Algorithms

Recurrence Relation

Recursive Definition

- Recursion เป็นวิธีการนิยามฟังก์ชันที่มีลักษณะถูกนิยามด้วยตัวฟังก์ชันนั้นเอง

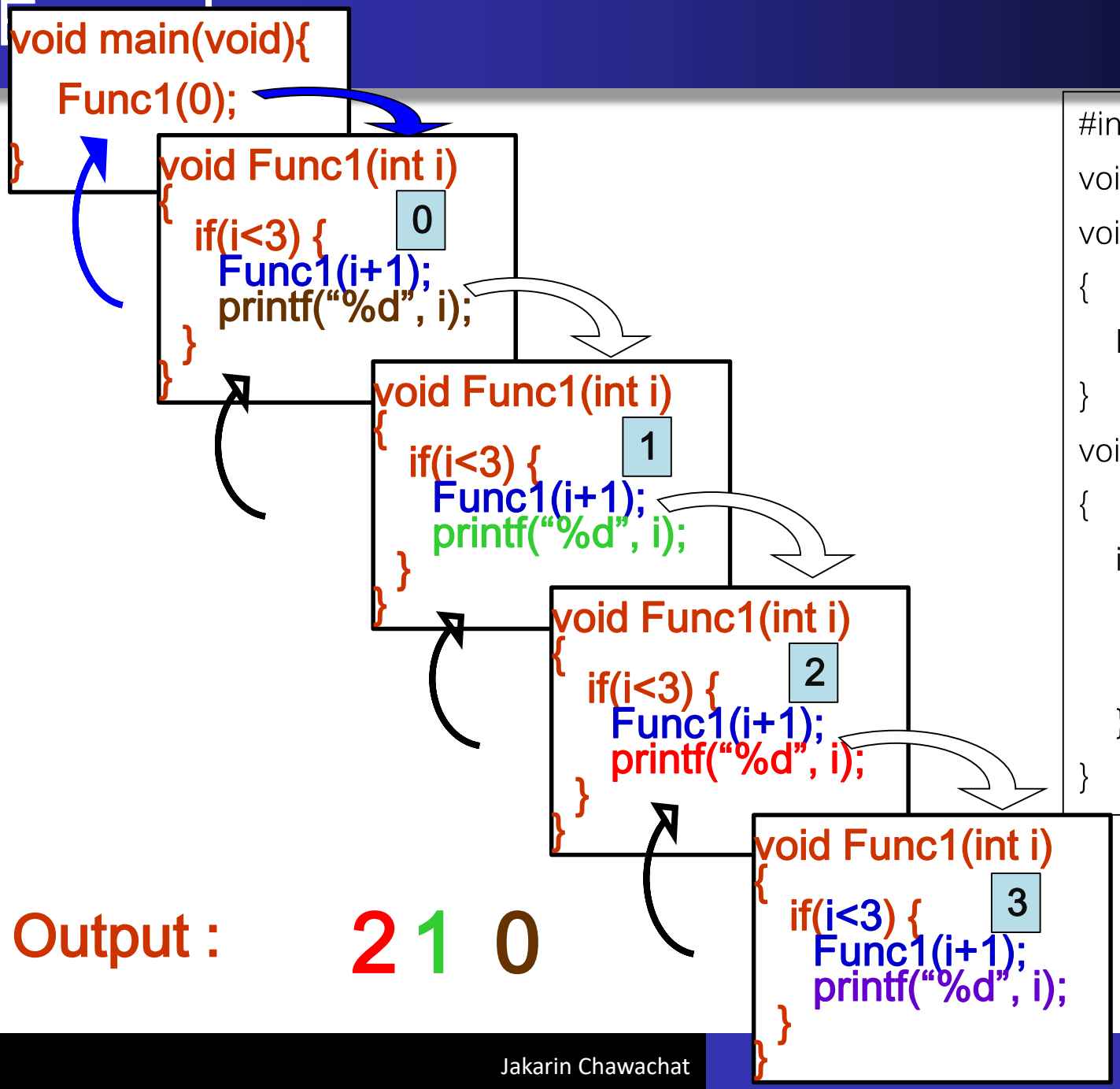
Recursive Function

- Recursive Function เป็นฟังก์ชันที่มีการเรียกตัวเอง
- ใช้กับปัญหาที่มีลักษณะเด่นคือ
  - ปัญหาหลักสามารถแบ่งออกเป็นปัญหาย่อยที่มีลักษณะของปัญหาเหมือนเดิมแต่มีขนาดเล็กลงกว่าเดิม
  - ปัญหาย่อย สามารถแบ่งออกเป็น ปัญหาย่อยๆ ที่มีลักษณะของปัญหาเหมือนเดิมแต่มีขนาดเล็กลงกว่าเดิม
  - ...
  - ปัญหาที่ย่อยที่สุด(base case) หาคำตอบได้ง่าย

Recursion Necessities

- ทุกๆ Recursive algorithm จะต้องประกอบด้วย
  - Base case ที่ไม่มีการเรียกตัวเอง
  - Recursive case

ต้องรับประกันได้ว่าจะต้องไปถึง Base case ถ้าไม่เช่นนั้น อัลกอริทึมจะวนรอบไม่รู้จบ (Infinite loop)




```
#include<stdio.h>
void Func1(int i);
void main(void)
{
    Func1(0);
}
void Func1(int i)
{
    if(i<3) {
        Func1(i+1);
        printf("%d", i);
    }
}
```

Output : 2 1 0

การออกแบบโปรแกรมแบบ Recursive

- การออกแบบโปรแกรมแบบ Recursive

  1. หา Recurrence Relation สำหรับปัญหา (Problem) ที่ต้องการเขียนโปรแกรมเพื่อแก้ไข

  2. เขียน Recursive Function จาก Recurrence Relation ที่หาได้ในข้อ1

ตัวอย่างการออกแบบโปรแกรมแบบ Recursive

ปัญหาการคำนวณค่า factorial

เช่น 5! = 5*4*3*2*1 5! = 5*4!

4! = 4*3*2*1 4! = 4*3!

3! = 3*2*1 3! = 3*2!

2! = 2*1 2! = 2*1!

1! = 1, 0! = 1 1! = 1, 0! = 1

หา Recurrence Relation สำหรับปัญหานี้

$$fac(n) = \begin{cases} 1 & n = 0,1 \\ n * fac(n-1) & n > 1 \end{cases}$$

ตัวอย่างการออกแบบโปรแกรมแบบ Recursive

ปัญหาการคำนวณค่า factorial

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

เงื่อนไขจบการทำงาน

การเรียกใช้ตัวเอง

ตัวอย่างการออกแบบโปรแกรมแบบ Recursive

- การหาค่าของเลขยกกำลัง Power(x,y) ➡ x^y
- วิธีการแก้

  1. หา Recurrence Relation

  2. เขียน Recursive Function

<u>Note</u> เราสามารถเขียนโปรแกรมแบบวน loop คำนวณได้

ตัวอย่างการออกแบบโปรแกรมแบบ Recursive

- การหาค่า 2^4, x=2,y=4

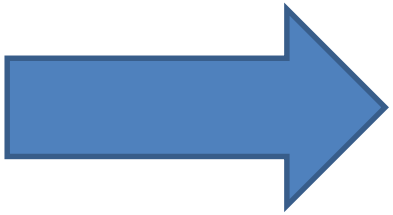
$2^4 = 2*2*2*2$

$2^3 = 2*2*2$

$2^2 = 2*2$

$2^1 = 2$

$2^0 = 1$



$2^4 = 2*2^3$

$2^3 = 2*2^2$

$2^2 = 2*2^1$

$2^1 = 2*2^0$

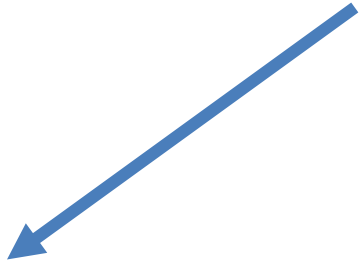
$2^0 = 1$

หา Recurrence Relation

$$\text{Power}(x,y) = \begin{cases} 1 & n = 0 \\ x * Power(y-1) & n > 0 \end{cases}$$

ปัญหาการคำนวณเพื่อหาค่า x^y

```
long power(int x,int y)
{
        if(y==0) return 1;
        else return (x*power(y-1));
}
```



Exercise

Fibonacci sequence ออกแบบอย่างไร

Recursion Vs. Iteration

ปัญหาที่สามารถเขียนแบบ recursive ได้ส่วนใหญ่จะสามารถเขียนแบบ iteration ได้

```
long factorial(int n){
        long ans=1;
        while(n>=1){
                ans=ans*n;
                n=n-1;
        }
        return ans;
}
```

```
int power(int x int y){
        long ans=1;
        while(y>=1){
                ans=ans*x;
                y=y-1;
        }
        return ans;
}
```

Analyzing Recursive Algorithm: Time complexity

- Running time ของ Recursive algorithm อธิบายได้ดังนี้

$$T(n) = \begin{cases} t_A & \text{กรณี } base\ case \\ t_B + t_C & \text{กรณีอื่นๆ} \end{cases}$$

- t_A = เวลาในการทำงานส่วน base case
- t_B = เวลาในการทำงานส่วนการเรียกตัวเองหรือเวลาของการแก้ปัญหาย่อย
- t_C = เวลาในการทำงานอื่นๆ ที่ไม่ใช่เวลาของการแก้ปัญหาย่อย

ตัวอย่างการวิเคราะห์ Selection Sort

```
Selection_sort_recursive(A){
    if(n<=1) return;
    j=FindIndexMax(A[1..n])
    swap(A,n,j)
    Selection_sort_recursive(A[1..n-1])
}
```

O(1) Base case $t_A = T(1) = O(1)$

O(n) ไม่ใช่ Recursive call t_C

$t_B = T(n-1)$

Running Time:

$$T(n) = \begin{cases} O(1) & if\ n = 1 \\ T(n-1) + O(n) & if\ n > 1 \end{cases}$$

```
BinarySearchRecursive(A[left..right])
    if (left > right) return(-1)
    m=(left + right)/2
    if x == A[m] return m;
    If x < A[m]
        return(BinarySearchRecursive (A[left..m-1]))
    else
        return(BinarySearchRecursive (A[m...right]))
```

$t_A = T(1) = O(1)$

ไม่ใช่ Recursive call $t_C = O(1)$

Recursive call $t_B = T(n/2)$

Running Time:

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

Exercise

```
Merge_Sort(A,p,r)
If p < r then
  q= ⌊(p+r)/2⌋
  Merge_Sort(A,p,q)
  Merge_Sort(A,q+1,r)
  Merge(A,p,q,r)
```

$$T(n) = \begin{cases}$$

```
Merge(A,p,q,r)
  i = p, j = q + 1, n = r - p + 1
  for k = 1 to n
      if ((A[i]< A[j]) or (j > r)) and (i≤q)
            B[k] = A[i]
          i = i +1
      else
            B[k] = A[j]
          j = j +1
  for k = 0 to n – 1
        A[p + k] = B[k]
```

Solving Recurrence Relations

- Guess and test method
- Iterative substitution method
- Recursive tree
- Master method

Iterative Substitution Method

วิธีการของวิธี Iterative Substitution คือ

- แทนค่าซ้ำกลับลงไปในสมการ
- พบรูปแบบ (pattern) ของสมการ
- หาคำตอบจากรูปแบบนั้น

Example: merge sort

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

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

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

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

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

= 8T(n/8)+3cn

= 2^kT(n/2^k)+kcn

เราต้องการทำจนหยุด นั่นคือ T(1) ดังนั้นเราจะทำจน 2^k=n นั่นคือ k=logn

T(n) = nT(1)+(logn)cn

= cn + c(nlogn) = O(nlogn)

Example: selection sort

$$T(n) = \begin{cases} O(1) & if\ n = 1 \\ T(n-1) + O(n) & if\ n > 1 \end{cases}$$

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

= T(n-2) + O(n-1) + O(n)

= T(n-3) + O(n-2) +O(n-1) + O(n)

= T(1) + … + O(n-3) + O(n-2) +O(n-1) + O(n)

= O(1) + … + O(n-3) + O(n-2) +O(n-1) + O(n)

$$= \sum_{i=1}^{n} O(i) = O(\sum_{i=1}^{n} i)$$

= O(n(n+1)/2) = $O(n^2)$

Example: binary search

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

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

= T(n/4) + O(1) +O(1)

= T(n/8) + O(1) +O(1) + O(1)

= $T(n/2^k)$ + k O(1)

= T(1) + logn(O(1))

=O(log n)

ระวัง O(1) + O(1) เฉยๆ จะเป็น O(1) แต่ถ้ามีติดพจน์ n เช่นบวกกัน n ครั้ง logn ครั้ง ต้องเขียนเป็น n หรือ log n

Example

$$T(n) = \begin{cases} 0 & if\ n = 2 \\ 2T(\sqrt{n}) + 1 & if\ n > 2 \end{cases}$$

$T(N) = 2T(N^{1/2})+1$

$= 2(2T(N^{1/4})+1)+1$

$= 4T(N^{1/4})+2+1$

$= 8T(N^{1/8})+4+2+1$

$= 2^kT(N^{1/2^k})+2^{k-1}+\ldots+2^2+2^1+2^0$

ต้องทำจนเป็น T(2) นั่นคือ $N^{1/2^k}=2$ ดังนั้น $k = \log\log n$

$T(N) = 2^{\log\log N}(T(2))+2^{\log\log N-1} + \ldots. +2^2+ 2^1 + 2^0$

$= \log N-1$ (ใช้ T(2)=0 และ $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$)

$\log N^{1/2^k} = \log 2$

$1/2^k \log N = 1$

$\log N = 2^k$

$\log\log N = \log 2^k$

$\log\log N = k \log 2$

$\log\log N = k$

สูตรที่ใช้บ่อย

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

Geometric series $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$ for real x>1

Inverse harmonic series

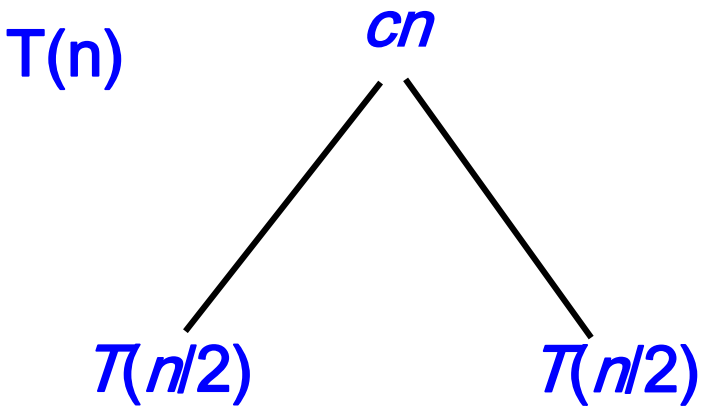
$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \text{ for } |x| < 1$$

Recursion tree method

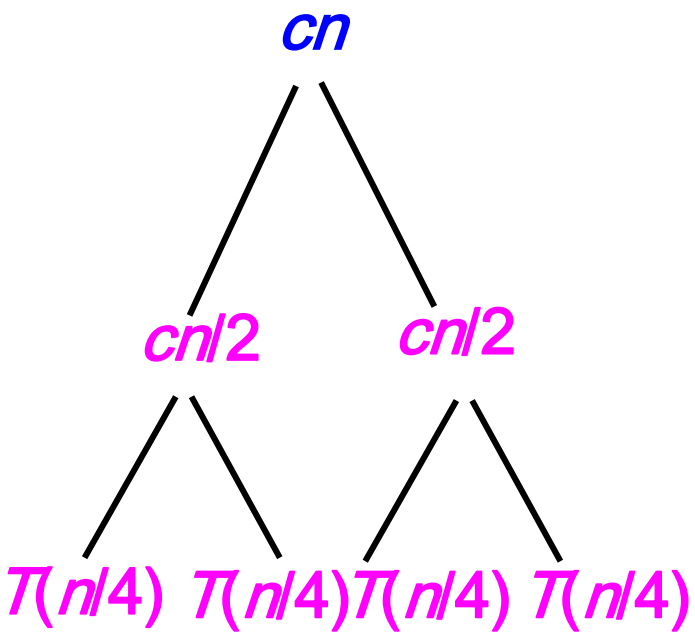
จะวาดต้นไม้แสดงค่าใช้จ่ายแทน โดยแต่ละชั้นของต้นไม้จะคือรอบในการทำซ้ำ

ตัวอย่างเช่น T(n) = 2 T(n/2) + cn

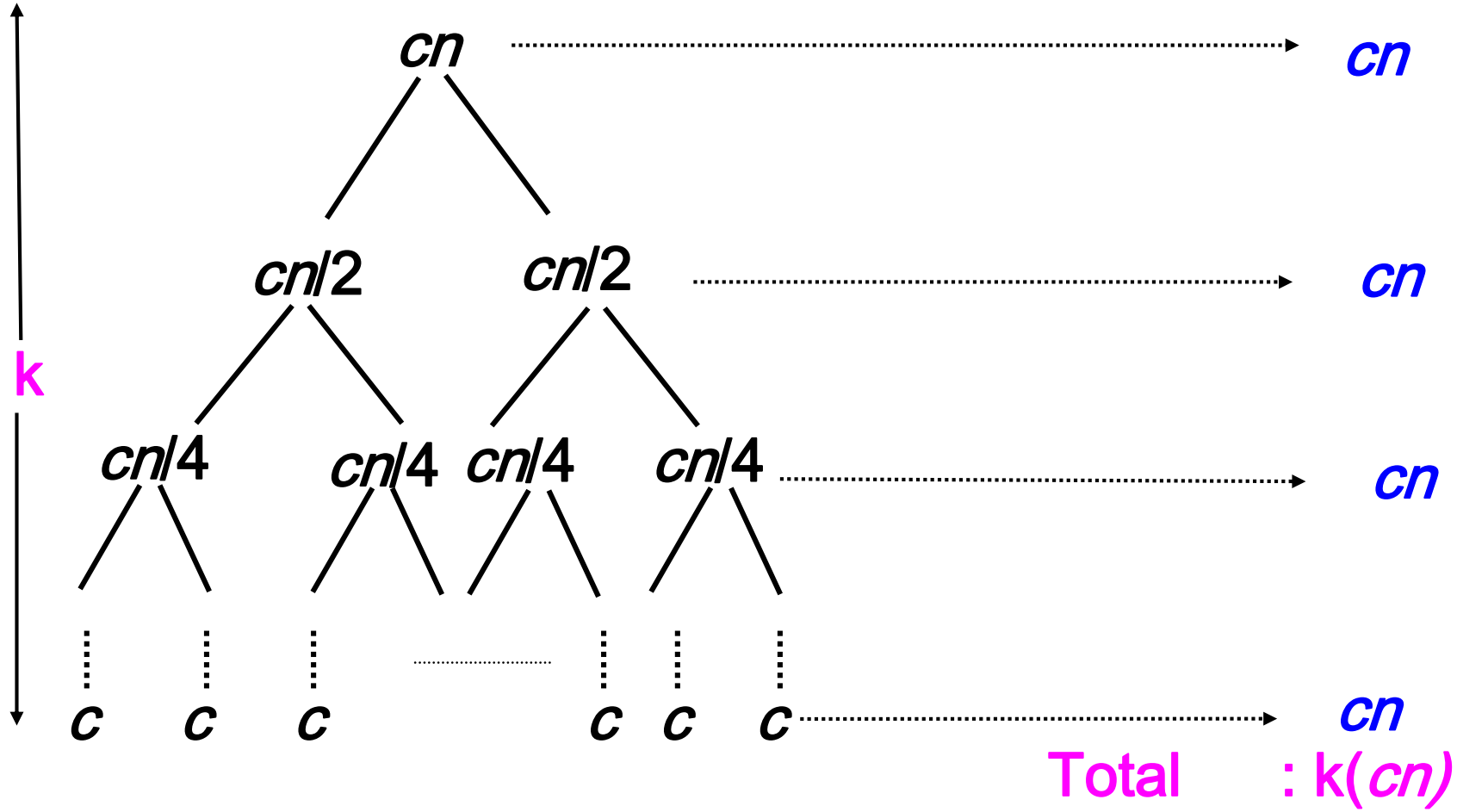
ในชั้นแรก จะมีค่าใช้จ่าย cn และจะมีการเรียก T(n/2) ในชั้นต่อไป 2 ครั้ง



- ในรอบต่อมาจะเสียค่าใช้จ่ายแต่ละโหนดเป็น cn/2 (ซึ่งได้จากการแทนค่า n/2 ลงในสมการ) และแต่ละโหนดจะมีโหนดลูกอีก 2 โหนดที่มี input ขนาดลดลงครึ่งหนึ่ง



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



แบบฝึกหัด

$T(n) = 4\,T(n/2) + n,\ T(1)=1$

แบบฝึกหัด

$T(n) = 3T(n/4) + cn^2$

แบบฝึกหัด

$$T(n) = 2T(n-1) + 1$$