Recursion, Divide and Conquer

21 - 3 - 2015 Alex Poon

Definition

Recursion : A function/procedure calls itself.

Divide & Conquer :
Divide → Divide problem into similar subproblems
Conquer → Solve the subproblems
Combine → Find out the relations between problem & subproblem.

i.e. Use the result of the subproblems to solve the original problem

Function

Similar meaning in Maths

Function : Input -> Process -> Output

e.g. $F(x) = x^2 + 2x + 1$

x is the input, F() is the process, y is the Output

Function in OI

```
int f(int x) {
    int y = x * x;
    return y;
}
......
cout << f(2);
```

Define a function (function name & type/no. of input)

Process of the function

Output of the function

Call function *f* by inputting 2 as x

Recursion

```
int f(int x) {
    int y = f(x);
    return y;
}
```

Calls function *f* itself in the process of function *f* = Recursion

Wait

The function can’t end!!!

The function calls itself repeatly

So when you are writing recursion, a base case is needed

Uses of recursion

Tackle problem which relates to itself or can be divided to same problems with smaller parameter

Example 1 : Calculating factorial

Let f(x) return factorial of x (1 * 2 * ... x)

How to write??

Step 1: Define name & input & output of function

What data type & how many data should be inputted in the function??

What data type should be outputted??

Example 1 : Calculating factorial

Step 2: Define how the function relates to itself i.e. How to use result of subproblems to complete the original problem

Step 3: Define the base case
i.e. When to stop calling itself

Example 1 : Calculating factorial

Name : up to you… e.g. *f*

Input & Output : a integer

How *f* relates to itself : f(x) = f(x - 1) * x

f(x - 1) = 1 * 2 * … x - 1

f(x) = 1 * 2 * … x

When to stop : if (x == 0) return 1 (By definition of factorial)

```
int f(int x) {
   int y = f(x - 1) * x;
   return y;
}
```

Example 1 : Calculating factorial

```
int f(int x) {
    if (x == 0) return 1;
    int y = f(x - 1) * x;
    return y;
}
```

Function Name

Type & number of input

Type of output

Relationship between subproblems and the original problem

Base case

KO. Easy Right??

Writing Recursion

There are 3 Main Points

1. Type & No. of Input & Output
2. Base cases (Determine when to stop recurring)
3. Recurrence relations (How f relates to itself)

Example 2: Fibonacci Sequence

Fibonacci Sequence : 1, 1, 2, 3, 5, 8, 13, 21…

Let F(x) = xth-term of Fibonacci Sequence

Let solve this question by the above steps!!!!

Example 2: Fibonacci Sequence

Question : Find $F(x)$

Input type & no. : 1 integer

Output : integer

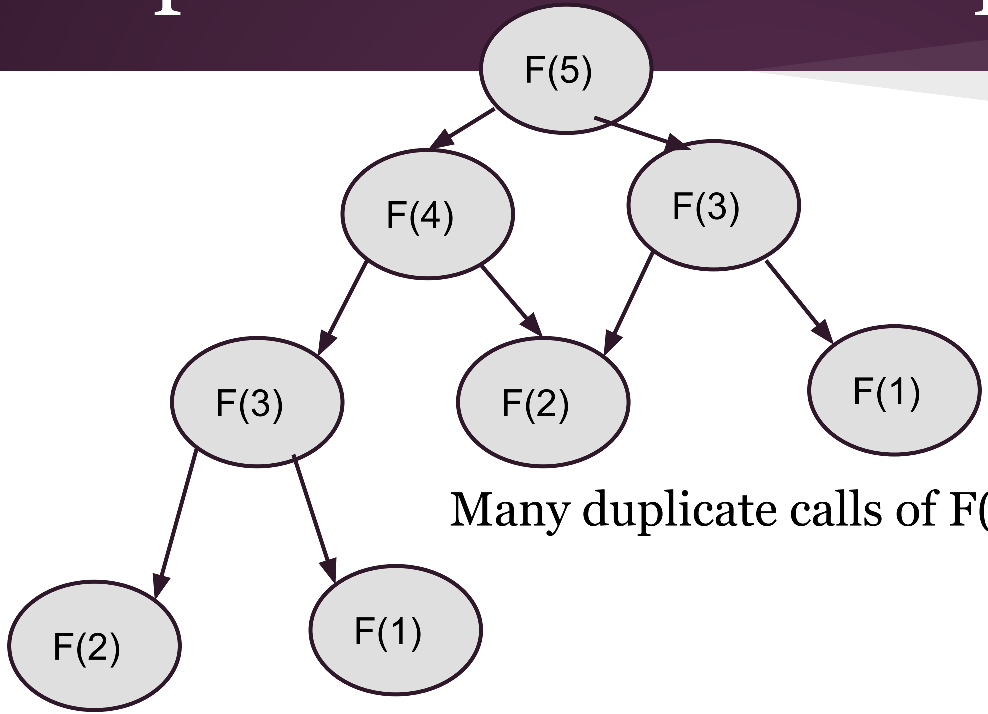
Relation : $F(x) = F(x - 1) + F(x - 2)$

Base case: $F(1) = F(2) = 1$;

Example 2: Fibonacci Sequence

```
int F(int x) {
        if (x == 2 || x == 1) return 1;
    int y = f(x - 2) + f(x - 1);
    return y;
}
```

Example 2: Fibonacci Sequence



Many duplicate calls of F(3), F(2), F(1)

Example 2: Fibonacci Sequenc

```
int F(int x) {
    if (x == 2 || x == 1) return 1;
  if (A[x - 1] == 0) A[x - 1] = f(x - 1);
  if (A[x - 2] == 0) A[x - 2] = f(x - 2);
  return A[x - 1] + A[x - 2];
}
```

Advance to higher level

In example 1, 2, the recurrence relation is simple & by definition.

However, in most case, the recurrence relations is more complicated.

e.g. We need to think how it relates to itself ourself but not by definition.

Example 3: Number Choosing

Question : Choose some number in A[1..n] such that : 1. Sum of chosen numbers <= M

2. Maximize the sum of chosen numbers.

E.g. A[1..4] = {1.2, 2.3, 3.4, 4.4} M = 5.8

Ans = 2.3 + 3.4 = 5.7

Example 3: Number Choosing

Simplifer the problem: Let f(x, y) = The ans of the question if we can choose number from A[1..x] where the sum of them should not greater than y.

i.e. We need to find f(n, M)

Example 3: Number Choosing

Input = 1 integer + 1 real, Output = real

Relationship :

For each number A[x]: we can opt :

1. choose A[x],   2. not to choose A[x]

If we choose A[x], the problem reduce to f(x-1, y-A[x])

If we give up A[x], the problem reduce to f(x-1, y)

Example 3: Number Choosing

So, f(x, M) = max(f(x-1, y), f(x-1,y-A[x]) +A[x]))

Base cases??

```
if (x == 0) {
    if (y >= 0) return 0;
    else return -inf
}
```

-inf denote it is impossible as sum of chosen number > M

Example 3: Number Choosing

```
double f(int x, double y) {
    if (x == 0 && y >= 0.0) return 0.0;
    else if (x == 0 && y < 0.0) return -inf;
   return max(f(x-1, y), f(x-1,y-A[x]) +A[x]));
}
ans = f(n, M);
```

Definition

Procedure: Same as function but there is no output.

i.e. Input + Process

Why no output??

Sometimes we need to compute an array by recursion instead of returning one data

Example 4: Permutation

Given some distinct characters, generate all permutation of them in alphabetical order

Permutation = Different arrangement using all of the given characters

e.g. char[] = {a, b, c, d}

Permutation of it can be : abcd, bcda, dcba....

Example 4: Permutation

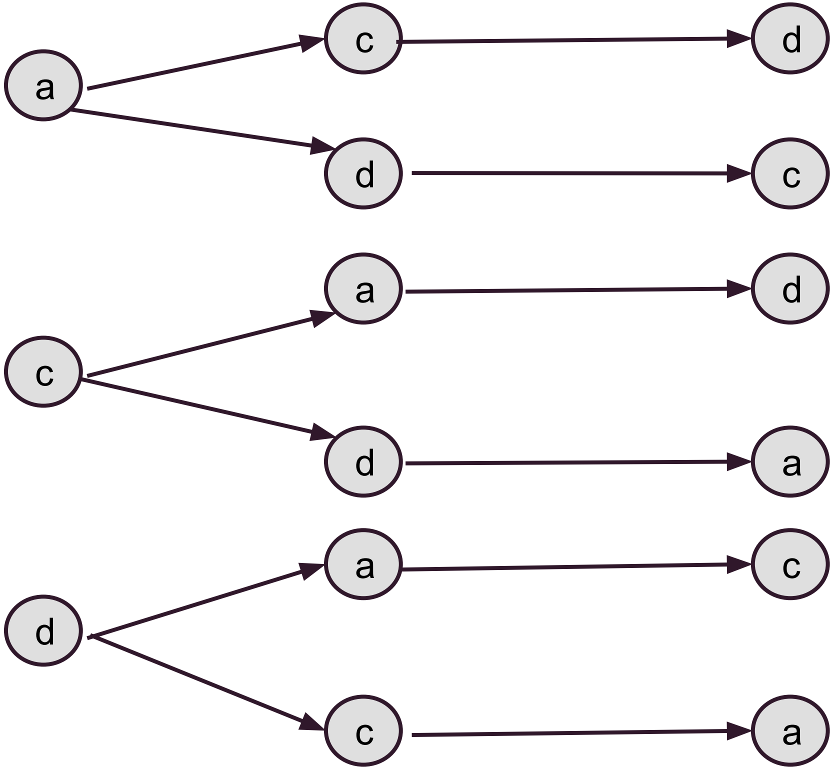
Sample Input: a d c

Sample Output:

acd, adc, cad, cda, dac, dca

First, let’s compute permutation by hands

1st 2nd 3rd Final string



acd

adc

cad

cda

dac

dca

Example 4: Permutation

Why recursion??

We fill the first character, then the second…

fill(1) -> fill(2) -> fill(3) …. fill(n)

Relationship between them??

In each step, we fill the position with all unuesd given characters

Example 4: Permutation

Then what is the base case??

After filling the *n* character, stop & output the permutation generated

Example 4: Permutation

```
void fill(int x) {
    if (x == n + 1) write(output);
    else {
     for(int i = 0; i < n; i++)
     if (used[i] == 0) {
          used[i] = 1;
          output[x] = S[i];
          fill(x + 1);
          used[i] = 0;
     }
    }
}
read(S[]); n = strlen(S);
sort(S); fill(1);
```

fill(x) = we try to fill the xth character of the output string with unused character

Base case, fill(x) = filling the *x*th character. So x == n + 1 denotes we have filled all n characters

Check has the *i*th character used

*i*th character has not been used, fill it to the *x*th character of the output string

The *x*th character is filled, fill the *x+1*th character

**After we fill the S[i] to output[x], that mean we have finished using S[i], i.e. S[i] is no longer used. Set used[i] back to 0.

Importance of Permutation

Many question in combinatorial optimization can be solved by generating permutations

e.g. HKOI 14/15 Senior Q3 - Secret Message

You can just generate all possible permutations of the original sequence & test whether it can encrype to the target sequence~ 25 marks get

Example 5: Combination

Question : HKOI 2009 JQ2 -- Dictionary

There are N words. The words contain only capital letter.

You need to choose some words such that:
1. All 26 letters are included in at least 1 chosen word
2. Choose as few words as you can.

Output which words you will choose

Example 5: Combination

Sample input:

```
5
ABCDEFGHIJ
LMNOPQRSTUV
KLMNOP
QRSTUVW
XYZ
```

Sample output:

```
ABCDEFGHIJ
KLMNOP
QRSTUVW
XYZ
```

Constraints: N <= 20

Example 5: Combination

Possible Algorithm : Try all combination!!!

For each words, you can opt:

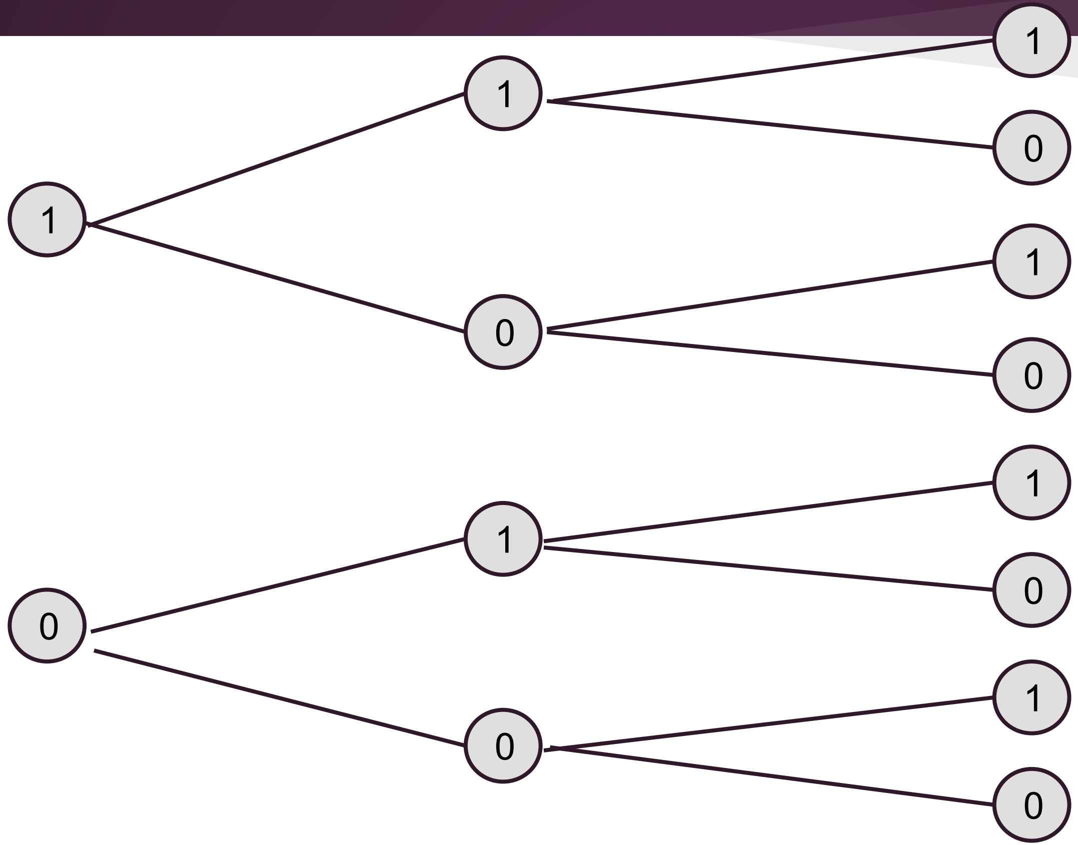
1. choose it, 2. not to choose it

All combination = 00000, 00001, 00010....11101, 11110, 11111

0 = not to choose, 1 = choose

e.g. 11101 = choose the 1, 2, 3, 5 words

1st word 2nd word 3rd word



111
110
101
100
011
010
001
000

ALL Combinations is generated

Huarry!!!

Example 5: Combination

Again, figure out the recurrence relation & base case!

Let opt(x) = we are deciding choose ot not to choose the *x*th word

We will call opt(1) -> opt(2) -> opt(3) -> ... opt(N)

Each time we call, we should consider choose x & not to choose $x \to 2$ Cases

Example 5: Combination

Base case : After we opt choose & not choose the *N*th word.
We stop opt the next word and check :

1. Are all 26 letters included in the choose words
2. How many words we have chosen

Example 5: Combination

```
void opt(int x) {
    if (x == n + 1) check();
    else {
        choose[x] = 1;
        opt(x + 1);
        choose[x] = 0;
        opt(x + 1);
    }
}
......
opt(1);
```

Base case, similarly, opt(x) = deciding whether uses/not to use the *x*th word. x == n + 1 denotes we have decided all *n* words.

Case 1: Choose the word → choose[x] = 1

Case 2: not to choose the word → choose[x] = 0

After deciding the *x*th word, decide the next word~

Importance of Combination

Again, many question in combinatorial optimization can be solved by generating combinations.

Remember example 3? It is a question about combinations actually, the approach in ex. 5 can be solve it as well.

Many question in HKOI can be partly solve by this approach : 2015SQ1, 2015SQ4, 2014JQ4……

Brief Summary

2 main uses of recursion have been introduced

**Main point in writing recursion, D&C solution

1. Base cases
2. Recurrence relations

Let’s see more examples → Try to be familiar with how recurrence relations can be setted

Example 6: Big mod

Given B, P, M. Find B^P mod M

B = 5, P = 3, M = 7, 5^3 = 125 mod 7 = 6

Easy ~ for(i, 1, p) ans = (ans * B) mod M;

**Note that
(a*b) mod M = (a mod M * b mod M) mod M

But…… B, P, M <= 10^9

Example 6: Big mod

Can B^P divides into subproblems

B^(P - 1) * B ? …… That what we do in for-loop

How about B^(P / 2) * B^(P / 2) ??

e.g. B^8 = B^4 * B^4

B^4 = B^2 * B^2

B^2 = B * B

Example 6: Big mod

Then, we only need to calculate B, B^2, B^4, B^8. Only lg (P) calculations are needed

Let f(x) = B^x, then f(x) = f(x/2) * f(x/2)

....

Wait, How about B^5

Example 6: Big mod

B^5 = B^2 * B^3, B^3 = B^2 * B^1

B^2 = B * B

We need to calculate B^5, B^3, B^2 for 2 times. Seems it is not efficient to calc B^2 and B^3 from B

How about B^1023 → B^512 * B^511…. Well it takes much more time to calc both B^511, B^512 from B

Example 6: Big mod

Combine the 2 thought

1. B^(P/2)*B(P/2) 2. B^(P-1) * B

Obviouly: B^512 = B^511 * B

So. B^1023 = B^511 * (B^511 * B)

We only need to calc B^511 this time, better~

Example 6: Big mod

```
int f(int x) {
if (x == 1) return B mod M;
else {
  int y = f(x / 2) mod M;
  if (x mod 2 == 0) return (y * y) mod M;
  else return (y * y * B) mod M;
}
```

Base case

Case 1: even number

Case 2: odd number

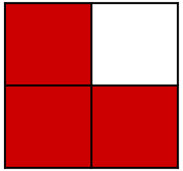
Example 7: L - Piece

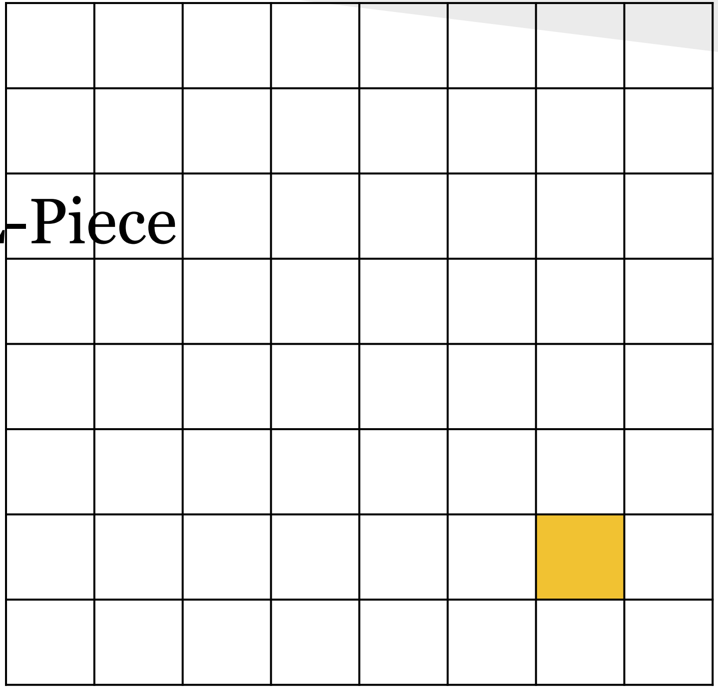
Q: Given a $2^n * 2^n$ grip

One cell is occipied

Find a way to fill the grip with L-Piece

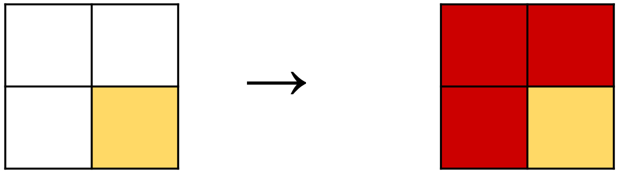
L-Piece can be rotated





Example 7: L - Piece

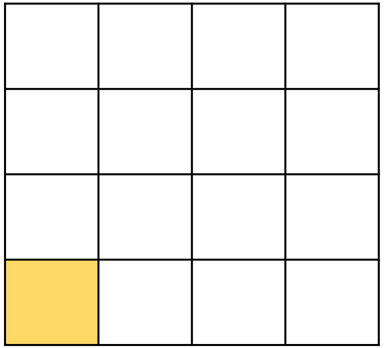
Let’s consider the easiest case first 2x2 grip



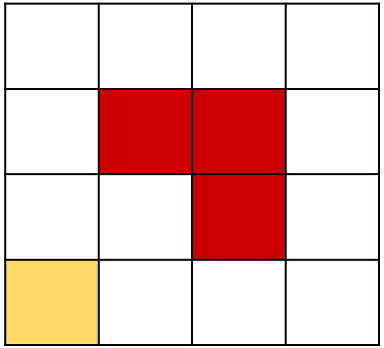
Easy!! Fill the 3 empty cells with a L-piece

Example 7: L - Piece

Let’s see 4 * 4 grip this time



What if we put a L-piece in the middle →



Example 7: L - Piece

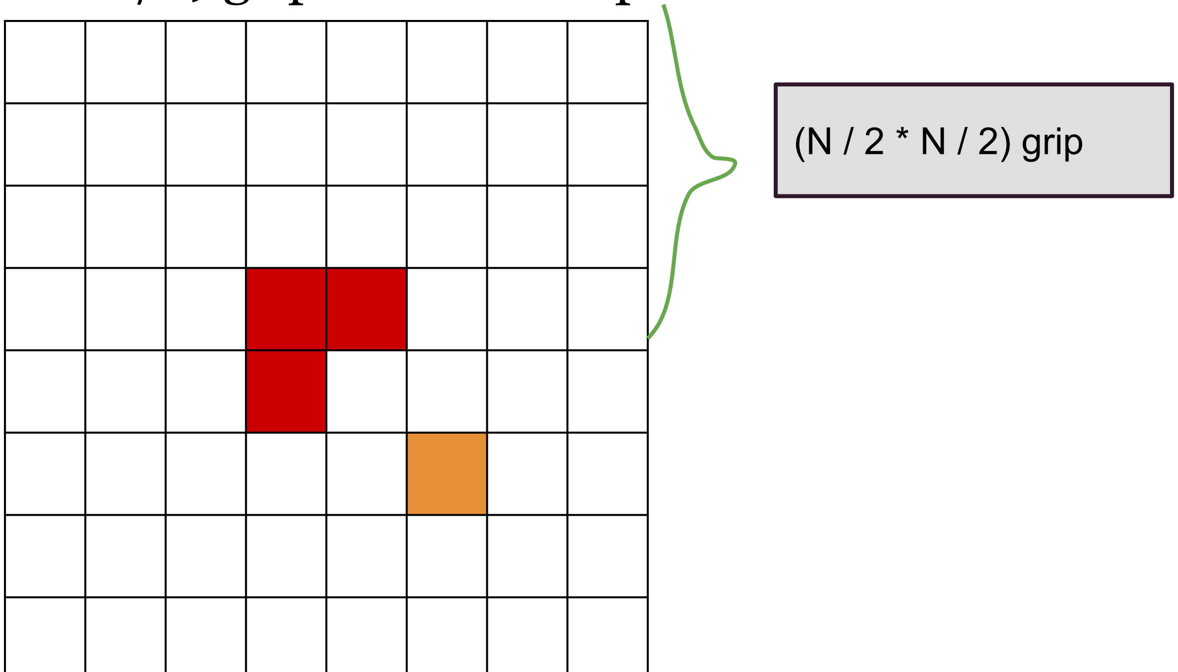
1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Oh… four 2x2 grip with 1 occupied cell fo

Then, fill all of the the 2x2 grips. KO!!

Example 7: L - Piece

Generally, we can put a L-piece in the middle of a N*N grip, then four (N/2 * N/2) grips with 1 occupied cell are formed



Example 7: L - Piece

1. Find where is the occupied cells (in which region? upper left, upper right, lower left, lower right)
2. Fill the middle with a L-piece such that all four region contain exactly 1 occupied cell
3. Recursion → fill the four divided regions
4. Until we meet the base case (2*2 grip)

Summary

Divide & Conquer steps:

1. Divide problem into subproblems
2. Find out the recurrence relation
3. Find out base cases

*Usually use for solve combinatorial optimization problems
→ Choose the best one in many combination/permutation

Advanced Level

An important topic in OI, dynamic programming (DP) is based on the thought of Divide and Conquer

Learn it on later training lesson :)

Practise problems

HKOJ : 01048, 30098, J092, 01046, 01003, 20374

plus partial score of many questions including: S114, S131, S133, S134, S151, S153, S103, J144 etc.

Reference

http://www.hkoi.org/training2012/files/recursion_dc.pdf

Last year training material by Bill Kwok Tsz Piu

Lunch Time~ :)

Add Oil & Have Fun in Minicomp~