**// ========== LIS ==========**

int C[maxn], B[maxn], dp[maxn];//C是辅助阵列

int LIS(int n){//n表示作LIS的数据范围

int mmin, mmax, ret=0;

memset(C, 0x3f, sizeof(C));//inf=0x3f3f3f3f

for(int i=1; i<=n; i++){//阵列从1开始

mmin=1, mmax=i;

while(mmin<mmax){//二分

int mid=(mmin+mmax)/2;

if(C[mid]<B[i]) mmin=mid+1;

else mmax=mid;

}

dp[i]=mmin;

C[mmin]=B[i];

ret=max(ret, dp[i]);//更新答案

}

return ret;

}

**// ========== LIS RUJIA=======**

// A[1] ~ A[N]

for (int i=2;i<=N+1;i++) g[i]=INF;

for (int i=1;i<=N;i++) {

int k=lower\_bound(g+2,g+N+2,A[i]) - g;

// non-decreasing: 用upper\_bound

dp[i] = k-1; // 以A[i]结尾的LIS长度

Ans = max(Ans, dp[i]);

g[k] = A[i];

}

**// ========== 四边形优化 ==========**

dp(i,j)=min{dp(i,k-1),dp(k,j)}+w(i,j)(i≤k≤j)

(min也可以改为max) 如果满足：

1.区间包含的单调性: if (i≤i'<j≤j') then w(i',j)≤w(i,j')

2. 四边形不等式: if (i≤i'<j≤j') then w(i,j)+w(i',j')≤w(i',j)+w(i,j')

// 再定义 s(i,j) 表示 m(i,j) 取得最优值时对应的下标(即 i≤k≤j 时,k 处的 w 值最大,则 s(i,j)=k) 我们有s(i,j)单调 ie.

s(i,j-1)≤s(i,j)≤s(i+1,j)

故而dp(i,j)=min{dp(i,k-1)+dp(k,j)}+w(i,j) 且s(i,j-1)≤k≤s(i+1,j) (min可以改为max)

**// ========== 状态压缩 ==========**

// O(3^n)

for (int S = 1; S < (1<<n); S++)

for (int S0 = (S-1)&S; S0; S0 = (S0-1)&S)

int S1 = S-S0; //S 被拆成两个集合 S0 & S1

**// ========== Min - Max ==========**

struct State{//状态

int score;//当前状态的打分, 对大分玩家越有利分越高

inline bool isFinal();//判断是否为终局的函数

inline void expand(int player,

vector<State>& children);

//拓展子节点的函数

};

int alphabeta(State& s, int player, int alpha, int beta){

//0是大分玩家, 1是小分玩家

if(s.isFinal()) return s.score;

vector<State> children;

s.expand(player, children);//生成儿子

for(auto child:children){

int v=alphabeta(child, player^1, alpha, beta);//对儿子状态打分

!player ?alpha = max(alpha, v)

:beta=min(beta, v);

//选取最有利的情况

if(beta<=alpha) break;

//alpha-beta剪枝

}

return !player?alpha:beta;

}

**// ========== 快速读入 ==========**

// 快速读入: Eg. read(N);

inline void read(int &x) {

x = 0; static char c;

for (; !(c >= '0' && c <= '9'); c = getchar());

for (; c >= '0' && c <= '9'; x = x \* 10 + c - '0', c = getchar());}

**// ========== 统计逆序对 ==========**

int cnt=0; // 逆序对个数

int a[MaxN], c[MaxN];

void MergeSort(int l, int r) {

// if a[1]~a[N] then r = N+1

int mid, i, j, tmp;

if (r>l+1) {

mid = (l+r)/2;

MergeSort(l, mid);

MergeSort(mid, r);

tmp = l;

for (i=l,j=mid; i<mid && j<r; )

if (a[i]>a[j])

// 如果算上(x,x)则改成a[i]>=a[j]

{

c[tmp++] = a[j++];

cnt += mid-i;

}

else

c[tmp++] = a[i++];

if (j<r) for (; j<r; j++) c[tmp++] = a[j];

else for (; i<mid; i++) c[tmp++] = a[i];

for (i=l; i<r; i++) a[i]=c[i];

}

}

int main() { a[1] = 4; a[2] = 2; a[3] = 1;

MergeSort(1,4); cout << cnt;}