



Data Structures

Algorithms

Teacher : Wang Wei

- 1. Ellis Horowitz, etc., Fundamentals of Data Structures in C++
- 2. 金远平, 数据结构
- 3. <http://inside.mines.edu/~dmehta/>
- 4. 殷人昆, 数据结构

王伟, 计算机工程系, 东南大学

1

Why need algorithms

- To computer science
 - The concept of an algorithm is fundamental
- In developing large-scale computer systems
 - Algorithms
 - exist for many common problems
 - designing efficient algorithms plays a crucial role

王伟, 计算机工程系, 东南大学

2

Algorithm

- Definition
 - is a step-by-step procedure
 - a finite set of instructions to be executed in a certain order to get the desired output
 - if followed, accomplishes a particular task
- Algorithms are generally created independent of underlying languages

王伟, 计算机工程系, 东南大学

3

Characteristics

- ◆ **Input**
 - ◆ Zero or more quantities are externally supplied
- ◆ **Output**
 - ◆ At least one quantity is produced
- ◆ **Definiteness**
 - ◆ Each instructions is clear abs unambiguous
- ◆ **Finiteness**
 - ◆ If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps
- ◆ **Effectiveness**
 - ◆ Every instruction must be basic enough to be carried out
 - ◆

王伟, 计算机工程系, 东南大学

4

Algorithm Analysis and Measurement

- Performance Criteria
- Posteriori testing
- Priori estimates
 - Asymptotic Analysis

王伟, 计算机工程系, 东南大学

5

Important Criteria

- **Correctness**
- **Readability**
- **Efficiency**
- **Robustness**
- **Usability**
- **Simplicity**

王伟, 计算机工程系, 东南大学

6

Complexities

- **Time Complexity** of a program
 - is the amount of computer time it needs to run to completion
 - Running time or the execution time of operations of data structure must be as small as possible
- **Space Complexity** of a program
 - is the amount of memory it needs to run to completion
 - Memory usage of a data structure operation should be as little as possible

王伟, 计算机工程系, 东南大学

7

Performance Measurement for Time Complexity

- **Posteriori testing**
 - is concerned with obtaining the **actual space and time requirements** of a program

王伟, 计算机工程系, 东南大学

8

Example : Sequential Search

```
int seqsearch ( int a[ ], int n, int x )
{
//在a[0],...,a[n-1]中搜索与给定值 x 相等的元
//素, 函数返回其位置, 失败返回-1。
    int i = 0;
    while ( i < n && a[i] != x )
        i++;
    if ( i == n ) return -1;
    return i;
}
```

王伟, 计算机工程系, 东南大学

9

- Measuring the computing time of a program
function *time()* or *clock()*

Example:

```
double runTime;  
double start, stop;  
time(&start);  
int k = seqsearch (a, n, x);  
time(&stop);  
runTime = stop - start;  
cout << " RunTime : " << runTime << endl;
```

- These quantities are dependent on the particular compiler and options used as well as on the specific computer on which the program is run

Performance analysis for Time Complexity

- **Priori estimates :**
 - to predict the growth in run time as the instance characteristics change
 - asymptotic notation
 - Big "oh" : O

Asymptotic Notation

- $f(n) = O(g(n))$
 - iff (if and only if) there exist positive constants c and n_0 such that $f(n) \leq cg(n)$ for all $n, n \geq n_0$
 - $g(n)$
 - is an **upper bound** on the value
 - should be **as small** a function of n as one come up

王伟, 计算机工程系, 东南大学

13

Theorem 1.2

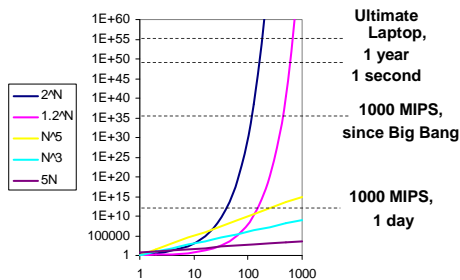
- if $f(n) = a_m n^m + \dots + a_1 n + a_0$, then $f(n) = O(n^m)$
 - Proof :

$$f(n) \leq \sum_{i=0}^m |a_i| n^i \leq n^m \sum_{i=0}^m |a_i| n^{i-m} \leq n^m \sum_{i=0}^m |a_i|, n \geq 1$$
 - So, $f(n) = O(n^m)$
 - When the complexity of an algorithm is actually, say, $O(\log n)$,
 - but we can only show that it is $O(n)$ due to the limitation of our knowledge
 - it is OK to say so.
 - This is one benefit of O notation as upper bound.

王伟, 计算机工程系, 东南大学

14

How the various functions grow with n ?



王伟, 计算机工程系, 东南大学

15

Time complexity

- The time taken by a program P
 $t(P) = c + t_p(n)$
 - c : constant
 - t_p : function $f_p(n)$
 - n : the number of the inputs and outputs
- $T(n) = O(f(n))$

王伟, 计算机工程系, 东南大学

16

- Compile time
- Run or execution time
 - ◆ **program step**
 - ☞ a syntactically or semantically meaningful segment of a program that has a run time
 - ☞ Run time is independent of n

王伟, 计算机工程系, 东南大学

17

■ Determine the number of steps : method 1

- ◆ Introduce a global variable *count* with initial value 0

```
int count=0;
float sum (float a[ ], int n)
{ float s = 0.0; //count++
  count++;
  for (int i = 0; i < n; i++) //count++ : <init>;<expr1>
  { count ++;
    s += a[i]; //count++
    count++;
  }
  count ++ //count++: <expr1>;<expr2>
  count++;
  return s; //count++ : return
}
```

王伟, 计算机工程系, 东南大学

18

■ Determine the number of steps : method 2

◆ build a *table*

s/e : steps per execution

program	s/e	frequency	steps
{	0	1	0
float s = 0.0;	1	1	1
for (int i=0; i<n; i++)	1	n+1	1n+1
s += a[i];	1	n	n
return s;	1	1	1
}	0	1	0
		total steps	2n+3

王伟, 计算机工程系, 东南大学

19

s/e : steps per execution

program	s/e	Frequency n=0/n>0	Steps n=0/n>0
{	0	1/1	0/0
if (n<=0)	1	1/1	1/1
return 0;	1	1/0	1/0
else			
return sum(a,n-1)+a[n-1];	1+f(n-1)	0/1	0/1+f(n-1)
}	0	1/1	0/0
		total steps	2/2+f(n-1)

王伟, 计算机工程系, 东南大学

20

$$T(n, m) = T_1(n) + T_2(m)$$

$$= O(\max(f(n), g(m)))$$

x = 0; y = 0;	$T_1(n) = O(1)$
for (int k = 0; k < n; k ++) x ++;	$T_2(n) = O(n)$
for (int i = 0; i < n; i ++) for (int j = 0; j < n; j ++) y ++;	$T_3(n) = O(n^2)$

$$T(n) = T_1(n) + T_2(n) + T_3(n) = O(\max(1, n, n^2)) = O(n^2)$$

王伟, 计算机工程系, 东南大学

21

```

void bubbleSort (int a[ ], int n)
{ //对表 a[ ] 逐趟比较, n 是表当前长度
  for (int i = 1; i <= n-1; i++)
  { //n-1 趟
    for (int j = n-1; j >= i; j--) //n-i次比较
      if (a[j-1] > a[j])
      { int tmp = a[j-1];
        a[j-1] = a[j];
        a[j] = tmp;
      } //一趟比较
  }
}

```

$$T(n, m) = T_1(n) * T_2(m)$$

$$= O(f(n)*g(m))$$

BubbleSort

外层循环 **n-1** 趟

内层循环 **n-i** 次比较

$$O(f(n)*g(n)) = O(n^2)$$

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

Execution Time Cases

three cases

- **Worst Case**
 - This is the scenario where a particular data structure operation takes maximum time it can take.
 - If an operation's worst case time is $f(n)$ then this operation will not take more than $f(n)$ time where $f(n)$ represents function of n
- **Average Case**
 - This is the scenario depicting the average execution time of an operation of a data structure.
 - If an operation takes $f(n)$ time in execution, then m operations will take $mf(n)$ time
- **Best Case**
 - This is the scenario depicting the least possible execution time of an operation of a data structure.
 - If an operation takes $f(n)$ time in execution, then the actual operation may take time as the random number which would be maximum as $f(n)$

Space complexity

- The space requirement of program P
 $S(P) = c + S_p(n)$
 - c : constant
 - S_p : function $f_p(n)$
 - n : the number of the inputs and outputs
- $S(n) = O(f(n))$

王伟, 计算机工程系, 东南大学

25

- **Fixed part**: is **independent** of the number of the inputs and outputs
 - Space for the code
 - Constant
 - Simple variables
 - Fixed-size component variables
- **Variable part**: is **dependent** on the particular instance
 - component variables
 - Referenced variables
 - Recursion stack space

王伟, 计算机工程系, 东南大学

26

Example

```
//iterative function
float Sum (float *a, const int n)
{ float s=0;
  for(int i=0;i<n;i++)
    s+=a[i];
  return s;
}

//recursive function
float Rsum (float *a, const int n)
{ if (n <=0) return 0;
  else return (Rsum(a,n-1)+a[n-1]);
}
```

27
