# 倍增

1. 引例

1.朴素 2.二分前缀和 3.倍增

例如序列：a1 , a2 … an

假设a9时是最大的k，满足 <= T

按照倍增算法是这样实现的：

1. a1 <= T （增加的步长为1）
2. a1 + (a2 + a3) <= T（增加的步长为2）
3. a1 + a2 + a3 + (a4 + a5 + a6 + a7) <= T （增加步长为4）
4. a1 + a2 + a3 + a4 + a5 + a6 + a7 + (a8 + a9 + … + a14 + a15) > T

（增加步长为8 此时>T 需要缩小）

5）a1 + a2 + a3 + a4 + a5 + a6 + a7 + (a8 + a9 + a10 +a11) > T （步长/2 此时步长为4）

6）a1 + a2 + a3 + a4 + a5 + a6 + a7 + (a8 + a9) <= T （找到答案）

**此题倍增算法实现**：计算前缀和O(n) + 找答案O(logn)

1）令p = 1 , k = 0 , sum = 0 (p为当前步长 k为答案 sum为当前的和)

2）若sum + S[k + p] – S[k] <= T ， 则sum+=S[k + p] – S[k] , k += p , p \*= 2;

否则sum + S[k + p] – S[k] > T，则p/=2

3）p=0时，k为答案

1. Genius ACM

题目：给定一个整数 M，对于任意一个整数集合 S，定义“校验值”如下:从集合 S 中取出 M 对数(即 2\*M 个数，不能重复使用集合中的数，如果 S 中的整数不够 M 对，则取到不能取为止)，使得“每对数的差的平方”之和最大，这个最大值 就称为集合 S 的“校验值”。现在给定一个长度为 N 的数列 A 以及一个整数 T。我们要把 A 分成若干段，使得 每一段的“校验值”都不超过 T。求最少需要分成几段。

分析：

1)如何使每对数的校验值最大：贪心 （对一个当前需要取校验值的区间排序后S1 , S2 , … , St ;需要取M对数使得“差的平方的和”最大。）

选择最大St , 最小S1凑一对，次大和S2凑一对……

该区间的校验值为: 或者数不够取到m

时间复杂度:区间有t个数的话 ， 时间复杂度为O(tlogt) 长度为N的序列进行log(N)次处理,总的复杂度为O(NN)

2)若要段尽量少的话，则需要每段尽量长，那么和上面的引例相似，只不过把前缀和换成了校验值，然后进行倍增算法。

3)每次求校验值的时候可以使用类似于归并排序的方法，只对新增长度排序，合并两段，时间复杂度会降到O(NlogN);

#include <iostream>

#include <algorithm>

**using** **namespace** std;

**typedef** **long** **long** LL;

**const** **int** N = 1e6 + 10;

**int** n, a[N], b[N], m, c[N];

LL k;

**long** **long** ans;

**void** merge (**int** l, **int** mid, **int** r){

**int** i = l, j = mid;

**for** (**int** k = l; k <= r; k++){

**if** (j > r || i < mid && a[i] < a[j])   b[k] = a[i++];

**else**    b[k] = a[j++];

    }

}

**bool** check(**int** l, **int** mid, **int** r){

**for** (**int** i = mid; i <= r; i++)   a[i] = c[i];

    sort (a + mid, a + r + 1); //对新增序列排序

    merge (l, mid, r);  //归并增加的序列

    LL sum = 0;

**for** (**int** i = 1; i <= (r - l + 1) >> 1 && i <= m; i++ ){ //计算校验值

        sum += (b[r - i + 1] - b[l + i - 1]) \* 1ll \*(b[r - i + 1] - b[l + i - 1]);

    }

**if** (sum <= k){

**for** (**int** i = l; i <= r; i++) a[i] = b[i];

**return** **true**;

    }

**return** **false**;

}

**int** main(){

    ios::sync\_with\_stdio(**false**);

    cin.tie(0); cout.tie(0);

**int** T;

    cin >> T;

**while**(T--){

        cin >> n >> m >> k;

**for** (**int** i = 1; i <= n; i++) cin >> c[i];

**int** p, l, r;

        p = l = r = 1, ans = 0;  //p为倍增的步长

        a[1] = c[1];

**while** (r <= n){

**if** (p == 0){ //不能再增加

                ans++; p = 1;

                l = ++r; a[l] = c[l];

            }**else** **if** (r + p <= n && check(l, r + 1, r + p)){ //check 判断校验值是否大于k

                r += p; p <<= 1;

            }**else** p >>= 1;

        }

        cout << ans << endl;

    }

**return** 0;

}

三、ST算法 （区间最值即RMQ问题） 书上很清楚

**void** ST\_prework(){

**for**(**int** i = 1 ; i <= n ; i ++) f[i][0] = a[i];

**int** t = **log**(n) / **log**(2) + 1;

**for**(**int** j = 1 ; j < t ; j ++){

**for**(**int** i = 1 ; i <= n - (1<<j) + 1 ; i ++)

            f[i][j] = max(f[i][j - 1] , f[i + (1 << (j - 1))][j - 1]);

}

**int** ST\_query(**int** l , **int** r){

**int** k = **log**(r - l + 1) / **log**(2);

**return** max(f[l][k],f[r - (1<<k) + 1][k]);

}