , undirected graph etc)



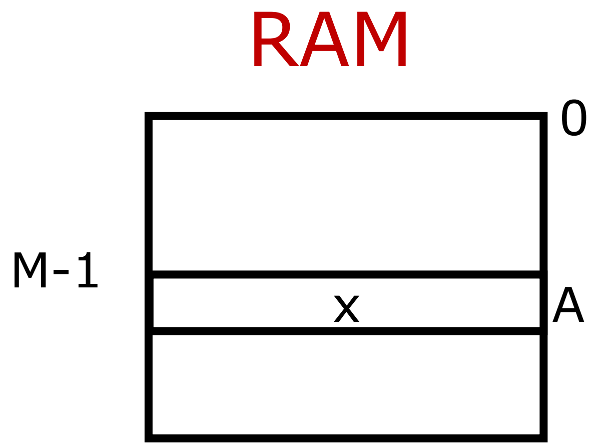
- Graph $\supseteq$ tree $\supseteq$ binary tree $\supseteq$ linear list

Storage structure of data

- **Mapping from logical structure to the physical storage space**

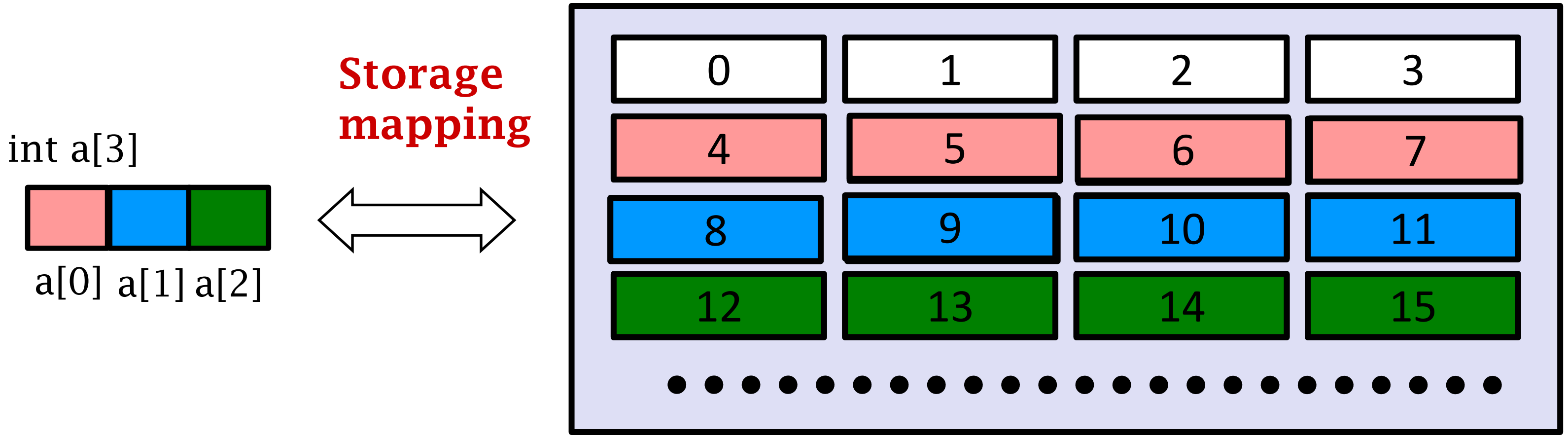
Main memory (RAM)

- Coded in non negative integer address , set of adjacent unit
- The basic unit is the byte
- The time required to access different addresses are basically the same (random access)



Storage structure of data

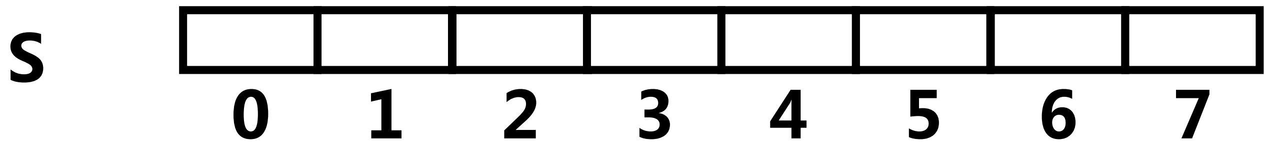
- For logical structure (K , r) , in which r∈R
  - For the node set K, establish a mapping from K to M memory unit : K→M , for every node j∈K , it corresponds to a unique continuous storage area C in M



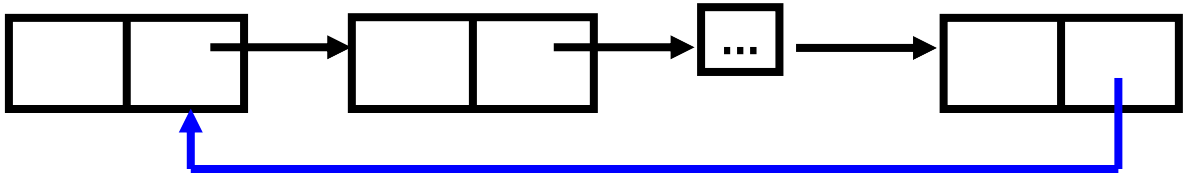
Main memory

Storage structure of data

- Relation tuple (j_1 , j_2) $\in$ r
  (j_1 , $j_2 \in$ K are nodes)
  - Sequence : storage units of data are adjacent



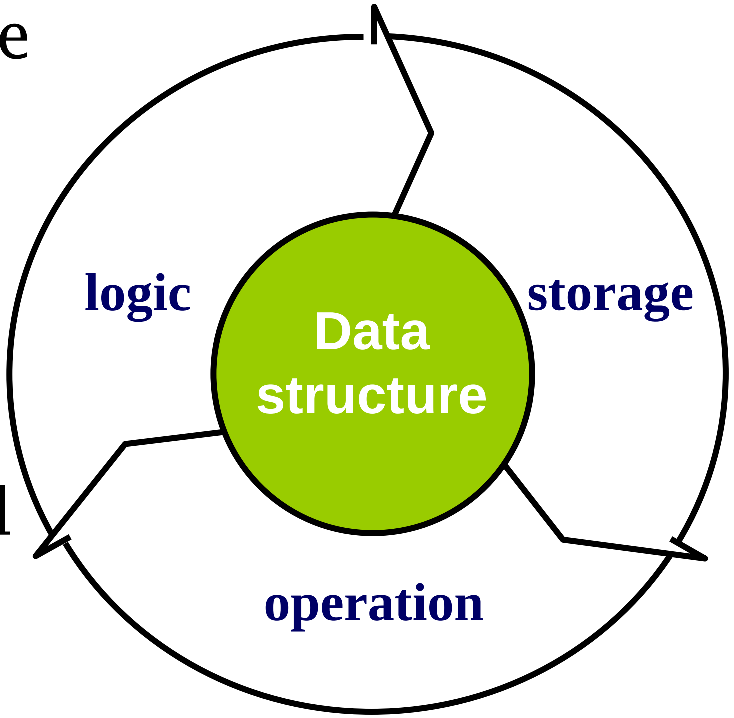
  - Link: a pointer points to the storage address, referring to a certain connection



- Four kinds : Sequence, link, index, hash

Abstract Data Type

- Abbreviated as **ADT** (Abstract Data Type)
  - A set of operations built upon a mathematical model
  - Has nothing to do with the physical storage structure
  - The software system is built upon the data model (object oriented)
- The development of **Modularization**
  - Hide the details of the implementation and operations of the internal data structures
  - Software reuse



ADT do not care about storage details

——for example , brackets matching algorithm of C++ version

```
void BracketMatch(char *str) {
  Stack<char> S; int i; char ch;
// The stack can be sequential
// or linked, both are referenced
// in the same way
  for(i=0; str[i]!='\0'; i++)  {
      switch(str[i])  {
        case '(':   case '[':   case '{':
              S.Push(str[i]); break;
        case ')':  case ']':   case '}':
           if (S.IsEmpty( )) {
            cout<<"Right brackets excess!";
                return;
           }
           else {
             ch = S.GetTop( );
             if (Match(ch,str[i]))
                  ch = S.Pop( );
             else {
                  cout << " Brackets do not match!";
                  return;
              }
           } /*else*/
      }/*switch*/
   }/*for*/
   if (S.IsEmpty( ))
       cout<<" Brackets match!";
   else cout<<"Left brackets excess!";
}
```

1.2 What is data structure

Sequential stack brackets matching algorithm of C version (different from the linked stack)

```
void BracketMatch(char *str) {
  SeqStack S; int i; char ch;
  InitStack(&S);
  for(i=0; str[i]!='\0'; i++)  {
      switch(str[i])  {
       case '(':    case '[':    case '{':
             Push(&S,str[i]); break;
       case ')':  case ']':    case '}':
             if (IsEmpty(&S)) {
                 printf("\nRight brackets
excess!");
                 return;
             }
             else {
                   GetTop (&S,&ch);
                   if (Match(ch,str[i]))
                        Pop(&S,&ch);
                   else {
                        printf("\nBrackets don’t
match!");
                        return;
                   }
                 } /*else*/
               }/*switch*/
             }/*for*/
             if (IsEmpty(&S))
                 printf("\nBrackets match!”)
             else printf("\nLeft brackets
excess" );}
```

1.2 What is data structure

Linked stack brackets matching algorithm of C version (different from the sequential stack)

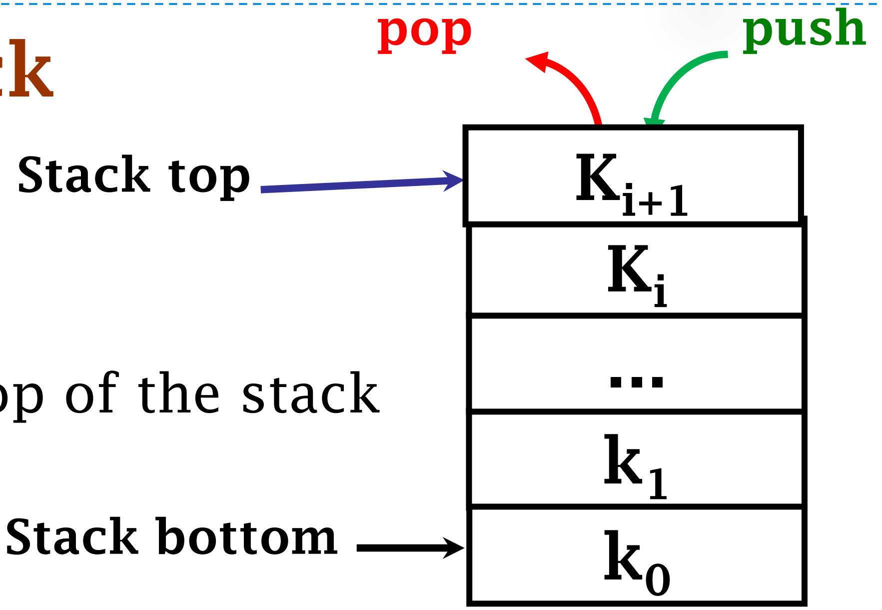
```
void BracketMatch(char *str) {
  LinkStack S; int i; char ch;
  InitStack(/*&*/S);
  for(i=0; str[i]!='\0'; i++)  {
      switch(str[i])  {
       case '(':    case '[':    case '{':
            Push(/*&*/S, str[i]);
            break;
       case ')':  case ']':    case '}':
            if (IsEmpty(S)) {
                printf("\nRight brackets
excess!");
                return;
            }
            else {
                GetTop (/*&*/S,&ch);
                if (Match(ch,str[i]))
                   Pop(/*&*/S,&ch);
                else {
                     printf("\nBrackets don't
match!");
                 return;
                 }
            } /*else*/
       }/*switch*/
  }/*for*/
  if (IsEmpty(/*&*/S))
       printf("\nBrackets match!")
  else printf("\nLeft brackets excess");}
```

Abstract Data Type

- Two-tuples of abstract data structure

  <Data object D , data operation P>
- Firstly, defines logical structure; then data operations
  - **Logical structure** : relationship between data objects
  - **Operations** : algorithms running on the data

Example : abstract data type of stack



- Logical structure : linear list
- Operation : Restricted access
  - Only allow for the insert, delete operation at the top of the stack
  - push 、 pop、 top 、 isEmpty

```
template <class T>            // Element type  of stack is T
class Stack {
public:                        // Stack operation set
    void clear();              // Turned into an empty stack
    bool push(const T item);// Push item into the stack , return true if succeed, otherwise false
    bool pop(T & item);// Pop item out of the stack ,  return true if succeed, otherwise false
    bool top(T& item); // Read item at the top of the stack, return true if succeed, otherwise false
    bool isEmpty(;       // If the stack is empty return true
    bool isFull();         // If the stack is full return true
};
```

Questions about abstract data type

- How to present a logical structure in an ADT ?
- Is abstract data type equivalent to the class definition ?
- Can you define a ADT without templates ?

Chapter 1 Overview

- Problem solving
- Data structures and abstract data types
- The properties and categories of algorithms
- Evaluating the efficiency of the algorithms

Problem——Algorithm—— Program

Goal : problem solving

- **Problem** (a function)
  - A mapping from input to output.
- **Algorithm** (a method)
  - The description for specific problem solving process is a finite sequence of instructions
- **Program**
  - It is the algorithm implemented using a computer programming language.

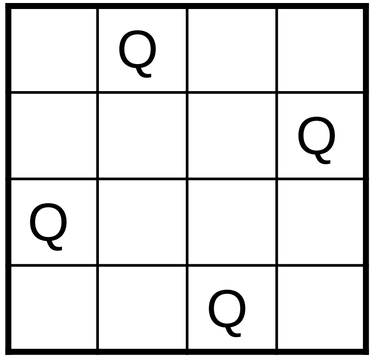
The properties of algorithms

	Q		
			Q
Q			
		Q	

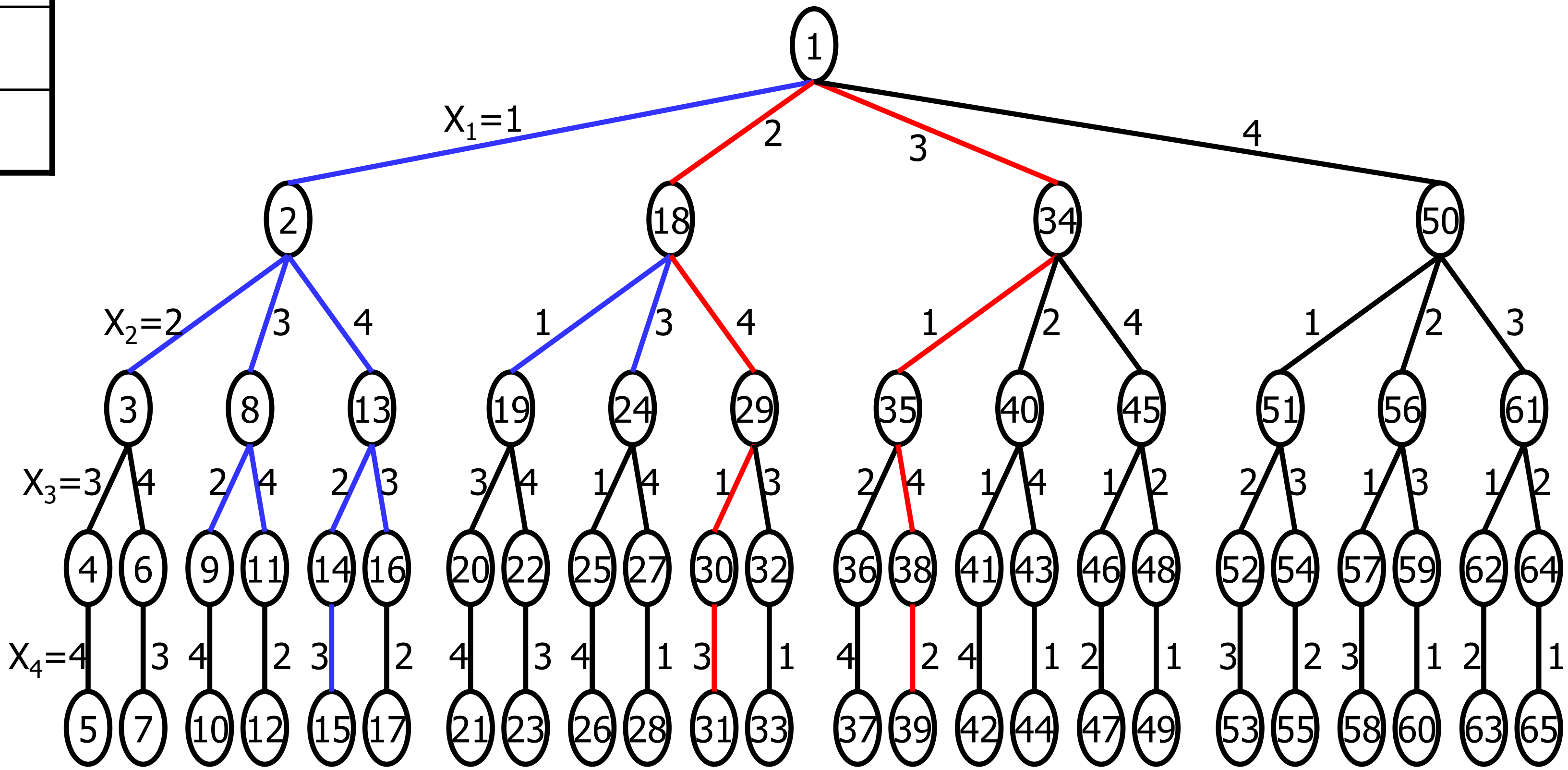
- Generality
  - Solve problems with parametric input
  - Ensure the correctness of the computation results
- Effectiveness
  - Algorithm is a sequence of finite instructions
  - It is made up of a series of concrete steps
- Certainty
  - In the algorithm description, which step will to be performed must be clear
- Finiteness
  - The execution of the algorithm must be ended in a finite number of steps
  - In other words, the algorithm cannot contain an endless loop

1.3 Algorithm

Queen problem (Four Queens)

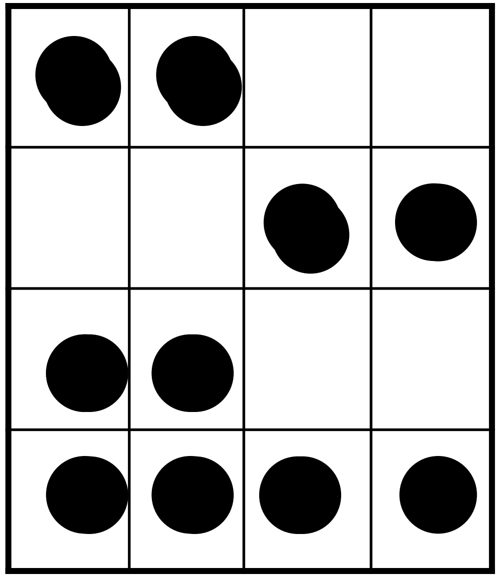


- **Solution**<$x1$,$x2$,$x3$,$x4$> (Place the column number)
- **Search space :** quadtree



1.3 Algorithm

Basic classification of algorithms



- **Enumeration**
  - Sequential search for value K
- **Backtracking、search**
  - Eight queens problem、traversal of trees and graphs
- **A recursive divide and conquer**
  - Binary search、quick sort、merge sort
- **Greedy**
  - Huffman coding tree、 Dijkstra algorithm for shortest path、Prim algorithm for minimum spanning tree
- **Dynamic programming**
  - Floyd algorithm for shortest path

Sequential Search

0	1	2	3	4	5	6	7	8
	17	35	22	18	93	60	88	52

```
template <class Type>
class Item {
private:
    Type key;                       // the key field
                                    //other fields
public:
    Item(Type value):key(value) {}
    Type getKey() {return key;}     // get the key
    void setKey(Type k){ key=k;}    // set the key
};
vector<Item<Type>*> dataList;
template <class Type> int SeqSearch(vector<Item<Type>*>& dataList, int length, Type k) {
    int i=length;
    dataList[0]->setKey (k);        // the zero-th element is a sentinel
    while(dataList[i]->getKey()!=k) i--;
    return i;                       // return the position of the element
}
```

Binary search

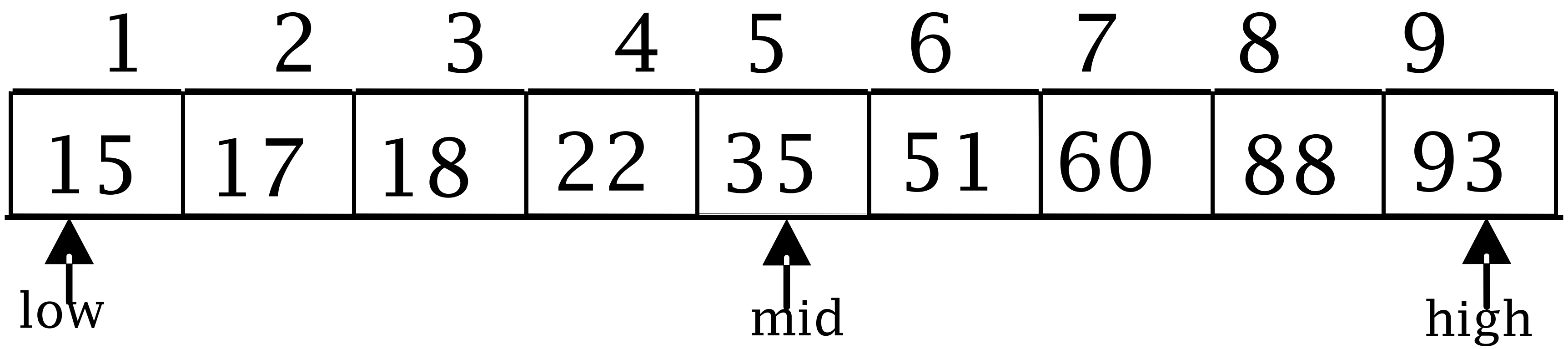
For sequential linear list that is in order

- $K_{mid:}$ The value of the element that is in the middle of the array
  - If $k_{mid} = k$, the search is successful
  - If $k_{mid} > k$, continue searching in the left half
  - Otherwise , if $k_{mid} < k$, You can ignore the part that before mid and search will go on in the right part
- Fast
  - $k_{mid} = k$, the search ends up successfully
  - $K_{mid} \neq k$, reduce half of the searching range at least

Use binary search to find value K

```
template  <class Type> int BinSearch (vector<Item<Type>*>& dataList,
int length, Type k){
    int low=1, high=length, mid;
    while (low<=high)  {
        mid=(low+high)/2;
        if (k<dataList[mid]->getKey())
              high = mid-1; // decrease the upper bound of the search interval
        else if (k>dataList[mid]->getKey())
              low = mid+1; // decrease the lower bound of the search interval
        else return mid;              // find value K and return the position
     }
    return 0;                         // fail to search and return 0
}
```

Illustration for binary search



Search the key value 18 low=1 high=9   K=18

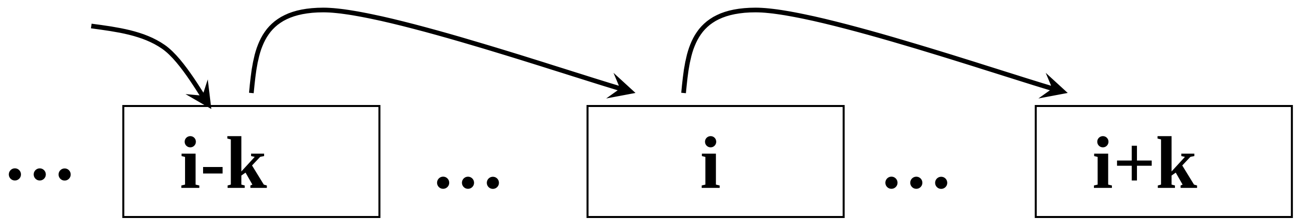
the first time : mid=5; array[5]=35>18
high=4; (low=1)
the second time : mid=2; array[2]=17<18
low=3; (high=4)
the third time : mid=3; array[3]=18=18
mid=3 ; return 3

Question： The time and space restrictions for algorithms

Design an algorithm that move the elements of the array A(0..n-1) to the right place by k positions circularly. The original array is supposed to be a_0, a_1, …, a_{n-2},a_{n-1} ; the array that has been moved will be a_{n-k}, a_{n-k+1}, …, a_0, a_1, …, a_{n-k-1}。 You are required to just use an extra space that is equivalent to an element, and the total number of moving and exchanging is only linearly correlated with n. 。 For example , n=10, k=3

The original array ： 0 1 2 3 4 5 6 7 8 9

The final array : 7 8 9 0 1 2 3 4 5 6



Chapter 1 Overview

- Problem solving
- Data structures and abstract data types
- The properties and categories of algorithms
- Evaluating the efficiency of the algorithms

Asymptotic analysis of algorithm

$$f(n) = n^2 + 100n + \log_{10}n + 1000$$

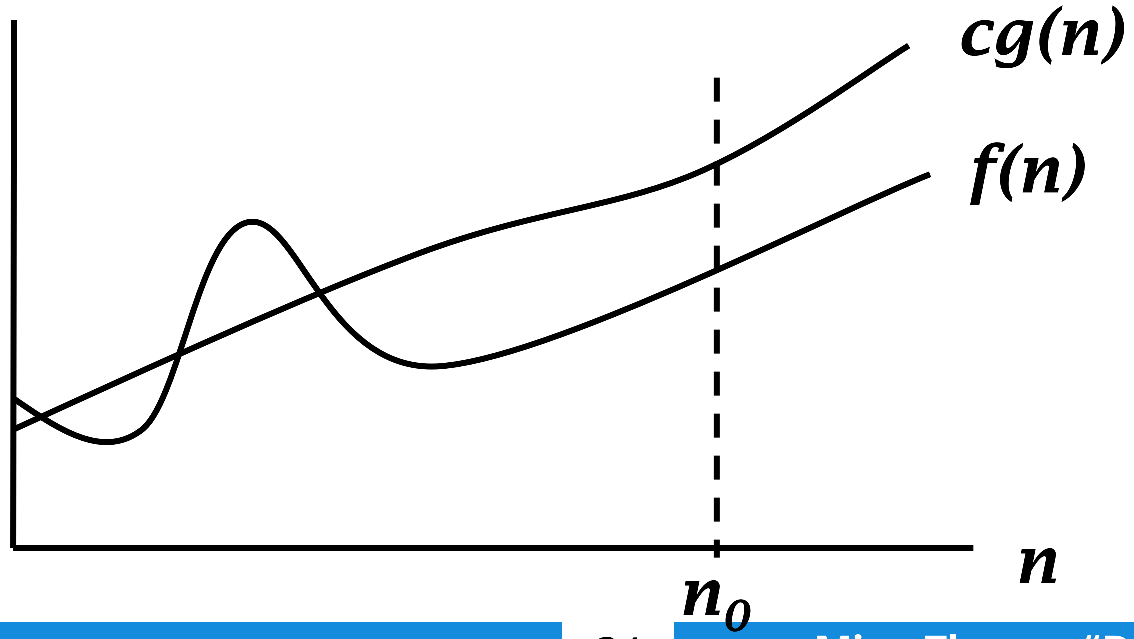
- f(n) is the growth rate as the data scale of n gradually increases
- When n increases to a certain value, the item with the highest power of n in the equation has the biggest impact
  - other items can be neglected.

Asymptotic analysis of algorithm : Big O notation

- The definition domain of function f and g is nature numbers , the range is non negative real numbers.
- **Definition :** If positive number c and n_0 exists , which makes for any $n \geq n_0$, $f(n) \leq cg(n)$,
- Then f(n) is said to be in the set of O(g(n)) , abbreviated as f(n) is O(g(n)) , or f(n) = O(g(n))
- Big O notation : it represents the upper bound of the growth rare of a function
  - There could be more than one upper bounds of the growth rare of a function
- When the upper bound and the lower bound are the same , you can use Big Θ notation.

Big O notation

- $f(n) = O(g(n))$, only when
  - There exists two parameters $c > 0$, $n_0 > 0$, for any $n \geq n_0$, $f(n) \leq cg(n)$
- iff $\exists\ c, n_0 > 0$ s.t. $\forall\ n \geq n_0 : 0 \leq f(n) \leq cg(n)$



n is large enough
g(n) is the upper bound of *f(n)*

Time unit of Big O notation

- Simple boolean or arithmetic operations
- Simple I/O
  - Input or output of a function

    For example , operations such as read data from an array
  - Files I/O operations or keyboard input are not excluded
- Return of function

Rules of operation of Big O notation

- **Rule of addition:** $f_1(n)+f_2(n)=O(\max(f_1(n), f_2(n)))$
  - Sequential structure , if structure , switch structure
- **Rule of Multiplication:** $f_1(n) \cdot f_2(n) = O(f_1(n) \cdot f_2(n))$
  - for, while, do-while structure

```
for (i=0; j<n; i++)
    for (j=i; j<n; j++)
        k++;
```

$n-i$

?

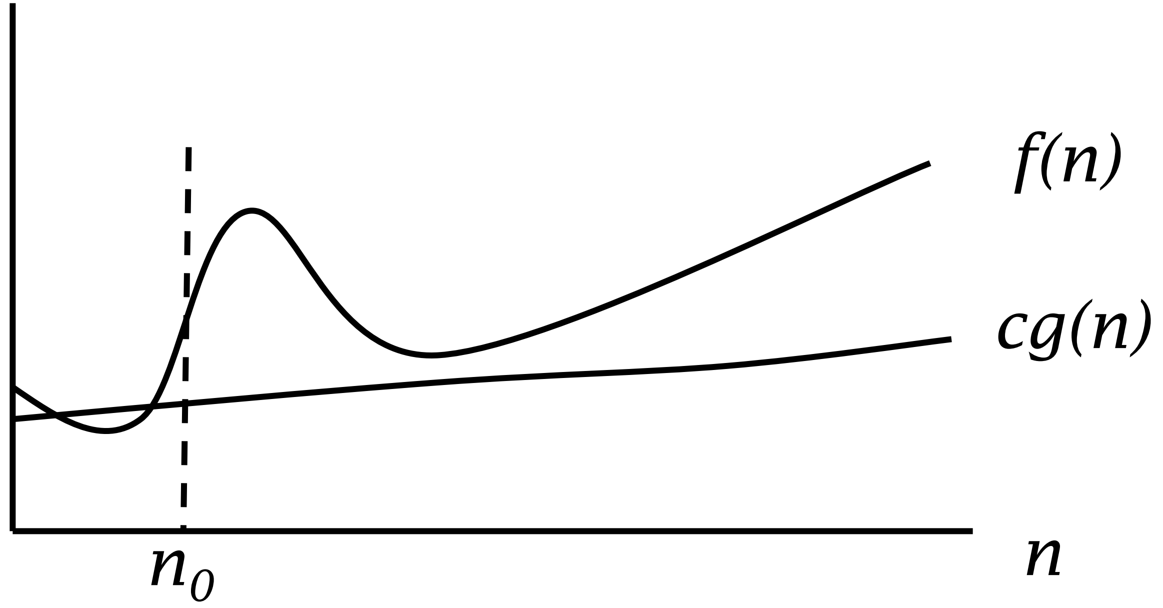
$$\sum_{i=0}^{n-1}(n-i)=\frac{n(n-1)}{2}=\frac{n^2-n}{2}=O(n^2)$$

Asymptotic analysis of algorithm ：Big Ω notation

- If positive number c and n_0 exists，which makes for any $n \geq n_0$， $f(n) \geq cg(n)$，
- Then f(n) is said to be in the set of O(g(n))， abbreviated as f(n) is O(g(n))，or f(n) = O(g(n))
- The only difference of Big O notation and Big Ω notation is the direction of inequation.
- When you adopt the Ω notation， you'd better find the tightest (largest) lower bound of all the lower bound of the growth rate of the function.

Big Ω notation

- $f(n) = \Omega(g(n))$
  - iff $\exists\, c, n_0 > 0$ s.t. $\forall\, n \ge n0$, $0 \le cg(n) \le f(n)$
- The only difference with Big O notation is the direction of inequation



n is large enough
g(n) is the lower bound of f(n)

Asymptotic analysis of algorithm ： Big Θ notation

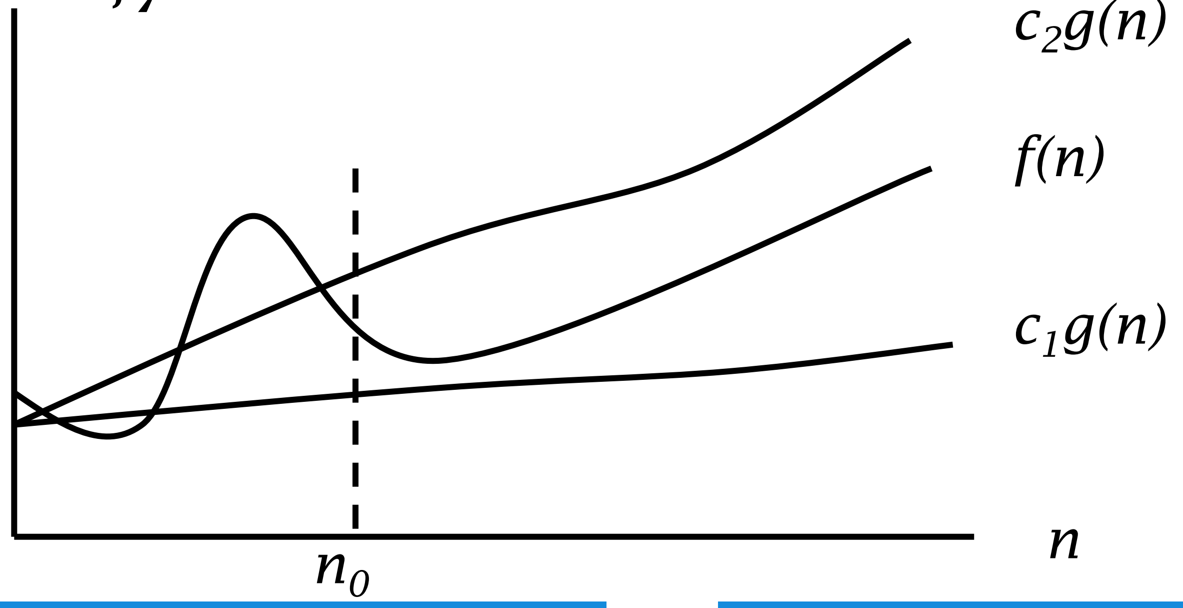
- When the upper bound and the lower bound are the same, you can use Θ notation.
- Definition :

  If a function is in the set of O (g(n)) and Ω (g(n)) ，it is called Θ (g(n))。
- In other words， When the upper bound and the lower bound are the same，you can use Big Θ notation.
- There exist c_1, c_2，and positive integer n_0，which makes for any positive integer $n > n_0$， The following two inequality are correct at the same time ：

$$c_1\, g(n) \le f(n) \le c_2\, g(n)$$

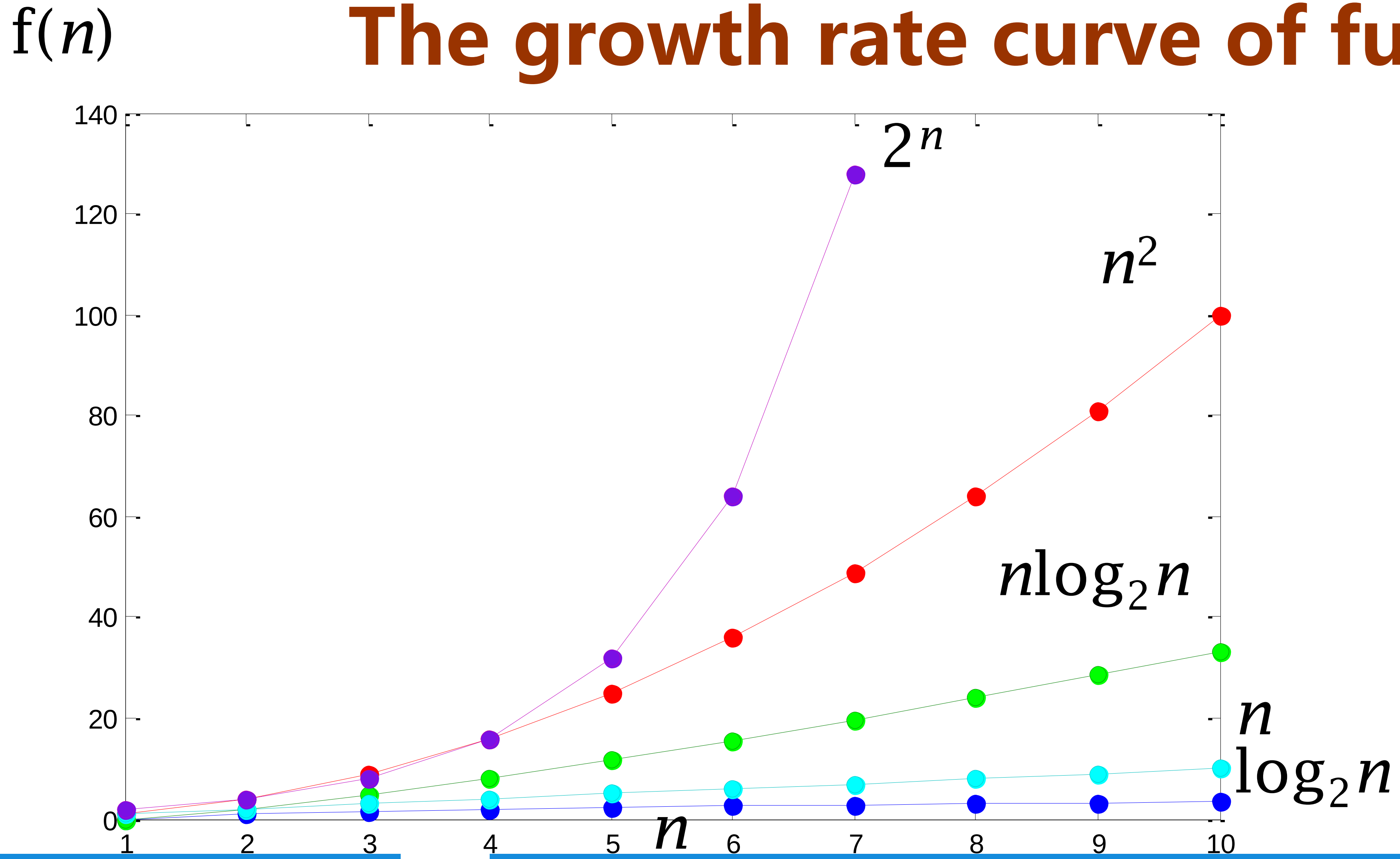
Big Θ notation

- $f(n) = \Theta(g(n))$
  - iff $\exists\, c_1, c_2, n_0 > 0$ s.t. $0 \le c_1 g(n) \le f(n) \le c_2 g(n), \ \forall n \ge n_0$
- When the upper bound and the lower bound are the same ,you can use Θ notation.

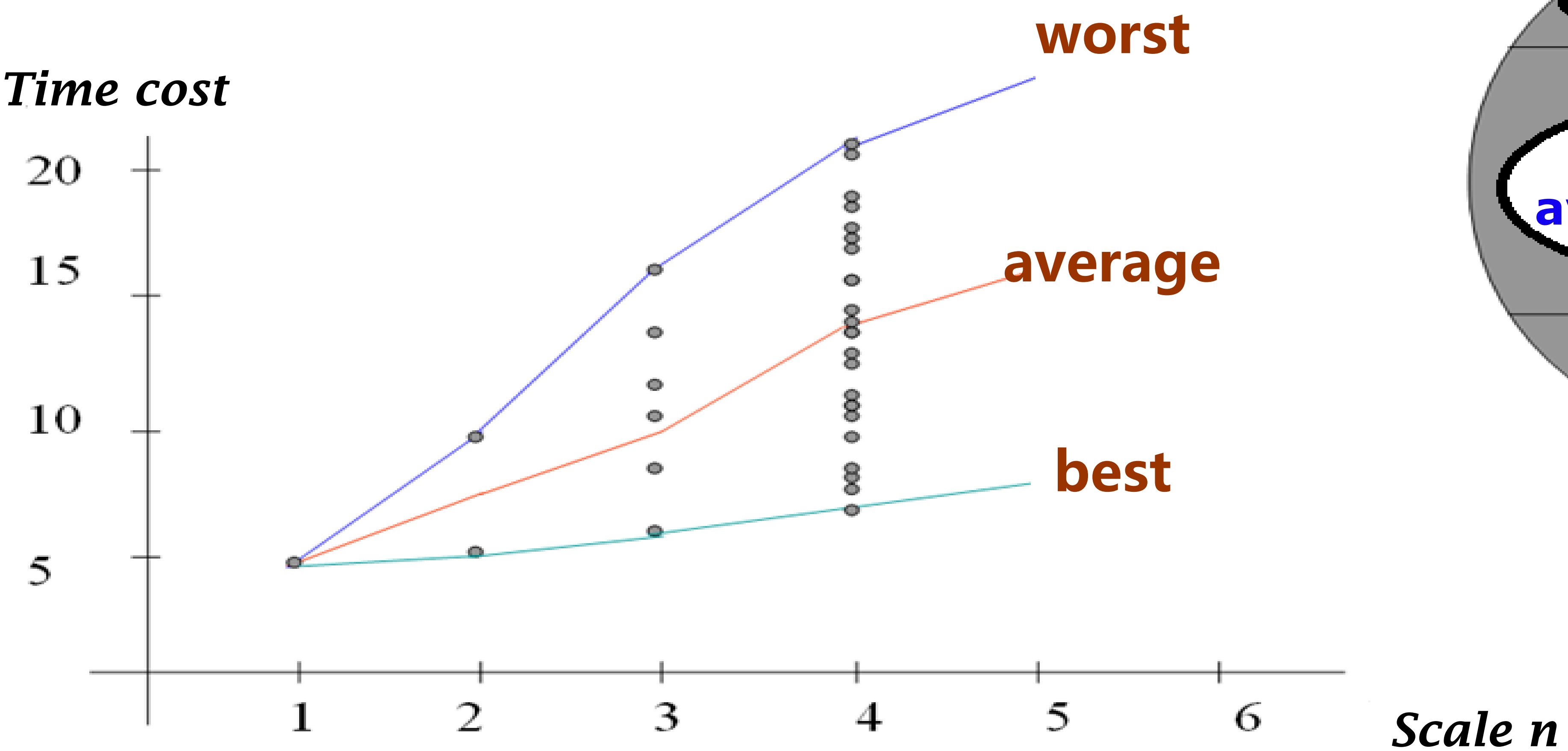


n is large enough
$g(n)$ has the same growth rate with $f(n)$

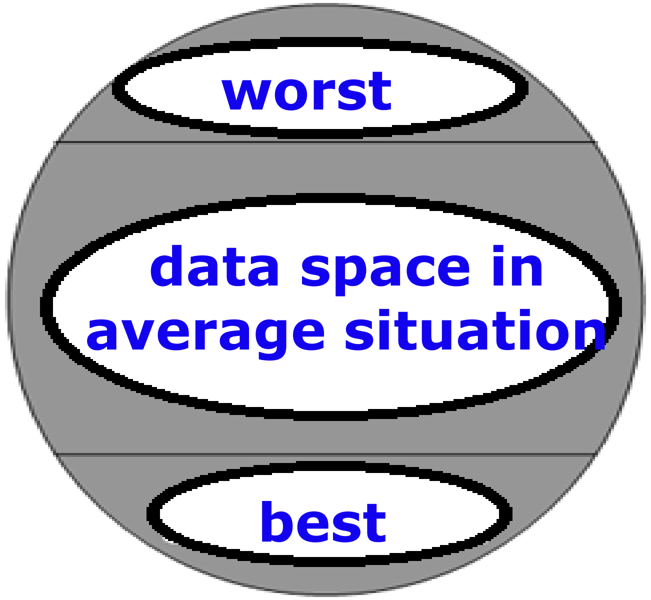
The growth rate curve of function



Problem space vs time overhead



The input data space of problem



Sequential Search

- You are required to find a given K in an array with a scale of n sequentially
- Best situation
  - The first element of the array is K
  - You only need to check one element
- Worst situation
  - K is the last element of the array
  - You need to check all the n elements of the array.

Find value k sequentially——the average case

- If value is distributed with equal probability
  - The probability that K occurs in every position is 1/n
- The average cost is O(n)

$$\frac{1+2+\ldots+n}{n}=\frac{n+1}{2}$$

Find value k sequentially——the average case

- Distributed with different probability
  - Probability that K occurs in position 1 is 1/2
  - Probability that K occurs in position 2 is 1/4
  - Probability that K occurs in other positions are all

$$\frac{1-1/2-1/4}{n-2}=\frac{1}{4(n-2)}$$

- The average cost is O(n)

$$\frac{1}{2}+\frac{2}{4}+\frac{3+\ldots+n}{4(n-2)}=1+\frac{n(n+1)-6}{8(n-2)}=1+\frac{n+3}{8}$$

Binary search

For sequential linear list that is in order

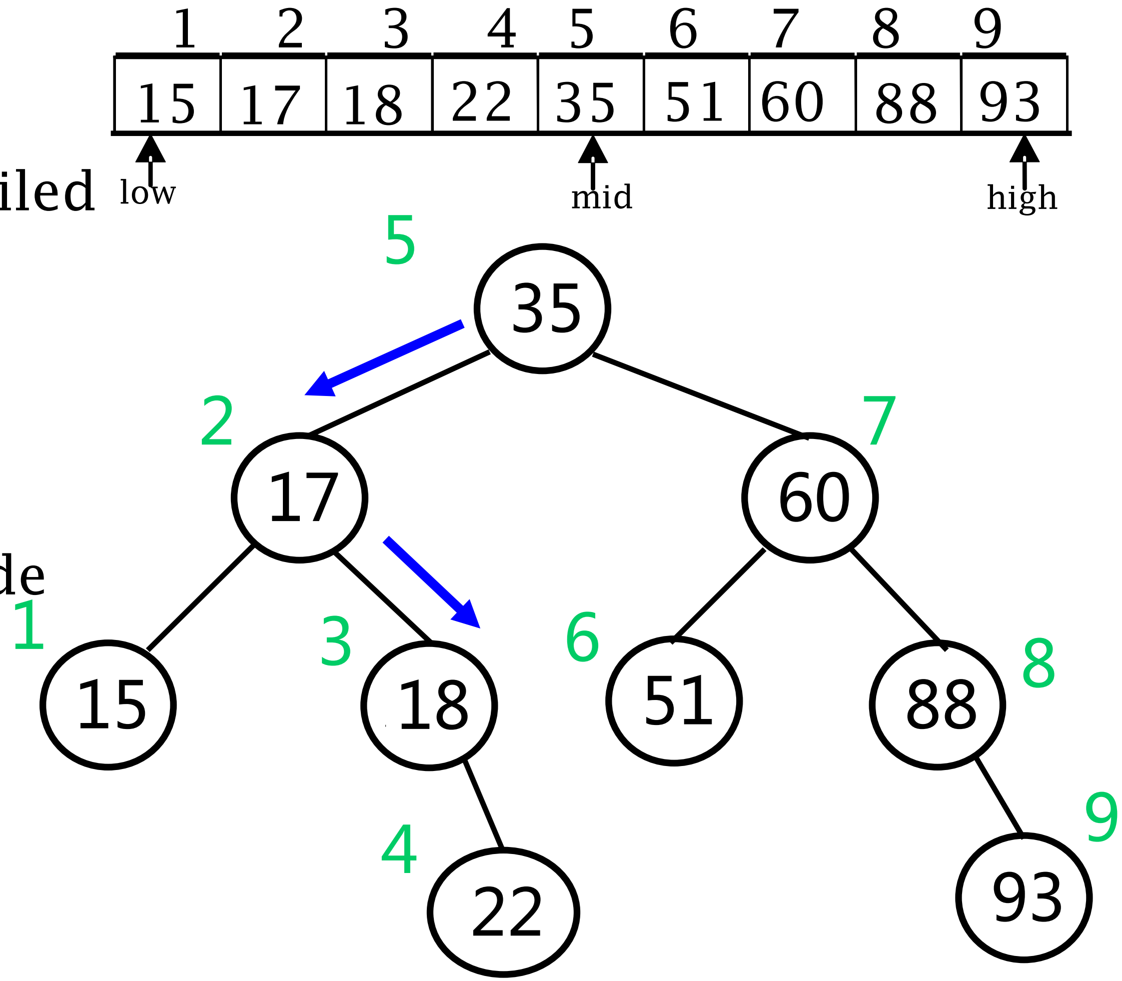
- K_{mid}: The value of the element that is in the middle of the array
  - If $k_{mid} = k$, the search is successful
  - If $k_{mid} > k$, the search continues in the left half
  - Otherwise , if $k_{mid} < k$, You can ignore the part that before mid and search will go on in the right part
- Fast
  - $k_{mid} = k$, search will be ended up
  - $K_{mid} \neq k$,reduce half of the searching range at least

Performance analysis of binary search

- The largest search length

  $$\lceil \log_2(n+1) \rceil$$

- The search length of the situation that failed is $\lceil \log_2(n+1) \rceil$ or $\lfloor \log_2(n+1) \rfloor$
- The average cost is $O(\log n)$
- In complexity analysis of algorithm
  - The base of log n is 2
  - When the base changed , the magnitude of algorithm will not change

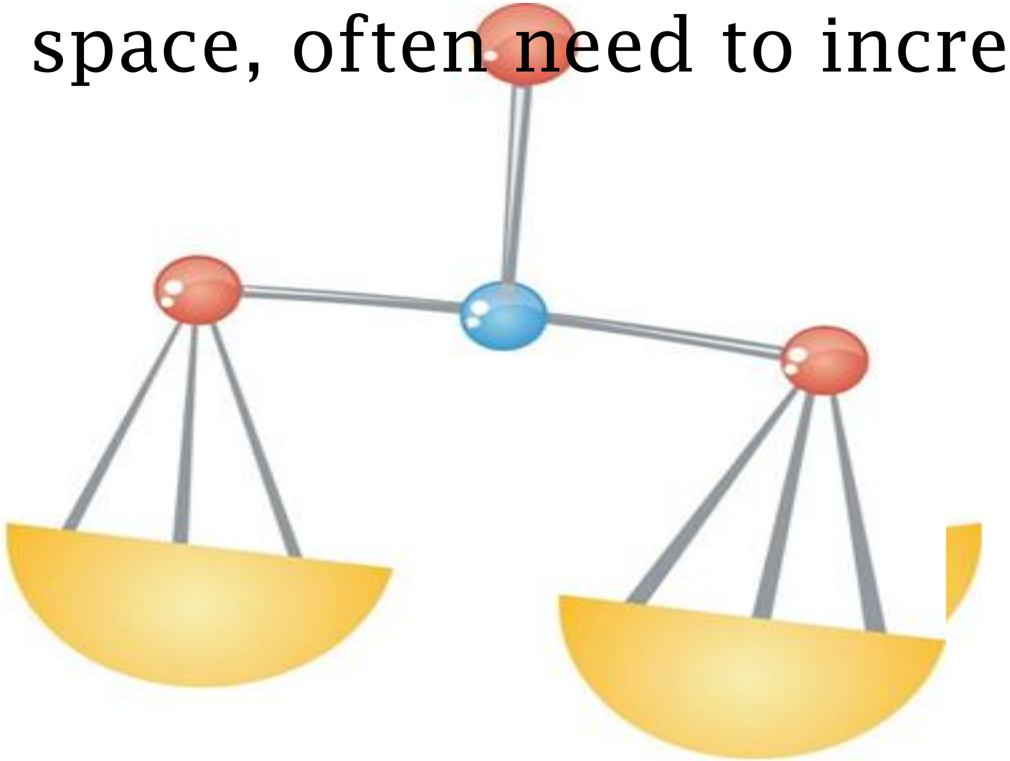


Time/Space tradeoff

- **Data structure**
  - A certain space to store every data item
  - A certain amount of time to perform a single basic operation
- **The cost and benefit**
  - limit of time and space
  - Software engineering

The space-time tradeoffs

- Increasing the space overhead may improve the algorithm's time overhead
- To save space, often need to increase the operation time



Selecting data structure and algorithm

- You need to analyze the problem carefully
  - Especially the logic relations and data types involved in the process of solving problems—problem abstraction、data abstraction
  - Preliminary design of data structure often precede the algorithm design
- Note the data structure of scalability
  - Consider when the size of input data changes , whether data structure is able to adapt to the evolution and expansion of problem solving

Question : Selecting data structure and algorithm

- Goal of problem solving ?
- Process of choosing data structure and algorithm ?

Question：three elements of data structure

Which of the structures below are logical structure and has nothing to do with the storage and operation().

A. Sequential table B. Hash table
C. Linear list D. Single linked list

The following terms（ _____ ）has nothing to do with the storage of data.

A. Sequential table B. Linked list
C. Queue D. Circular linked list

Data Structures and Algorithms

Thanks

the National Elaborate Course (Only available for IPs in China)
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (awarded as the "Eleventh Five-Year" national planning textbook)