CSP

n<=10	O(n!)-O(n*n!)，枚举全排列
n<=20	O(2^n)-O(n*2^n)，状压DP，枚举是否选择
n=40	O(2^(n/2))，折半搜索
n<=100	O(n^3)，Floyd，区间DP,网络流，高斯消元
n<=1000	O(n^2)-O(n^2*logn)，二维DP，枚举，二分+枚举区间检查
n<=50000	O(n*sqrt(n))，分块，二分图匹配
n<=100000	O(n*logn^2)，二维树状数组，树剖，CDQ，二分图匹配
n<=500000	O(n*logn)，线段树，二分，树状数组，最短路，树形DP
n<=1e6	O(n)，一般是线性DP，超纲一点就是斜率优化，贪心，单调栈
n<=1e18	O(logn)，矩阵快速幂优化转移，数论
n<=1e18	O(1)，结论题

提示：某数可能很大===> **long long（int类型)/long double**

归并排序

```
// 最开始调用，n是最初的数组长度
void mergeSort(int arr[], int start, int end, int n) {
    if (start >= end) {
        return;
    }
    int mid = (start + end) / 2;
    mergeSort(arr, start, mid, n);
    mergeSort(arr, mid+1, end, n);
    merge(arr, start, mid, end, n);
}

void merge(int arr[], int start, int mid, int end, int n) {
    int * result = (int *)malloc(sizeof(int) * n);
    int i = start, j = mid + 1, k = 0;
    while (i <= mid && j <= end) {
        if (arr[i] < arr[j]) {
            result[k++] = arr[i++]; // 从大到小，j++
        } else {
            result[k++] = arr[j++]; // 从大到小i++
        }
    }
    if (i == mid + 1) {
        while (j <= end) {
            result[k++] = arr[j++];
        }
```

```
    }
    if (j == end + 1) {
        while (i <= mid) {
            result[k++] = arr[i++];
        }
    }
    for (j = 0, i = start; j < k; i++, j++) {
        arr[i] = result[j];
    }
}
```

计算最大非零段

$A_1, A_2, \cdots, A_n$ 是一个由 n 个自然数（非负整数）组成的数组。我们称其中 $A_i, \cdots, A_j$ 是一个非零段，当且仅当以下条件同时满足：

- $1 \le i \le j \le n$；
- 对于任意的整数 k，若 $i \le k \le j$，则 $A_k > 0$；
- $i = 1$ 或 $A_{i-1} = 0$；
- $j = n$ 或 $A_{j+1} = 0$。

下面展示了几个简单的例子：

- $A = [3, 1, 2, 0, 0, 2, 0, 4, 5, 0, 2]$ 中的 4 个非零段依次为 $[3, 1, 2]$、$[2]$、$[4, 5]$ 和 $[2]$；
- $A = [2, 3, 1, 4, 5]$ 仅有 1 个非零段；
- $A = [0, 0, 0]$ 则不含非零段（即非零段个数为 0）。

现在我们可以对数组 A 进行如下操作：任选一个正整数 p，然后将 A 中所有小于 p 的数都变为 0。试选取一个合适的 p，使得数组 A 中的非零段个数达到最大。若输入的 A 所含非零段数已达最大值，可取 $p = 1$，即不对 A 做任何修改。

// 70分

用两次归并排序，

// 100分

借用岛屿情况来分析这个题。考虑p足够大的情况，所有的数都被海水淹没了，只有0个岛屿。然后，海平面逐渐下降，岛屿数量出现变化。每当一个凸峰出现，岛屿数就会多一个；每当一个凹谷出现，原本相邻的两个岛屿就被这个凹谷连在一起了，岛屿数减少一个。使用数组cnt[]，cnt[i] 表示海平面下降到i时，岛屿数量的变化。

差分法是最简洁的解题程序。数组元素d[i]中存储该元素被替换为0时，划分数变化的差分值。最大值则只需要从其前缀和（程序中实际为后缀和）中找出最大值就是所要的结果。

程序代码中，STL算法函数unique()用来去除相邻重复的元素。

语句“a[0] = a[n + 1] = 0;”用来设置边界值，起辅助计算作用，可以简化程序代码。

```
#include <bits/stdc++.h>

using namespace std;

const int N = 500000;
const int M = 10000;
int a[N + 2], d[M + 1];

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    a[0] = a[n + 1] = 0;

    n = unique(a, a + n + 2) - a - 1;
```

```
    memset(d, 0, sizeof d);
    // *****
    for (int i = 1; i < n; i++)
        if (a[i - 1] < a[i] && a[i] > a[i + 1]) d[a[i]]++;
        else if (a[i - 1] > a[i] && a[i] <a[i + 1]) d[a[i]]--;
    // *****
    int ans = 0, sum = 0;   // 差分前缀和即为答案
    for (int i = M; i >= 1; i--)
        sum += d[i], ans = max(ans, sum);

    printf("%d\n", ans);

    return 0;
}
```

矩阵的子矩阵的和——前缀和

```
int n;
scanf("%d", &n);

int m = n+1;
int **predSum = new int *[m];// 行前缀和
int **array = new int *[m];// 保存数组值
for (int i = 0; i < m; i++) {
    predSum[i] = new int [m];// 留一个0
    array[i] = new int [m];
}

for (int i = 1; i <= n; i++) {
    predSum[i][0] = 0;
    for (int j = 1; j <= n; j++) {
        scanf("%d", &array[i][j]);
        predSum[i][j] = predSum[i][j-1] + array[i][j];
    }
}
// 计算上边缘为top、下边缘为down、左边缘为left、右边缘为right的矩阵和
// 注意是left-1
int sum = 0;
for (int k = top; k <= down; k++) {
                sum += predSum[k][right] - predSum[k][left-1];// l-1不是l
            }
```

计算线段和mod r 不用遍历，分段算法

```
long long sum = 0;
void calculate(int r, int begin, int end, int fx) {
    // [begin, end] 中按 模r相同 进行分组计算
    int gx = begin / r;
    if (gx == end / r) {
        sum +=  (end - begin + 1)*abx(gx - fx);
    } else {
        // 跨越
        int front = (r*(gx+1) - 1);
        sum += (front - begin + 1)*abx(gx - fx);
        int j;
        for (j = front + 1; j + r <= end && j <= end; j += r) {
```

```
            sum += r*abx(j/r - fx);
        }
        if (j <= end) {
            sum += (end - j + 1) * abx(j/r - fx);
        }
    }
}
```

计算多项式除法

```
// leftequal = xishu[k]*qx - f(k-1)
void calqx(int k, int n) {
    // xishu[k]是最高系数为k的多项式
    // leftequal是最高系数为k+n-1的多项式
    // f(k-1)是最高系数为k-1的多项式
    // leftequal
    int begin = n-1+k;
    for (int i = begin-k; i>= 0; i--, begin--) {
        int tmp = leftequal[begin];
        // 更新left参数
        for (int j = k; j >= 0; j--) {
            leftequal[j+i] -= xishu[k][j]*tmp;
        }
    }
    //now leftequal取负就是f(k-1)
}
```