***Divide and Conquer***

***Steps:***

* ***Divide:*** Divide the question into several smaller questions, and the format of smaller question is the same as the original one but just with smaller scale.
* ***Conquer:*** Solve questions recursively. As long as size of sub-question is small enough, then stop recursion and solve the question directly.
* ***Combine:*** Combine all solutions into the original solution.

***Recursive Case:***

When the question is big enough and need to be solved recursively, then such situation is called *Recursive Case*.

***Base Case:***

When the recursive case is small enough and no need further recursion, then the recursion is called *“Touch Bottom”*.

*(Attention: we need to solve the case that just the same as the original case, also sometimes we need to solve the case that totally different from the original one.)*

***Recurrence:***

Recurrence is closely related with Divide and Conquer, since by using the Recurrence, we can easily describe the Running Time of Divide and Conquer Algorithm. Recurrence is the *Equal Expression* or *Non-Equal Expression*, it uses function to describe the function with smaller input.

***Example - Maximum Sub-Array Question:***

*Situation:*

Assume that you get the chances to invest the Chemical Company. Just like the Product you produced, the stock price of this company is unstable. You are allowed to buy the stock of the company in sometime, and can sold out someday. You can choose to buy or sell out the stock before the end of day. Also, in order to compensate this limit, you can get knowledge about the price of the stock in the future.Your aim is to get the maximum profit.

*The Stock Price just like the picture 4 - 1:*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Day* | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| *Price* | 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 |
| *Chg* |  | 13 | -3 | -25 | 20 | -3 | -16 | -23 | 18 | 20 |
| *Day* |  |  |  | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| *Price* |  |  |  | 94 | 106 | 101 | 79 | 94 | 90 | 97 |
| *Chg* |  |  |  | -7 | 12 | -5 | -22 | 15 | -4 | 7 |

Normally, we may can solve the issue by buying the stock at the low point and selling the stock at the high point. But it can not be realized actually, since between the low point and high point, there has possibility that several minus value exists between low point and high point.

Therefore, in order to solve the problem, we can abstract the issue into situation that it wants us to get *Continuous Maximum Array for Change Row*.

* Random Continuous Maximum Array for Change Row is located at A [ low, mid ] -> low =< i <= j <= mid.
* Random Continuous Maximum Array for Change Row is located at A [ mid + 1, high ] -> mid =< i <= j <= high.
* Random Continuous Maximum Array for Change Row is located at A [ low, high ] -> low =< i <= mid <= j.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 13 | -3 | -25 | 20 | -3 | -16 | -23 | 18 | 20 | -7 | 12 | -5 | -22 | 15 | -4 | 7 |

For example, in the stock array, then we get the A[1, 2, 3, ..., 14, 15, 16], therefore the Longest Continuous Maximum Array of this array is A[8, 9, 10, 11] = 18 + 20 - 7 + 12 = 50 - 7 = 43.

***Main Thinking:***

* For each n length array, divide the array from Middle Index, and get two independent array, from Low Index to Middle Index, from Middle Index to High Index.
* Take the random array into consideration, and locate the middle array element. Calculate sum from Middle Index to Low Index, sum from (Middle + 1) Index to High Index, and sum from Low Index to High Index diversely.
* Get the Largest Continuous Sum among Three Sums and keep the Largest Continuous Sum and record its Index into memo.

***Algorithm:***

***Function:***

*Get\_Max\_Crossing\_Sub\_Array (A, low, high)*

*int A[ ];*

*int low;*

*int high;*

*Middle = ( low + high ) / 2;*

*Left\_Sum = -8;*

*Sum = 0;*

*For ( i = Middle; i >= low; i -- )*

*{*

*Sum += A[ i ];*

*IF ( Sum > Left\_Sum )*

*{*

*Left\_Sum = Sum;*

*Left\_Low = i;*

*}*

*}*

*Right\_Sum = -8;*

*Sum = 0;*

*For ( i = Middle + 1; i <= high; i++ )*

*{*

*Sum += A[ i ];*

*IF ( Sum > Right\_Sum )*

*{*

*Right\_Sum = Sum;*

*Right\_High = i;*

*}*

*}*

*Return ( Left\_Low, Right\_High, Left\_Sum + Right\_Sum );*

***Function:***

*Find\_Maximum\_Sub\_Array(A, low, middle, high)*

*Int A[ ];*

*Int low;*

*Int middle;*

*Int high;*

*IF ( high == low )*

*{*

*RETURN A [ low ];*

*}*

*ELSE IF ( middle = ( low + high ) / 2 )*

*{*

*( Left\_Low, Left\_High, Left\_Sum ) = Find\_Maximum\_Sub\_Array(A, low, , middle );*

*( Right\_Low, Right\_High, Right\_Sum ) = Find\_Maximum\_Sub\_Array(A, middle + 1, high );*

*( Cross\_Low, Cross\_High, Cross\_Sum ) = Find\_Maximum\_Sub\_Array(A, low, middle, high );*

*IF ( Left\_Sum >= Right\_Sum && Left\_Sum >= Cross\_Sum )*

*{*

*RETURN ( Left\_Low, Left\_High, Left\_Sum );*

*}*

*ESLE IF ( Right\_Sum >= Left\_Sum && Right\_Sum >= Cross\_Sum )*

*{*

*RETURN ( Right\_Low, Right\_High, Right\_Sum );*

*}*

*ESLE IF ( Cross\_Sum >= Left\_Sum && Cross\_Sum >= Right\_Sum )*

*{*

*RETURN ( Cross\_Low, Cross\_High, Cross\_Sum );*

*}*

*}*