Data Structures And Algorithms

References :

- Cormen, Leiserson, Rivest, Stein, *Introduction To Algorithms, 2nd Ed.*, McGraw Hill / The MIT Press, 2001

- Ellis Horowitz, Sartaj Sahni, *Fundamentals Of Data Structures In C++*

- A.V. Aho, J.E. Hopcroft, J.D. Ullman, *Data Structures And Algorithms, 1983*

U. Manber, *Introduction To Algorithms: A Creative Approach*, Addison Wesley, 1989

Topics

- Trees
  1. Definitions
  2. Implementing methods
  3. Traversing methods
  4. Expression tree
  5. Threaded tree
  6. Huffman tree and coding
  7. Heap
  8. Binary search tree
  9. Selection tree
- Hashing
- Algorithm Analysis Methods
  1. Definitions
  2. Complexity computation
  3. Methods for solving recursive problems
- Recursive Algorithms
- Basic Data Structures
  1. Abstract Data Types
  2. Array
  3. Stack
  4. Queue
  5. Link list

Topics (contd.)

- Problem solving methods
  1. Divide and conquer
  2. Dynamic programming
  3. Greedy
  4. Backtracking
- Trees (contd)
  1. B-tree
  2. AVL
  3. 2-3 Tree
  4. 2-3-4 tree
- Advanced Data Structures
  1. Red-Black tree
  2. Binomial heap
  3. Fibonacci heap
  4. Disjoint set
- Sorting Algorithms
  - ❖ Based on compare
    1. Bubble sort
    2. Selection sort
    3. Quick sort
    4. Heap sort
    5. Merge sort
    6. Shell sort
  - ❖ Not based on compare
    1. Count sort
    2. Radix sort
    3. Bucket sort
- Graphs
  1. Definitions
  2. Implementing methods
  3. Traversing methods
  4. Famous problems

Algorithm Analysis

Example : Sorting

Sorting an array with n items ascending :

a[0]<=a[1]<=a[2]<=…..<=a[n]

Insertion Sort

- Start with a single-size subset
- Each time *Insert* the next item into the sorted subset

Inserting 5 into 3,6,9,14

Initial	3	6	9	14	5	…
First	3	6	9	14	14	…
Second	3	6	9	9	14	…
Third	3	6	6	9	14	…
Last	3	5	6	9	14	…

Hold 5 in a temporary variable and put it in the array when needed

Sorting With Insertion Sort

	Sorted		Unsorted			
Original List	23	78	45	8	32	56
After Pass 1	23	78	45	8	32	56
After Pass 2	23	45	78	8	32	56
After Pass 3	8	23	45	78	32	56
After Pass 4	8	23	32	45	78	56
After Pass 5	8	23	32	45	56	78

Insertion Sort Algorithm

Insertion Sort	Cost * Times Executed
For k=2 to length(A)	$c_1 * n$
Key = A[k]	$c_2 * n\text{-}1$
i=k-1	$c_3 * n\text{-}1$
While(i>0 and A[i] > key)	$c_4 * \sum_{k=2}^{n} t_k$ $(t_k \; depends \; on \; items \; order)$
A[i+1]= A[i]	$c_5 * \sum_{k=2}^{n} (t_k\text{-}1)$
i=i-1	$c_6 * \sum_{k=2}^{n} (t_k\text{-}1)$
A[i+1]=key	$c_7 * n\text{-}1$

$$T(n) = c_1 n + c_2 * (n-1) + c_3 * (n-1) + c_4 * \sum_{k=2}^{n} t_k + c_5 * \sum_{k=2}^{n} (t_k - 1) + c_6 * \sum_{k=2}^{n} (t_k - 1) + c_7 * (n-1)$$

$$T(n) = c_1 n + c_2 * (n-1) + c_3 * (n-1) + c_4 \sum_{k=2}^{n} t_k + c_5 \sum_{k=2}^{n} (t_k - 1) + c_6 \sum_{k=2}^{n} (t_k - 1) + c_7 * (n-1)$$

$$T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{k=2}^{n} t_k + (c_5 + c_6) \sum_{k=2}^{n} (t_k - 1)$$

Best Situation : $t_k = 1$:

$$T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4(n-1) = c_1 n + (c_2 + c_3 + c_4 + c_7)(n-1) \rightarrow T(n) = A_1 n + B_1$$

Worst Situation : t_k = k :

$$\sum_{k=2}^{n} t_k = \sum_{k=2}^{n} k = \frac{n(n-2)}{2} - 1 \qquad \sum_{k=2}^{n} (t_k - 1) = \sum_{k=1}^{n-1} k = \frac{(n-1)(n-2)}{2}$$

$$T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \left[\frac{n(n-1)}{2} - 1\right] + (c_5 + c_6) \left[\frac{(n-1)(n-2)}{2}\right]$$

$$T(n) = A_2 n^2 + B_2 n + C_2$$

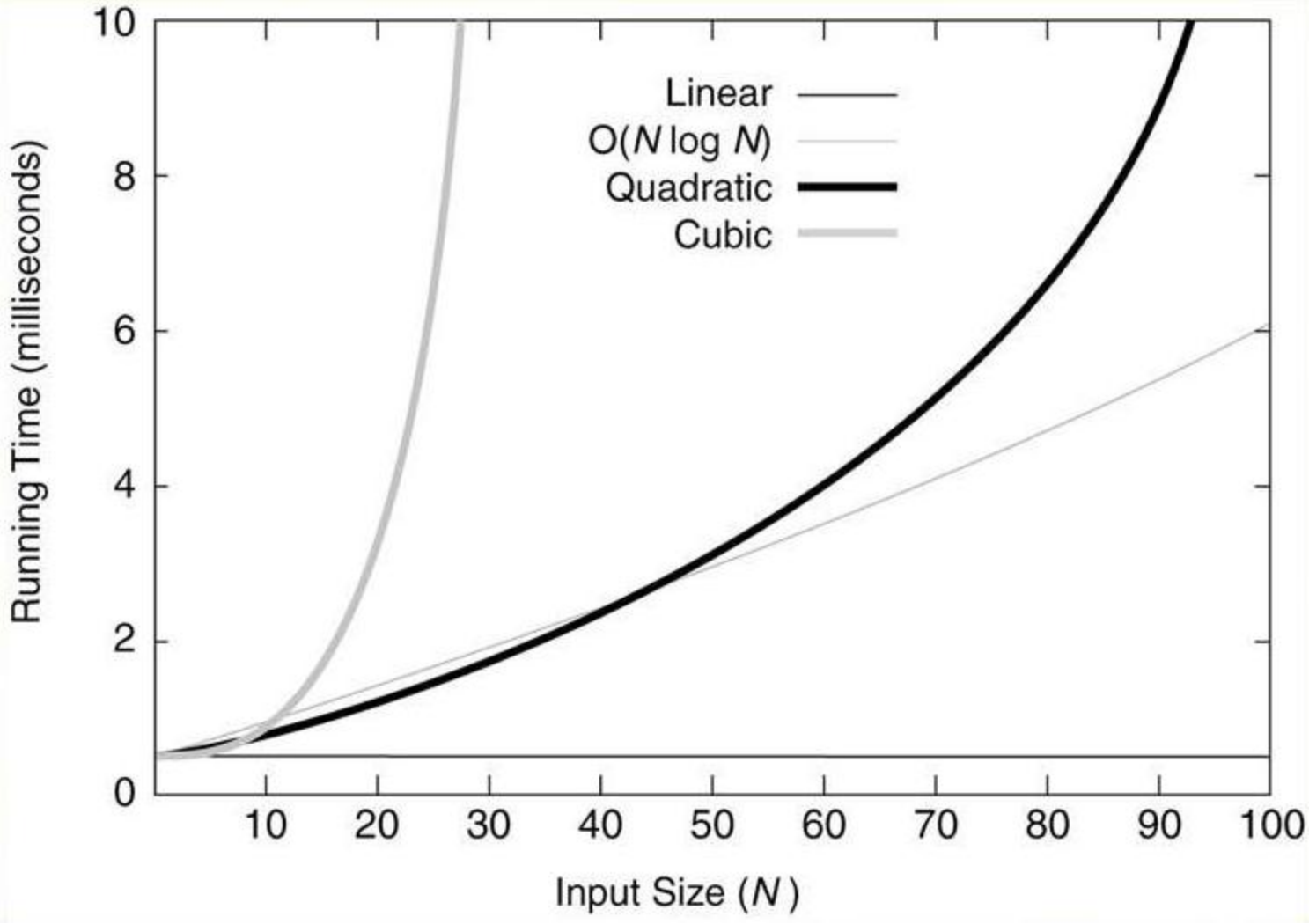
Average Situation :

$$T(n) = A \sum_{k=1}^{n} t_k + Bn + C = A \sum_{k=1}^{n} \left(\frac{1}{k} \sum_{i=1}^{k} i \right) + Bn + C \Longrightarrow T(n) = A_3 n^2 + B_3 n + C_3$$

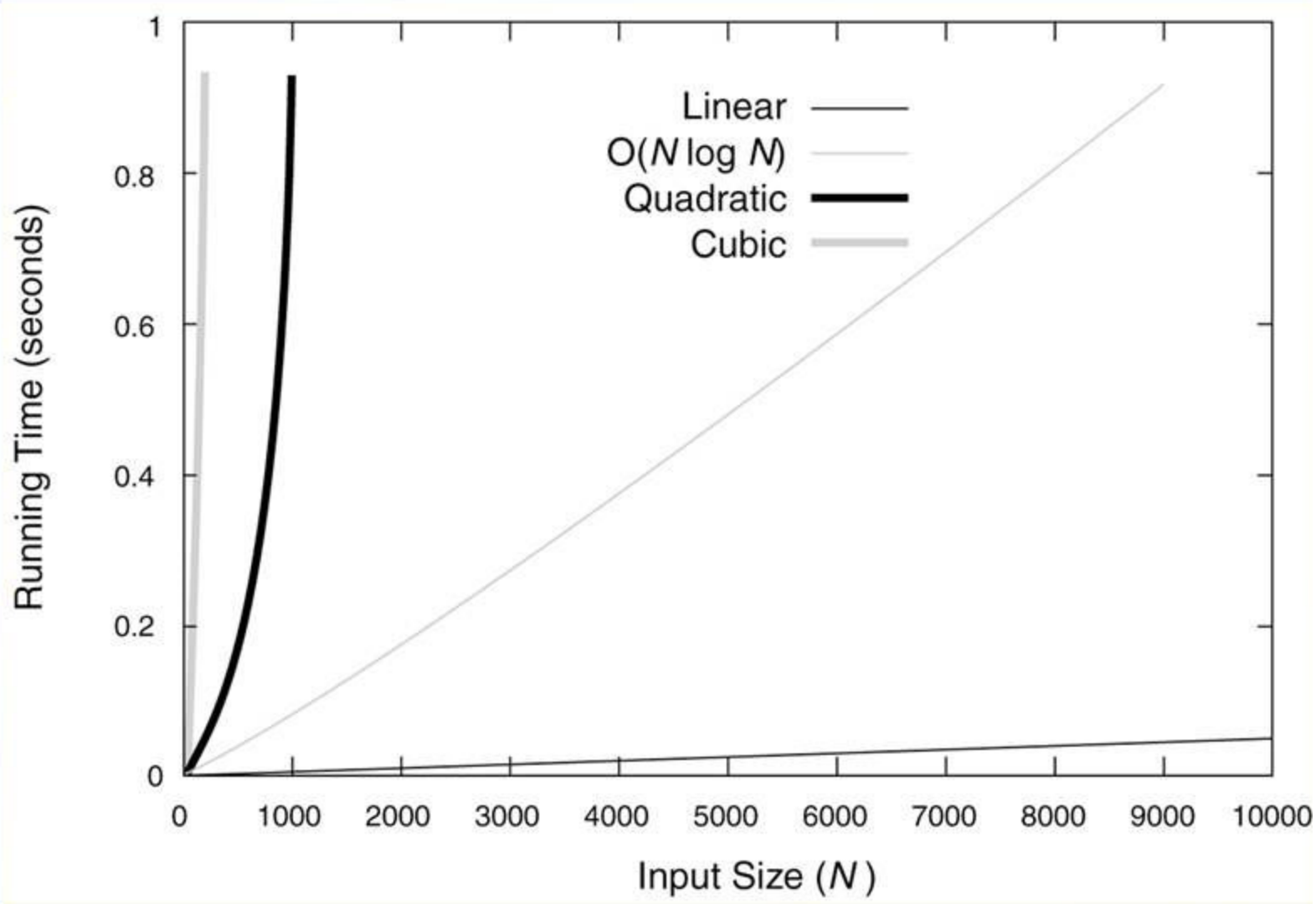
Comparing Different Algorithms

	T(n)	"n" that can be solved in 1000s	For 10 times faster machine	Ratio	Ratio if we use a machine 1000 times faster
$O(n)$	$100n$	10	100	10	1000.00
$O(n^2)$	$5n^2$	14	45	3.2	31.94
$O(n^3)$	$\frac{n^3}{2}$	12	27	2.3	10.50
$O(2^n)$	2^n	10	13	1.3	2.00

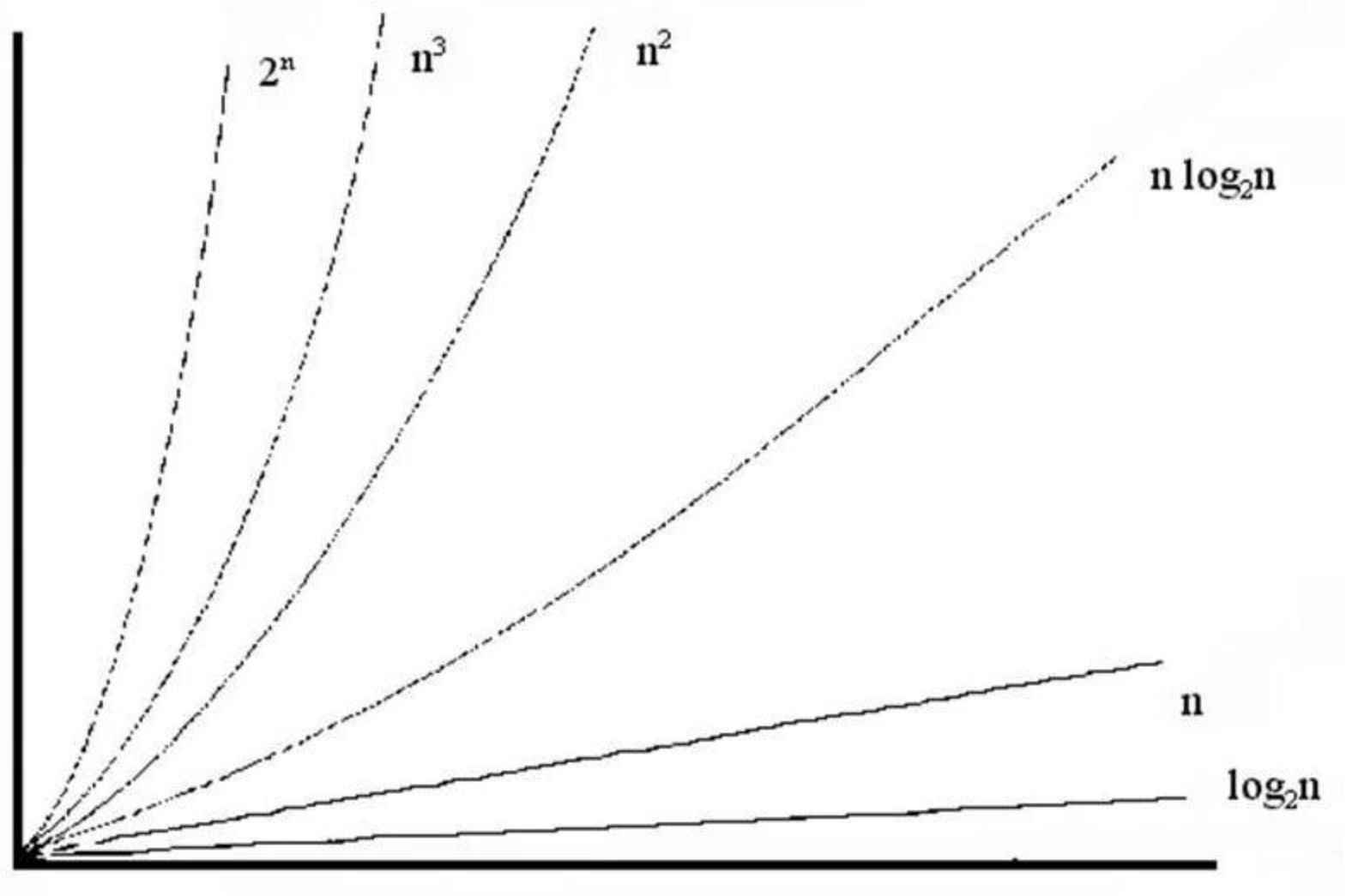
Running Times For Small Inputs



Running times For Moderate Inputs



Common Growth Rates



Common Growth Rates

Function	Growth Rate Name
c	Constant
$\log n$	Logarithmic
$\log_2 n$	Log-Squared
n	Linear
$n \log n$	
n^2	Quadratic
n^3	Cubic
2^n	Exponential

Comparison of N, $\log n$ *and* n^2

$O(n)$	$O(\log n)$	$O(n^2)$
16	4	256
64	6	4K
256	8	64K
1,024	10	1M
16,384	14	256M
131,072	17	16G
262,144	18	6.87E+10
524,188	19	2.74E+11
1,048,576	20	1.09E+12
1,073,741,824	30	1.15E+18

Algorithm Analysis : Big Oh

$$O\big(g(n)\big) = \{f(n)\colon \exists c, n_0 > 0 \mid \forall n \geq n_0\ 0 \leq f(n) \leq cg(n)\}$$

In other words c*g(n) is larger than f(n) for a big N

$3n + 2 = O(n)$ $\quad *** 3n + 2 \leq 4n\ for\ n \geq 2$ ***

$3n + 3 = O(n)$ $\quad *** 3n + 3 \leq 4n\ for\ n \geq 3$ ***

$100n + 6 = O(n)$ $\quad *** 100n + 6 \leq 101n\ for\ n \geq 10$ ***

$10n^2 + 4n + 2 = O(n^2)$ $\quad *** 10n^2 + 4n + 2 \leq 11n^2\ for\ n \geq 5$ ***

$6 * 2^n + n^2 = O(2^n)$ $\quad *** 6 * 2^n + n^2 \leq 7 * 2^n\ for\ n \geq 4$ ***

$2n = O(n^2)\big(also\ O(n)\big), 1000n^3 = O(n^{10})$

Algorithm Analysis : Omega

$\Omega\big(g(n)\big) = \{f(n): \exists c, n_0 > 0 | \forall n \geq n_0\ 0 \leq cg(n) \leq f(n)$

$In\ other\ words\ c * g(n) is\ smaller\ than\ f(n) for\ a\ big\ n$

$2n + 10 = \Omega(n) \quad * 2n + 10 \geq 2n\ for\ n \geq 1 \quad *$

$3n^2 + 2n + 20 = \Omega(n^2) * 3n^2 + 2n + 20 \geq 3n^2\ for\ n \geq 1 *$

$3n^2 + 2n + 20 = \Omega(n)$

$an^2 + bn + c\ is\ \Omega(n^2)$

Algorithm Analysis : Theta

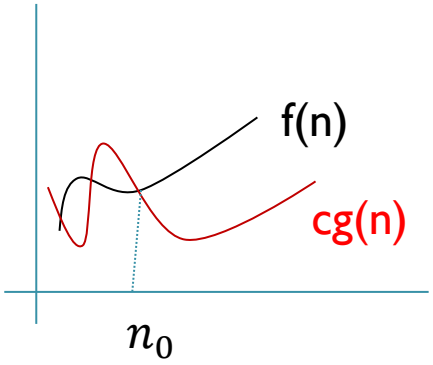
$$\theta\big(g(n)\big) = \{f(n): \exists c_1, c_2, n_0 > 0 | \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

In other words f(n) and g(n) have the same growth rate

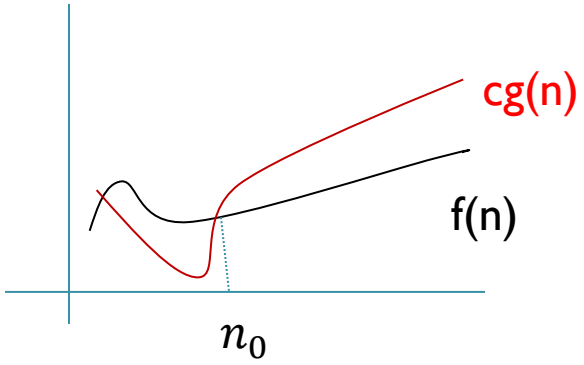
F(n) is $\theta\big(g(n)\big)$ if it is both O(g(n)) and $\Omega(g(n))$

F(n) = 2n+1 is $\theta(n)$

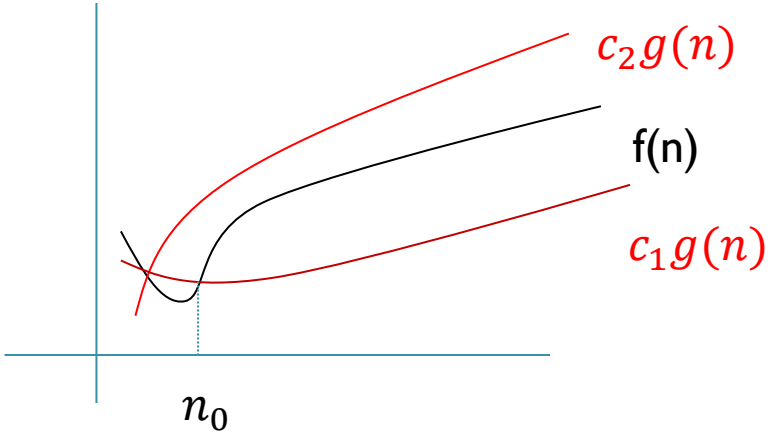
Example



$f(n) = \Omega(g(n))$



f(n) = O(g(n))



$f(n) = \theta(g(n))$

Algorithm Analysis : Little Oh and Little Omega

$o(g(n)) = \{f(n): \forall c > 0\ \exists n_0 > 0 | \forall n \geq n_0, 0 \leq f(n) < cg(n)\}$

f(n)=3n+4 is $o(n^2) \Rightarrow 3n + 4 < cn^2 \Rightarrow\ cn^2 - 3n - 4 > 0$

$\Rightarrow n_0 > 3 + \frac{\sqrt{9 + 16c}}{2c}$

f(n) = 2n $< cn^2 \Rightarrow\ n_0 > \frac{2}{c}$ it is also O(n) but is not o(n)

$\omega(g(n)) = \{f(n): \forall c > 0 \exists n_0 > 0\ | \forall n \geq n_0, 0 \leq cg(n) < f(n)\}$

$f(n) = \frac{n^2}{2}\ is\ \omega(n) but\ is\ not\ \omega(n^2)$ $\frac{n^2}{2} > cn\ , n_0 > 2c$

Some Properties

Transitivity

- $if f(n) = \theta(g(n)) and\ g(n) = \theta(h(n)) then\ f(n) = \theta(h(n))$
- $if f(n) = O(g(n)) and\ g(n) = O(h(n)) then\ f(n) = O(h(n))$
- $if f(n) = \Omega(g(n)) and\ g(n) = \Omega(h(n)) then\ f(n) = \Omega(h(n))$
- $if f(n) = o(g(n)) and\ g(n) = o(h(n)) then\ f(n) = o(h(n))$
- $if f(n) = \omega(g(n)) and g(n) = \omega(h(n)) then\ f(n) = \omega(h(n))$

Reflexivity

- $f(n) = \theta(f(n))$
- $f(n) = O(f(n))$
- $f(n) = \Omega(f(n))$

Symmetry

- $f(n) = \theta(g(n)\ if\ and\ only\ if\ g(n) = \theta(f(n))$

Some Properties(contd)

Transpose Symmetry

- $f(n) = O\big(g(n)\big)\ if\ and\ only\ if\ g(n) = \Omega\big(f(n)\big)$
- $f(n) = o\big(g(n)\big)\ if\ and\ only\ if\ g(n) = \omega\big(f(n)\big)$
- $if\ f(n) = O\big(kg(n)\big)\ for\ constant\ k > 0 \Rightarrow f(n) = O\big(g(n)\big)$
- $if\ f_1(n) = O(g_1(n)), f_2(n) = O(g_2(n))$

  $$\Rightarrow (f_1 + f_2)(n) = O(\max(g_1(n), g_2(n)))$$
- $if\ f_1(n) = O(g_1(n)), f_2(n) = O(g_2(n))$

  $$\Rightarrow f_1(n)f_2(n) = O(g_1(n)g_2(n))$$

Analogy

- $f(n) = O\big(g(n)\big) \approx a \leq b$
- $f(n) = \Omega\big(g(n)\big) \approx a \geq b$
- $f(n) = \theta\big(g(n)\big) \approx a = b$
- $f(n) = o\big(g(n)\big) \approx a < b$
- $f(n) = \omega\big(g(n)\big) \approx a > b$

Runtime Analysis

Simple Sentences : assign – read- write … O(1) (constant)

Simple Rules : + - * / == > >= < <= … O(1) (constant)

Sequence of simple rules/sentences : O(1)

Loops : for – do – while … loop body cost* time loop executes

Conditional sentences : if O(max(T(body1),T(body2))

A part of program that executes many times has the most cost

Running Time Example

- a=b;

  Constant time => $\theta(1)$

- sum=0;

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

    sum+=n;

  Asymptotic Complexity : O(n)

Running Time Example

```
sum=0;

for (i=1;i<=n;i++)
    for (j=1;j<=i;j++)
        sum++;

for (k=0;k<n;k++)
    A[k]=k;

for (i=1;i<=n;i++)
      for(j=1;j<=n;j++)
            sum++;
```

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

$$\sum_{k=0}^{n-1} 1 = n$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n^2$$

Asymptotic Complexity : $O(n^2)$

Running Time Example

```
for(i= l ;i<=n;i++)
   if A[i]>maxVal then
      maxVal = A[i];
      maxpos =i ;
```

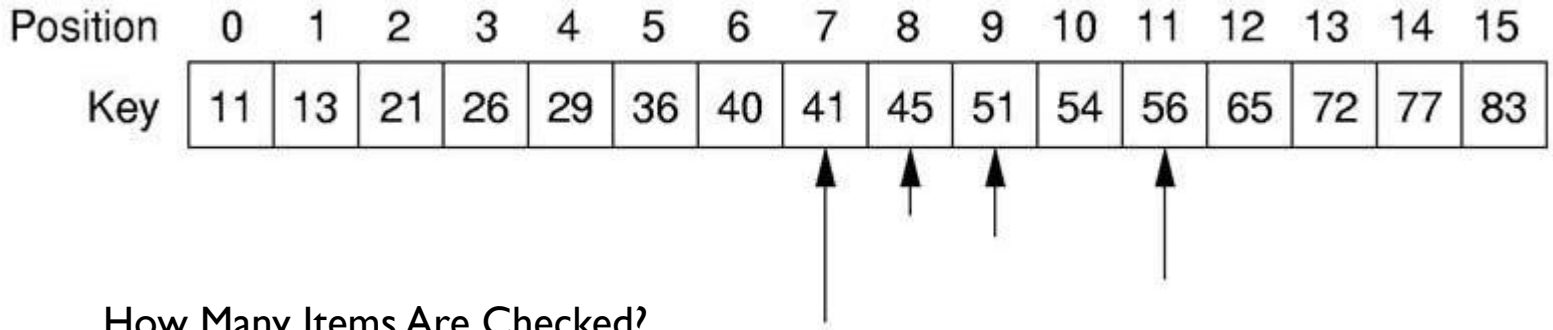
Asymptotic Complexity : O(n)

Running Time Example

```
for (i=1;i<=n;i++)
   for (j=n;j>=i+1;j--)
      if (A[j-1] > A[j] ) then
        temp = A[j-1];
        A[j-1]=A[j];
        A[j]=temp;
      endif
   endfor
endfor
```

Asymptotic Complexity : $O(n^2)$

Binary Search



How Many Items Are Checked?

```
//return position of element in sorted array of size n with value k
int BS(int array[],int n,int k)
{
        int l=-1;
        int r=n; //l and r are beyond array bounds
        while(r!=l+1) // stop when l and r meet
        {
                int i= (l+r)/2;
                if (k<array[i]) r=i; //left half;
                if (k==array[i]) return i; //found
                if (k>array[i]) l=i; //right half;
        }
        return n; // not found;
}
```

Asymptotic Complexity is $O(\log_2 n)$

Max Subsequence Problem

- We are given a sequence of numbers, find a subsequence which has the most sum.
- the numbers can be negative
- if all the numbers are negative the answer is 0

Example : -2,11,-4,13,-5,-2
Answer : A[2]+A[3]+A[4] = 20

We check four different algorithms with $O(n), O(n \log n), O(n^2), O(n^3)$ Complexities.

Algorithm 1

$Given\ A_1, \dots, A_n\ find\ the\ maximum\ value\ of\ A_i, A_{i+1}, \dots, A_j$

int maxSum=0;	O(1)
for (int i=0;i<a.size();i++)	$O(\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i))$
for (int j=i;j<a.size();j++)	$O(\sum_{j=i}^{n-1}(j-i))$
int thisSum=0;	O(1)
for (int k=i;k<=j;k++)	O(j-i)
thisSum+=a[k];	O(1)
if(thisSum > maxSum)	O(1)
maxSum=thisSum;	O(1)
return maxSum;	O(1)

Time Complexity : $O(n^3)$

Algorithm 2

Main Idea : if we have sum of numbers from i to j-1,we can have i to j in constant time so we can omit the inner loop and the complexity decreases to $O(n^2)$

```
int maxSum=0;
for (int i=0;i<a.size();i++)
{
        int thissum=0;
        for (int j=i;j<=a.size();j++)
        {
                thisSum+=a[j];
                if (thisSum>maxSum)
                        maxSum=thisSum;
        }
}
return maxSum;
```

Algorithm 3

Use "divide and conquer" method.

We divide the input into 2 nearly equal parts.

The answer can be found either in right half or left half or a part of it is in left half and a part is in right half.

Example :

Left Half	|||	Right Half
1 -3 5 2	|||	-1 2 6 -2
Left Max: 7	MAX:14		Right Max: 8

We examine each part with recursion. For finding the maximum which is a part of both halves, we find the maximum in the end of left half and the beginning of right half. (O(n))

$$T(n) = \begin{cases} 1 & n=1 \\ 2T(n/2)+n & n>1 \end{cases}$$

$T(n) = O(n \log n)$

Algorithm 4

```
int maxSum=0,thisSum=0;
for (int j=0;j<a.size();j++)
{
        thisSum+=a[j];
        if (thisSum>maxSum)
                maxSum=thisSum;
        else if (thisSum<0)
                thisSum=0;
}
return maxSum;
```

- The sequence can't start with a negative number.
- The sequence can't contain a negative prefix.
- So if the sum of $A_i \; to \; A_j$ is less than 0 we can move i forward to j+1 (and change thisSum to 0)
- So the complexity is O(n)

Summary

We gave 4 algorithms for a problem,with these complexities :

- $O(n^3)$
- $O(n^2)$
- $O(n \log n)$
- $O(n)$

For a problem with n = 10^6 the first algorithm wont answer in our lifetime, but the last one will answer in less than 1 second.

Recursive Algorithms Analysis

T(n) will be recursively computed by T(k) k<n

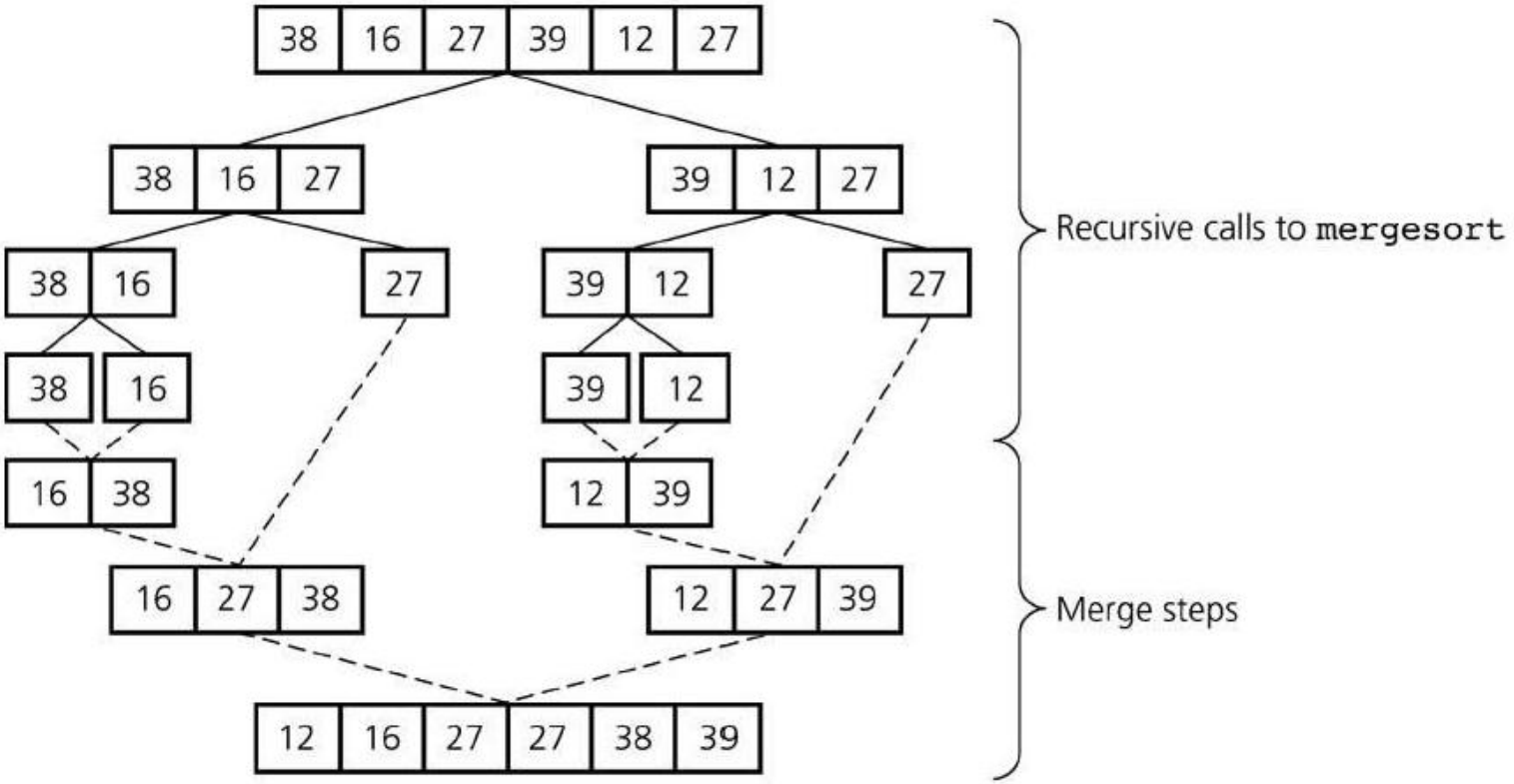
We have return conditions. (like T(0) or T(1))

Methods:

- Guess and prove
- Repeat by replacing recurrence relation
- Master theorem

Guess And Prove

- Guessing the answer and proving it(usually by induction)



Merge Sort

```
MergeSort(A[i..j])
if (i<j)
{
        mid=(i+j)/2;
        MergeSort(A[i..mid]);
        MergeSort(A[mid+1…j]);
        Merge(A[i..mid],A[mid+1,j]);
}
```

$T(n)=2T(n/2)+n$

Proving

Guess : T(n) = O(n log n)

We should prove there exists a c such that T(n) <= cn log n

T(n) = 2T(n/2)+n<=2(c(n/2)log(n/2))+n
T(n) <=cn logn(n/2)+n = cn log(n) – cn log 2 +n = cn log n – cn+n => T(n)<= cn log n

prove $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1$ is O(n)

Wrong guess : T(n) <= cn

Right guess : T(n) <= cn-b b>=1

Changing Variables

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \log n$$

$$T(2^m) = 2T(2^{\frac{m}{2}}) + m$$

$$S(m) = T(2^m)$$

$$S(m) = 2S\left(\frac{m}{2}\right) + m$$

$$S(m) = O(m \log m)$$

$$T(n) = T(2^m) = S(m) = O(m\ log m)$$

$$= O(\log n \log\log n)$$

Repeat by replacing recurrence relation

T(n) = 2T(n/2) + n, T(1) = 1

= 2 (2T(n/4)+n/2)+n = 4T(n/4) + n +n

= 4 (2T(n/8)+n/4)+2n = 8T(n/8)+3n

$=2^i T\left(\frac{n}{2^i}\right) + in$

we impose $n = 2^k$

$= 2^k T(1) + kn$

= n + nlogn

Factorial

```
int factorial(int n){
if(n<=1) return 1;
else return n*factorial(n-1);
}
```

T(n) = T(n-1) +d
T(n) = T(n-2) +d +d
T(n) = T(n-3) +d +d +d
.
.
.
=T(1)+(n-1)*d
=c+(n-1)*d
=O(n)

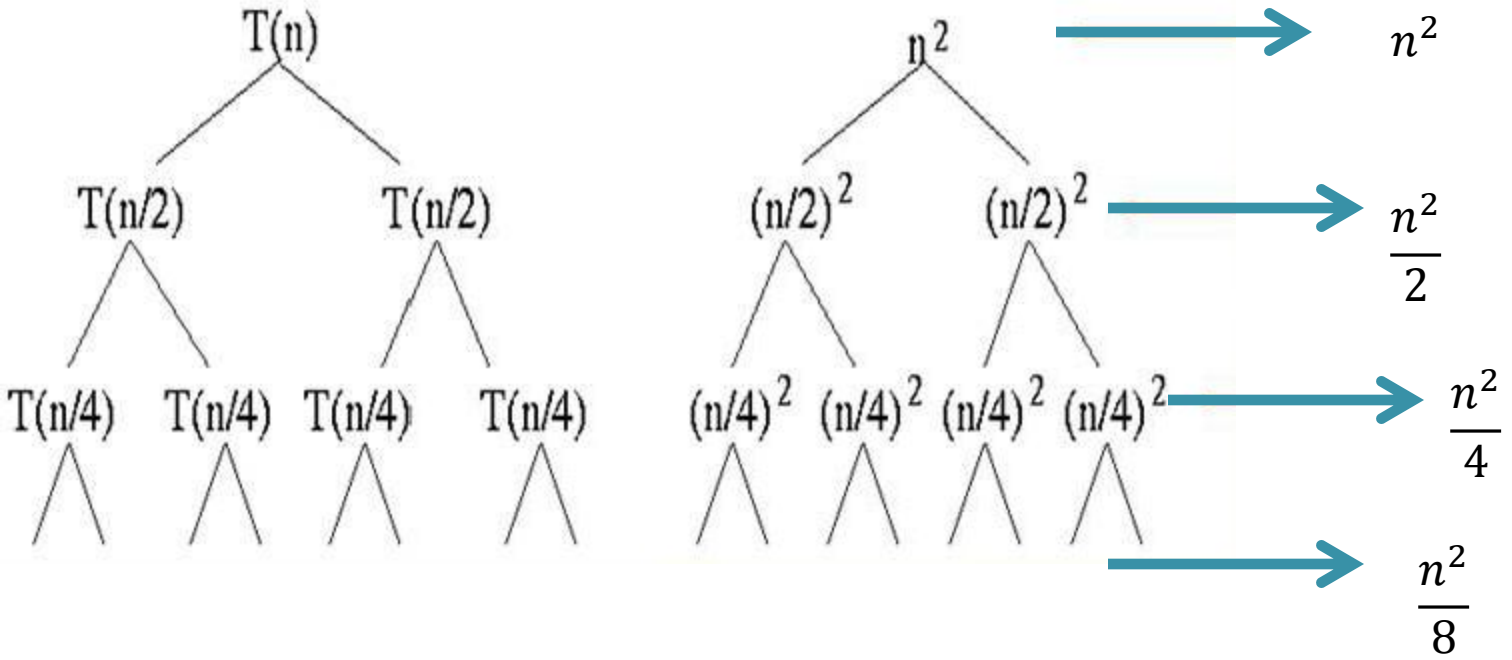
Hanoi Tower

```
void tower(int n, char A,char B,char C)
{
If(n==1)
        move(A,B);
else
{
tower(n-1,A,C,B);
move (A,B);
tower(n-1,C,B,A);
}
}
```

Hanoi(n,A,B,C)::
  Hanoi(n-1,A,C,B)
  Hanoi(1,A,B,C)
  Hanoi(n-1,C,B,A)

T(n)=2T(n-1)+c
=4T(n-2)+2c+c
=8T(n-3)+4c+2c+c
=…
$=2^{n-1}T(1)+(2^{n-2}+\cdots+2+1)c$
$=(2^n-1)c$
$=O(2^n)$

Example



$$T(n) = n^2(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{\log n}})$$

$$T(n) = O(n^2)$$

Master Theorem

T(n) = a T(n/b)+f(n) a>=1 , b>1 $\varepsilon > 0$

$if\ f(n) = O(n^{\log_b a - \varepsilon})$ $T(n) = \theta(n^{\log_b a})$

$if\ f(n) = \theta(n^{\log_b a})$ $T(n) = \theta(n^{\log_b a} \log n)$

$if\ f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right) and$ $T(n) = \theta(f(n))$
$af\left(\frac{n}{b}\right) \leq cf(n)$ (c<1)

Examples

First state

$T(n) = 9T\left(\frac{n}{3}\right) + n \qquad T(1) = c$

$a = 9, b = 3, f(n) = n, n^{\log_b a} = n^2$

$n = O(n^{2-\varepsilon}) \Rightarrow T(n) = \theta(n^2)$

Second state

$T(n) = T\left(\frac{2n}{3}\right) + 1, T(1) = c$

$a = 1, b = \frac{3}{2}, f(n) = 1, \qquad n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = 1$

$\Rightarrow T(n) = \theta(\log n)$

Third state

$T(n) = 3T\left(\frac{n}{4}\right) + nlog\, n \quad T(1) = c$

$a = 3, b = 4, f(n) = nlogn \quad n^{\log_b a} = n^{\log_4 3} = n^{0.793}$

$nlgn = \Omega\left(n^{\log_4 3+\varepsilon}\right), af\left(\frac{n}{b}\right) \leq cf(n)$

$\Rightarrow T(n) = \theta(nlogn)$

Example

$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

$$T(n) = 4T\left(\frac{n}{2}\right) + \log n$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

$$T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{2}$$

$$T(n) = 7T(n/3) + n^2$$

$$T(n) = 8T\left(\frac{n}{3}\right) + 2^n$$

$$T(n) = 3T\left(\frac{n}{3}\right) + n\ logn$$

$$T(n) = T(n-1) + n$$

Example

$T(n) = 7T\left(\frac{n}{2}\right) + n^2$	First state $\theta(n^{\log 7})$
$T(n) = 4T\left(\frac{n}{2}\right) + \log n$	First state $\theta(n^2)$
$T(n) = 4T\left(\frac{n}{2}\right) + n^2$	Second State $\theta(n^2 \log n)$
$T(n) = 3T(\frac{n}{3}) + \frac{n}{2}$	Second State $\theta(n\, logn)$
$T(n) = 7T\left(\frac{n}{3}\right) + n^2$	Third State $\theta(n^2)$
$T(n) = 8T\left(\frac{n}{3}\right) + 2^n$	Third State $\theta(2^n)$
$T(n) = 3T\left(\frac{n}{3}\right) + nlogn$	Can't use master theorem
$T(n) = T(n-1) + n$	Can't use master theorem

Example : Combination

Comb(m,n) = Comb(m-1,n-1)+Comb(m,n-1)

Comb(0,n)=Comb(i,i)=1

```
int Comb(int m,int n)
{
if(m==n || m==0 ) return 0;
return Comb(m-1,n-1)+Comb(m,n-1);
}
```

Example : Combination

$T(m,n)=T(m-1,n-1)+T(m,n-1)$

$T(n,n) = T(n,0)=1$

??????

Examples

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n, T(1) = c$$

$$T(n) = T\left(\frac{n}{q}\right) + T\left(n - \frac{n}{q}\right) + n\ , T(1) = c$$

$$T(n) = T(n-q) + T(q) + n, T(1) = c$$