**Mini heap**

**priority\_queue<int, vector<int>, greater<int> > my\_min\_heap;**

**segment tree**

**build**

**void build(int node,int st,int en)**

**{**

**if (st==en)**

**{**

**treemin[node]=v[st];**

**treemax[node]=v[st];**

**return ;**

**}**

**else**

**{**

**int mid=st+(en-st)/2;**

**build(node\*2+1,st,mid);**

**build(node\*2+2,mid+1,en);**

**// treemin[node]=min(treemin[node\*2+1],treemin[node\*2+2]);**

**treemax[node]=max(treemax[node\*2+1],treemax[node\*2+2]);**

**}**

**}**

**Update single elment**

**void updatesingleelment(int node,int st,int en,int idx,int val)**

**{**

**if (st==en)**

**{**

**v[idx]+=val;**

**treemin[node]+=val;**

**treemax[node]+=val;**

**}**

**else**

**{**

**int mid=st+(en-st)/2;**

**if (st<=idx&&idx<=mid)**

**{**

**updatesingleelment(node\*2+1,st,mid,idx,val);**

**}**

**else**

**{**

**updatesingleelment(node\*2+2,mid+1,en,idx,val);**

**}**

**treemax[node]=max(treemax[2\*node+1],treemax[2\*node+2]);**

**treemin[node]=min(treemin[2\*node+1],treemin[2\*node+2]);**

**}**

**}**

**Find minimum element in range**

**ll querymin(int node,int st,int en,int l,int r)**

**{**

**if (r<st||en<l)**

**{**

**return 1e18;**

**}**

**if (l<=st&&en<=r)**

**{**

**return treemin[node];**

**}**

**int mid=st+(en-st)/2;**

**ll p1=querymin(node\*2+1,st,mid,l,r);**

**ll p2=querymin(node\*2+2,mid+1,en,l,r);**

**return min(p1,p2);**

**}**

**Update range naïve**

**void updaterangenaive(int node,int st,int en,int l,int r,int val)**

**{**

**if (r<st||en<l)**

**{**

**return ;**

**}**

**if (st==en)**

**{**

**treemax[node]+=val;**

**treemin[node]+=val;**

**return ;**

**}**

**int mid=st+(en-st)/2;**

**updaterangenaive(node\*2+1,st,mid,l,r,val);**

**updaterangenaive(node\*2+2,mid+1,en,l,r,val);**

**treemax[node]=max(treemax[node\*2+1],treemax[node\*2+2]);**

**treemin[node]=min(treemin[node\*2+1],treemin[node\*2+2]);**

**}**

**Update range using lazy (faster)**

**void updateRange(int node, int st, int en, int l, int r, int val)**

**{**

**if(lazy[node]!=0)**

**{**

**treemax[node]+=lazy[node];**

**// treemin[node]+=lazy[node];**

**if(st!=en)**

**{**

**lazy[node\*2+1]+=lazy[node];**

**lazy[node\*2+2]+=lazy[node];**

**}**

**lazy[node]=0;**

**}**

**if(st>en||st>r||en<l){return;}**

**if(st>=l&&en<=r)**

**{**

**// treemin[node]+=val;**

**treemax[node]+=val;**

**if(st!=en)**

**{**

**lazy[node\*2+1]+=val;**

**lazy[node\*2+2]+=val;**

**}**

**return;**

**}**

**int mid=st+(en-st)/2;**

**updateRange(node\*2+1,st,mid,l,r,val);**

**updateRange(node\*2+2,mid+1,en,l,r,val);**

**//treemin[node]=min(treemin[node\*2+1],treemin[node\*2+2]);**

**treemax[node]=max(treemax[node\*2+1],treemax[node\*2+2]);**

**}**

**Find maximum or minmum or summation in range using lazy(faster)**

**ll queryRangemax(int node,int st,int en,int l,int r)**

**{**

**if(st>en||st>r||en<l){return -1e18;}**

**if(lazy[node]!=0)**

**{**

**treemax[node]+=lazy[node];**

**if(st!=en)**

**{**

**lazy[node\*2+1]+=lazy[node];**

**lazy[node\*2+2]+=lazy[node];**

**}**

**lazy[node]=0;**

**}**

**if(st>=l&&en<=r){return treemax[node];}**

**int mid=st+(en-st)/2;**

**ll p1=queryRangemax(node\*2+1,st, mid, l, r);**

**ll p2=queryRangemax(node\*2+2, mid + 1, en, l, r);**

**return max(p1,p2);**

**}**