# Data Structures & Algorithms Assumption

## Theoretical model of a computer

no cache

uniform memory access

all unit times the same: no difference between \* + if ( … )

**Ignore constant factors**

int n = ??

Formal f(n) = O(n) means there exists some c for which cn > f(n)

f(n) = 600n + 20000000000 c = 601 O(n)

Formal f(n) = Ω(n) means there exists some c for which cn < f(n)

A function that is f(n) = O(n) and Ω(n) is Ө(n)

O(n) Ω(1)

n n2 1

1 is fastest (no change regardless of n)

all log are the same

n

n2 worst

**考察复杂度，常数复杂度最低。**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| O(1) |  |  |  | n | n2 |
| 1 | 1 | 1 | 1 | 1 |  |
| 1 | 3 | 2 | 3 | 10 |  |
| 1 | 10 | 4 | 6 | 102 |  |
| 1 | 1000 |  | 20 | 106 |  |
| 1 | 104.5 |  | 30 | 109 |  |
| 1 | 106 |  | 40 | 1012 |  |

**忽略常数，所以复杂度一样**

**O(n2)**

Sum 🡨 0 O(1)

For i 🡨 5 to n – 18 step 92 O(n)

For i 🡨 1 to n O(n)

End

End

O(2n) = O(3n) = O(n)

//O(n) Ω(1)

Ispreme

For i 🡨 2 to n-1 O(n)

If n mod i == 0 O(1)

Return false

End

End

//O(sqrt(n)) Ω(1)

Ispremebeter

For i 🡨 2 to quar O(sqrt(n))

If n mod i == 0 O(1)

Return false

End

End

//O(n log n)

for (int i = 1; i < n; i \*= 2) //O(log n)

for (int j = 1; j < n; j++) //O(n)

Combining Complexities

//O(n\*n) = O(n2)

for (int i = 0; i < n; i++) //O(n)  
 for (int j = 0; j < n; j++) { //O(n)  
 }

}

Sqrt(n) 🡨 n O(n)

Recursion can reduce the complexity.

Sometimes:

Call self twice and more can improve the complexity.

//O(n+n) = O(n)

for (int i = 0; i < n; i++) //O(n)  
}

for (int j = 0; j < n; j++) { //O(n)  
}

for (int i = 0; i <= n; i++) //O(n)  
 for (int j = 0; j <= i; j++) { // 1 + 2 + 3 + … + n-1+ n   
 }

}

sum (1..n) =

f(n) = n2+n O(n2) c1 = 2 2n2

for (int i = 0; i <= n; i++) { //O(n)  
 for (int j = i; j < n; j++) { // n + n-1 + n-2 + … +1   
 }

}

//O(n log2n)

for (int i = 0; i < n; i++) { // O(n)  
 for (int j = 1; j < n; j \*= 2) { // log2n

}

}

for (int i = 1; i < n; i \*= 2) { // O(log2n)  
 for (int j = 1; j < n; j++) { // O(n)

}

}

for (int i = 1; i <= n; i \*= 2) { // O(log2n)  
 for (int j = 1; j < i; j++) { // O(1 + 2 + 4 + 8 + …. + n)

}

}

suppose n=1024

n ( 1+½+¼+⅛ + …)

//O(n)\*O(m) = O(mn)

for (int i = 0; i < n; i++) { //O(n)

for (int j = 0; j < m; j++) { //O(m)  
 cout << i \* j; //O(1)  
 }

}

//O(n+m) if n ≈ m O(2n) = O(n)

for (int i = 0; i < n; i++) { //O(n)

}

for (int j = 0; j < m; j++) { //O(m)  
 cout << i \* j; //O(1)  
}

//O(n log n)

for (int i = 1; i < n; i \*= 2) //O(log n)

for (int j = 1; j < n; j++) //O(n)

System.out.println(i\*j);

for (int j = 1; j < n; j++) //O(n)

for (int i = 1; i < n; i \*= 2) //O(log n)

System.out.println(i\*j);

for (int i = 0; i < n; i++) //O(n)  
 for (int j = i; j < n; j++) // n + (n-1) + (n-2) + … 1  
 println(“x”);

O(n(n+1)/2) = O(n(n+1)) = O(n2+n) = O(n2) definition O( ) c1n2 > an2+bn

sum (1..n) =

O(n2)

sum← 912851; //O(1)

for i ← 1 to n

for j← 3 to i // 4 + 5 … n = sum (1..n) = n \* (n+1) / 2

sum ← sum + i\*j;

end

end

# Functions and Recursion

f(x) = x2

f(3) //O(1)

f(918275912857192851982791827591875198257) // big numbers handled separately!

int f(int n) { //O(1)

if (n <= 0)  
 return 1;

return n \* f(n-1);

}

5 \* f(4) = 120

4\* f(3) =24

3\* f(2) =6

2\*f(1) =2

1\*f(0) = 1

f(1)

f(5) = 120

f(10) = 3628800

//10328

double f(int n) {

if (n <= 0)  
 return 1;

return n \* f(n-1);

}

Exercise:

int count(int n) {

int sum = 0;

for (int i = 1; i <= n; i++)

sum++;

return sum;

}

1. Write this recursively count(5) → 1 + count(4) count(4) → 1 + count(3) ...
2. sum an array, recursively sum(a, 0) = a[0] + sum(a, 1) …
3. Implement the Ackermann function http://mathworld.wolfram.com/AckermannFunction.html

int sum(int[] array, int pos) {  
  
}

1, 1, 2, 3, 5, 8, 13, …

1 2 3 4 5 6 7

int fibo(int n) {

int a = 1, b = 1, c;

for (int i = 0; i < n; i++) {

c = a + b;

a = b;

b = c;

}

return c;

}

int fibo2(int n) {

if ( n <= 2)

return 1;

return fibo2(n-1) + fibo2(n-2);

}

O(2n)

+

+

+

1 + 2 + 4 + …. 2n = 20 + 21 + 22 + …+2n

210 = 1024

220 = 10242

230 = 10243

240 = 10244

Memoization = dynamic programming

int fibo2(int n) {

if ( n <= 2)

return 1;

static int memo[200] = {0};

if (memo[n] != 0)  
 return memo[n];

return memo[n] = fibo2(n-1) + fibo2(n-2);

}

n! = n factorial = n \* (n-1) \* ….

nCr = choose(n, r) =

==O(n)+O(r)+O(n-r)=O(n)

1. Compute brute force. What is the complexity? choose (52,6) n < 1000, r <= n
2. Memoize. How fast can you make this? How big a table do you need?

# Number Theoretic Algorithms

prime number is a whole number only divisible by 1 and itself. 1 is not prime.

2, 3, 5, 7, 11, …

//

bool isPrime(int n) {  
 for (int i = 2; i < n; i++) {

if (n % i == 0)

return false;

}  
 return true;  
}

isPrime(1001);

isPrime(1000);

//O(n) Ω(1)

bool isPrime(int n) {  
 for (int i = 2; i < n/2; i++) {

if (n % i == 0)

return false;

}  
 return true;  
}

//O() Ω(1)

bool isPrime(int n) {  
 for (int i = 2; i <= sqrt(n); i++) {

if (n % i == 0)

return false;

}  
 return true;  
}

28 = 2 4 \* 7 14

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | n2 | n/2 | sqrt(n) | log2n |
| 1 | 1 | 0 | 1 | 0 |
| 10 | 100 | 5 | 3 | 3 |
| 100 | 10000 | 50 | 10 | 6 |
| 1000 | 1000000 | 500 | 33 | 10 |
| 106 | 1012 | 5x105 | 1000 | 20 |

n = 1001

//O(sqrt(n))

bool isPrime(int n) {

if (n == 2)

return true;

if (n % 2 == 0)

return false;  
 for (int i = 3; i <= sqrt(n); i+=2) { // 2, 3, 4, 5, 6 // now 3, 5, 7, ...

if (n % i == 0)

return false;

}  
 return true;  
}

# Eratosthenes’ Sieve

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21

2, 3, --, 5, --, 7, --, 9, --, 11, --, 13, --, 15, --, 17, --, 19, --, 21

2, 3, --, 5, xx, 7, --, --, --, 11, xx, 13, --, --, --, 17, xx, 19, --, --

void eratosthenes(int n) {

boolean[] sieve = new boolean[n]; //O(n)

for (int i = 2; i < n; i++) //O(n)

sieve[i] = true;

for (int i = 2; i < n; i++) {

if (sieve[i]) {

print(i);

for (int j = 2\*i; j < n; j += i)

sieve[j] = false;

}

}

Better Eratosthenes

2, 3, -, 5, -, 7, -, 9, 10, 11, 12, 13, 14, 15,, 16, 17, 18, 19, 20, 21

2\*i 3\*i 4\*i…

start with j2

3\*3 is odd (all primes except 2 are odd, and odd\*odd = odd)

odd + odd = even, so 9 + 3 → 12 already done

so add 2\*3

2, 3, -, 5, -, 7, -, 9, -, 11, -, 13, -, 15, - 17, -, 19, -, 21

2, 3, -, 5, -, 7, -, \*, -, 11, -, 13, -, \*, - 17, -, 19, -, \*

97

start with j2