// Splay

const int MAXN=5e5;

namespace splay

{

struct node

{

int val,ch[2],father,sz,cnt;

inline void clean() {sz=cnt=father=val=ch[0]=ch[1]=0;}

}tree[MAXN+48];int tot=0,root;

inline void print(int cur)

{

if (tree[cur].ch[0]) print(tree[cur].ch[0]);

cout<<tree[cur].val<<endl;

if (tree[cur].ch[1]) print(tree[cur].ch[1]);

}

Inlinevoidupdate(intcur){if(cur) tree[cur].sz=tree[cur].cnt+tree[tree[cur].ch[0]].sz+tree[tree[cur].ch[1]].sz;}

inline void rotate(int x)

{

int y=tree[x].father,z=tree[y].father;

int k=(tree[y].ch[1]==x);

tree[z].ch[tree[z].ch[1]==y]=x;

tree[y].ch[k]=tree[x].ch[k^1];

tree[x].ch[k^1]=y;

tree[x].father=z;tree[y].father=x;tree[tree[y].ch[k]].father=y;

update(y);update(x);

}

inline void splay(int x,int target)

{

while (tree[x].father!=target)

{

int y=tree[x].father,z=tree[y].father;

if (z!=target)

((tree[z].ch[1]==y)^(tree[y].ch[1]==x))?rotate(x):rotate(y);

rotate(x);

}

if (!target) root=x;

}

inline int find(int val)

{

int cur=root,res=0;

for (;;)

{

if (tree[cur].val==val) {res+=tree[tree[cur].ch[0]].sz;splay(cur,0);break;}

if (val<tree[cur].val)

{

if (!tree[cur].ch[0]) break;

cur=tree[cur].ch[0];

}

else

{

res+=tree[tree[cur].ch[0]].sz+tree[cur].cnt;

if (!tree[cur].ch[1]) break;

cur=tree[cur].ch[1];

}

}

return res;

}

inline int findx(int k)

{

int cur=root;

for (;;)

{

if (tree[tree[cur].ch[0]].sz>=k) {cur=tree[cur].ch[0];continue;}

if (tree[tree[cur].ch[0]].sz+tree[cur].cnt>=k) return tree[cur].val;

k-=tree[tree[cur].ch[0]].sz;k-=tree[cur].cnt;

cur=tree[cur].ch[1];

}

}

inline void insert(int val)

{

if (!root)

{

root=++tot;

tree[root].ch[0]=tree[root].ch[1]=0;

tree[root].sz=tree[root].cnt=1;tree[root].val=val;

return;

}

int cur=root,k;

while (tree[cur].val!=val)

{

k=(val>tree[cur].val);

if (tree[cur].ch[k]) cur=tree[cur].ch[k]; else break;

}

if (tree[cur].val==val)

tree[cur].cnt++,tree[cur].sz++;

else

{

int cc=cur;tree[cur].ch[k]=++tot;cur=tot;

tree[cur].val=val;tree[cur].cnt=tree[cur].sz=1;

tree[cur].ch[0]=tree[cur].ch[1]=0;tree[cur].father=cc;

}

splay(cur,0);

}

inline int find\_pre(int val)

{

if (!tree[root].ch[0]) return -2147483647;

int cur=tree[root].ch[0];

while (tree[cur].ch[1]) cur=tree[cur].ch[1];

return cur;

}

inline int find\_nxt(int val)

{

if (!tree[root].ch[1]) return 2147483647;

int cur=tree[root].ch[1];

while (tree[cur].ch[0]) cur=tree[cur].ch[0];

return cur;

}

inline void del(int val)

{

find(val);

if (tree[root].cnt>1)

{

tree[root].cnt--;tree[root].sz--;

return;

}

if (!tree[root].ch[0] && !tree[root].ch[1])

{

tree[root].clean();root=0;

return;

}

if (!tree[root].ch[0] || !tree[root].ch[1])

{

int k=(tree[root].ch[1]==0);

int oldroot=root;root=tree[root].ch[k^1];

tree[root].father=0;

tree[oldroot].clean();return;

}

int cur=find\_pre(val);splay(cur,root);

int oldroot=root;root=tree[root].ch[0];

tree[root].ch[1]=tree[oldroot].ch[1];tree[tree[root].ch[1]].father=root;

tree[root].father=0;

update(root);tree[oldroot].clean();

}

}

// 可持久化Treap

struct Treap

{

int left,right;

int val,priority;

int sz;

}tree[500048\*50];

int root[500048];

int tot=0,rtot=0;

inline void update(int root)

{

tree[root].sz=1;

if (tree[root].left!=-1) tree[root].sz+=tree[tree[root].left].sz;

if (tree[root].right!=-1) tree[root].sz+=tree[tree[root].right].sz;

}

int Create(int val,int priority,int left,int right)

{

tree[++tot]=Treap{left,right,val,priority,0};

update(tot);

return tot;

}

void cop(int &a,int b)

{

if (b==-1) a=-1; else a=Create(0,rand(),-1,-1),tree[a]=tree[b];

}

int LeftSize(int root)

{

int res=1;

if (tree[root].left!=-1) res+=tree[tree[root].left].sz;

return res;

}

void split(int root,int &a,int &b,int key)

{

if (root==-1)

{

a=-1;b=-1;

return;

}

if (tree[root].val>=key)

{

cop(b,root);

split(tree[root].left,a,tree[b].left,key);

update(b);

}

else

{

cop(a,root);

split(tree[root].right,tree[a].right,b,key);

update(a);

}

}

void split2(int root,int &a,int &b,int key)

{

if (root==-1)

{

a=-1;b=-1;

return;

}

if (tree[root].val<=key)

{

cop(a,root);

split2(tree[root].right,tree[a].right,b,key);

update(a);

}

else

{

cop(b,root);

split2(tree[root].left,a,tree[b].left,key);

update(b);

}

}

void split3(int root,int &a,int &b,int key)

{

if (root==-1)

{

a=-1;b=-1;

return;

}

if (LeftSize(root)>key)

{

cop(b,root);

split3(tree[root].left,a,tree[b].left,key);

update(b);

}

else

{

cop(a,root);

split3(tree[root].right,tree[a].right,b,key-LeftSize(root));

update(a);

}

}

void merge(int root1,int root2,int &res)

{

if (root1==-1) {res=root2;return;}

if (root2==-1) {res=root1;return;}

if (tree[root1].priority<=tree[root2].priority)

{

cop(res,root1);

merge(tree[root1].right,root2,tree[res].right);

}

else

{

cop(res,root2);

merge(root1,tree[root2].left,tree[res].left);

}

update(res);

}

void Treap\_insert(int rt,int val)

{

Pair splitted;split(root[rt],splitted.x,splitted.y,val);

int tmp=-1;

merge(splitted.x,Create(val,rand(),-1,-1),tmp);

merge(tmp,splitted.y,root[++rtot]);

}

void Treap\_delete(int rt,int val)

{

Pair splitted;split(root[rt],splitted.x,splitted.y,val);

Pair splitted2;split3(splitted.y,splitted2.x,splitted2.y,1);

if (splitted2.x==-1 || tree[splitted2.x].val==val)

merge(splitted.x,splitted2.y,root[++rtot]);

else

cop(root[++rtot],root[rt]);

}

void Treap\_xrank(int rt,int val)

{

Pair splitted;split(root[rt],splitted.x,splitted.y,val);

printf("%d\n",1+(splitted.x==-1?0:tree[splitted.x].sz));

cop(root[++rtot],root[rt]);

}

void Treap\_rankx(int rt,int key)

{

Pair splitted;split3(root[rt],splitted.x,splitted.y,key-1);

Pair splitted2;split3(splitted.y,splitted2.x,splitted2.y,1);

printf("%d\n",tree[splitted2.x].val);

cop(root[++rtot],root[rt]);

}

void Treap\_prev(int rt,int val)

{

Pair splitted;split(root[rt],splitted.x,splitted.y,val);

if (splitted.x==-1)

{

printf("-2147483647\n");

cop(root[++rtot],root[rt]);

return;

}

Pair splitted2;split3(splitted.x,splitted2.x,splitted2.y,tree[splitted.x].sz-1);

printf("%d\n",tree[splitted2.y].val);

cop(root[++rtot],root[rt]);

}

void Treap\_next(int rt,int val)

{

Pair splitted;split2(root[rt],splitted.x,splitted.y,val);

if (splitted.y==-1)

{

printf("2147483647\n");

cop(root[++rtot],root[rt]);

return;

}

Pair splitted2;split3(splitted.y,splitted2.x,splitted2.y,1);

printf("%d\n",tree[splitted2.x].val);

cop(root[++rtot],root[rt]);

}

int main ()

{

int q,type,rt,x;

q=getint();rtot=tot=0;

root[0]=-1;

while (q--)

{

rt=getint();type=getint();x=getint();

if (type==1) Treap\_insert(rt,x);

if (type==2) Treap\_delete(rt,x);

if (type==3) Treap\_xrank(rt,x);

if (type==4) Treap\_rankx(rt,x);

if (type==5) Treap\_prev(rt,x);

if (type==6) Treap\_next(rt,x);

}

return 0;

}

// 非旋Treap

struct Treap

{

int left,right;

int val,priority;

int sz;

}tree[1500048];

int tot=0,Root;

inline void update(int root)

{

tree[root].sz=1;

if (tree[root].left!=-1) tree[root].sz+=tree[tree[root].left].sz;

if (tree[root].right!=-1) tree[root].sz+=tree[tree[root].right].sz;

}

int Create(int val,int priority,int left,int right)

{

tree[++tot]=Treap{left,right,val,priority,1};

update(tot);

return tot;

}

int LeftSize(int root)

{

int res=1;

if (tree[root].left!=-1) res+=tree[tree[root].left].sz;

return res;

}

Pair split(int root,int key)

{

if (root==-1) return mp(-1,-1);

if (tree[root].val>=key)

{

Pair splitted=split(tree[root].left,key);

tree[root].left=splitted.y;

update(root);

return mp(splitted.x,root);

}

else

{

Pair splitted=split(tree[root].right,key);

tree[root].right=splitted.x;

update(root);

return mp(root,splitted.y);

}

}

Pair split2(int root,int key)

{

if (root==-1) return mp(-1,-1);

if (tree[root].val<=key)

{

Pair splitted=split2(tree[root].right,key);

tree[root].right=splitted.x;

update(root);

return mp(root,splitted.y);

}

else

{

Pair splitted=split2(tree[root].left,key);

tree[root].left=splitted.y;

update(root);

return mp(splitted.x,root);

}

}

Pair split3(int root,int key)

{

if (root==-1) return mp(-1,-1);

if (LeftSize(root)>key)

{

Pair splitted=split3(tree[root].left,key);

tree[root].left=splitted.y;

update(root);

return mp(splitted.x,root);

}

else

{

Pair splitted=split3(tree[root].right,key-LeftSize(root));

tree[root].right=splitted.x;

update(root);

return mp(root,splitted.y);

}

}

int merge(int root1,int root2)

{

if (root1==-1) return root2;

if (root2==-1) return root1;

if (tree[root1].priority<=tree[root2].priority)

{

tree[root1].right=merge(tree[root1].right,root2);

update(root1);

return root1;

}

else

{

tree[root2].left=merge(root1,tree[root2].left);

update(root2);

return root2;

}

}

void Treap\_insert(int val)

{

Pair splitted=split(Root,val);

Root=merge(merge(splitted.x,Create(val,rand(),-1,-1)),splitted.y);

}

void Treap\_delete(int val)

{

Pair splitted=split(Root,val);

Pair splitted2=split(splitted.y,val+1);

if (splitted2.x==-1)

Root=merge(splitted.x,splitted2.y);

else

Root=merge(merge(merge(splitted.x,tree[splitted2.x].left),tree[splitted2.x].right),splitted2.y);

}

void Treap\_xrank(int val)

{

Pair splitted=split(Root,val);

printf("%d\n",1+(splitted.x==-1?0:tree[splitted.x].sz));

Root=merge(splitted.x,splitted.y);

}

void Treap\_rankx(int x)

{

Pair splitted=split3(Root,x-1);

Pair splitted2=split3(splitted.y,1);

printf("%d\n",tree[splitted2.x].val);

Root=merge(merge(splitted.x,splitted2.x),splitted2.y);

}

void Treap\_prev(int val)

{

Pair splitted=split(Root,val);

if (splitted.x==-1)

{

printf("-2147483647\n");

Root=merge(splitted.x,splitted.y);

return;

}

Pair splitted2=split3(splitted.x,tree[splitted.x].sz-1);

printf("%d\n",tree[splitted2.y].val);

Root=merge(merge(splitted2.x,splitted2.y),splitted.y);

}

void Treap\_next(int val)

{

Pair splitted=split2(Root,val);

if (splitted.y==-1)

{

printf("2147483647\n");

Root=merge(splitted.x,splitted.y);

return;

}

Pair splitted2=split3(splitted.y,1);

printf("%d\n",tree[splitted2.x].val);

Root=merge(merge(splitted.x,splitted2.x),splitted2.y);

}

int main ()

{

q=getint();Root=-1;tot=0;

while (q--)

{

type=getint();x=getint();

if (type==1) Treap\_insert(x);

if (type==2) Treap\_delete(x);

if (type==3) Treap\_xrank(x);

if (type==4) Treap\_rankx(x);

if (type==5) Treap\_prev(x);

if (type==6) Treap\_next(x);

}

}