Divide & Conquer Optimization
and
Knuth Optimization

This optimization for dynamic programming solutions uses the concept of divide and conquer. It is only applicable for the following recurrence:

$$\mathrm{dp}[i][j] = \min_{k<j}\{dp[i-1][k] + \mathrm{C}[k][j]\}$$

This optimization for dynamic programming solutions uses the concept of divide and conquer. It is only applicable for the following recurrence:

$$\mathrm{dp}[i][j] = \min_{k<j}\{dp[i-1][k] + \mathrm{C}[k][j]\}$$

$$\min[i][j] \leq \min[i][j+1]$$

$$\min[i][j] \text{ is the smallest k that gives the optimal answer}$$

This optimization reduces the time complexity from $O(KN^2)$ to $O(KNlog\ N)$

Given an array of N integers, partition the array into K disjoint sub-arrays
so that the sum over the beauty of each sub-array is maximized
The beauty of a sub-array is defined as the bitwise OR of the numbers in the sub-array
1 <= N, K <= 5000

Given an array of N integers, partition the array into K disjoint sub-arrays
so that the sum over the beauty of each sub-array is maximized
The beauty of a sub-array is defined as the bitwise OR of the numbers in the sub-array
1 <= N, K <= 5000

```
for(int groupNo=1; groupNo<=K; groupNo++){
    for(int pos=1; pos<=N; pos++){
        for(int endOfLast=0; endOfLast<pos; endOfLast++){
            int ret = dp[groupNo-1][endOfLast] + rangeOR(endOfLast+1, pos)
            if(ret > dp[groupNo][pos]){
                dp[groupNo][pos] = ret;
                opt[groupNo][pos] = endOfLast;
            }
        }
    }
}
```

Given an array of N integers, partition the array into K disjoint sub-arrays
so that the sum over the beauty of each sub-array is maximized
The beauty of a sub-array is defined as the bitwise OR of the numbers in the sub-array
$1 <= N, K <= 5000$

$$Opt(groupNo, pos) <= Opt(groupNo, pos+1)$$

```
for(int groupNo=1; groupNo<=K; groupNo++){
    for(int pos=1; pos<=N; pos++){
        for(int endOfLast=opt[groupNo][pos-1]; endOfLast<pos; endOfLast++){
            int ret = dp[groupNo-1][endOfLast] + rangeOR(endOfLast+1, pos);
            if(ret > dp[groupNo][pos]){
                dp[groupNo][pos] = ret;
                opt[groupNo][pos] = endOfLast;
            }
        }
    }
}
```

Opt(groupNo, pos) <= Opt(groupNo, pos+1)

Opt(groupNo, pos) >= Opt(groupNo, pos-1)

Divide and Conquer Optimization

```
void compute(int groupNo, int L, int R, int OptL, int OptR){
    if(L > R) return;
    int mid = (L+R)/2;

    dp[groupNo][mid] = 0;
    int optNow = optL;

    for(int endOfLast=OptL; endOfLast<=optR && endOfLast<mid; endOfLast++){
        int ret = dp[groupNo-1][endOfLast] + rangeOR(endOfLast+1, mid);
        if(ret > dp[groupNo][mid]){
            dp[groupNo][mid] = ret;
            optNow = endOfLast;
        }
    }
    compute(groupNo, L, mid-1, OptL, optNow);
    compute(groupNo, mid+1, R, optNow, OptR);
}

for(int groupNo=1; groupNo<=K; groupNo++) compute(groupNo, 1, N, 1, N);
```

How to understand whether D&C property will hold?

When will Opt(groupNo, pos) <= Opt(groupNo, pos+1) hold true?

A sufficient condition is satisfying Quadrangle Inequality

Cost(L, j + 1) - Cost(L, j) <= Cost(k, j + 1) - Cost(k, j) for any(L < k < j) For Max Query

Cost(L, j + 1) - Cost(L, j) >= Cost(k, j + 1) - Cost(k, j) for any(L < k < j) For Min Query

Name	Original Recurrence	Sufficient Condition of Applicability	Original Complexity	Optimized Complexity	Links
Convex Hull Optimization1	$dp[i] = min_{j<i}\{dp[j] + b[j] \star a[i]\}$	$b[j] \geq b[j+1]$ ~~optionally~~ $a[i] \leq a[i+1]$	$O(n^2)$	$O(n)$	1 2 3 p1
Convex Hull Optimization2	$dp[i][j] = min_{k<j}\{dp[i-1][k] + b[k] * a[j]\}$	$b[k] \geq b[k+1]$ ~~optionally~~ $a[j] \leq a[j+1]$	$O(kn^2)$	$O(kn)$	1 p1 p2
Divide and Conquer Optimization	$dp[i][j] = min_{k<j}\{dp[i-1][k] + C[k][j]\}$	$A[i][j] \leq A[i][j+1]$	$O(kn^2)$	$O(knlogn)$	1 p1
Knuth Optimization	$dp[i][j] = min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$	$A[i,j-1] \leq A[i,j] \leq A[i+1,j]$	$O(n^3)$	$O(n^2)$	1 2 p1