Basic Codes:

```
//////////in the name of almighty Allah//////////////
//////////////coded by: pranonraian/////////////////
#include<bits/stdc++.h>
using namespace std;
#define FAST ios_base::sync_with_stdio(0); cin.tie(NULL)
#define PI acos(-1.0)
#define TEST(n) cout<<"dhukse"<<n<<'\n'
#define endl '\n'
#define FILEREAD freopen("input.txt","r",stdin)
#define FIXED(n) cout << fixed << setprecision(n);
#define CASE(n) cout<<"Case " << n << ": ";
long long Set(long long N,long long pos){ return N=N | (1<<pos);}
long long reset(long long N,long long pos){return N= N & ~(1<<pos);}
bool check(long long N,long long pos){return (bool)(N & (1<<pos));}
long long min(long long a, long long b){if(a<b)return a;else return b;}
long long max(long long a, long long b){if(a>b)return a;else return b;}


int main()
{
    FAST;
    long long t=1;
    cin >> t;
    long long T = t;
    while(t--)
    {
    }

    return 0;
}
```

Segment Tree with Lazy Propagation:

```
#include<bits/stdc++.h>
using namespace std;

int arr[1000007];
vector<pair<int,int>>tree(1000007*3);


//be is the beginning of the range in the node
//en is the ending of the range in the node



void init_tree(int node,int be,int en)
{
    if(be==en)
    {
        tree[node].first=arr[be];
        tree[node].second=0;
        return ;
    }

    int left = node*2;
    int right = (2*node)+1;

    int mid = be+en;
    mid/=2;
    init_tree(left,be,mid);
    init_tree(right,mid+1,en);
    tree[node].first=tree[left].first+tree[right].first;
    tree[node].second=tree[left].second+tree[right].second;
}
```

```
int query(int node, int be, int en, int i,int j,int carry)   //carry used to pass
the propagated value
{
    if(i>en or j<be)  return 0; //out of range
    if(be>=i and en<=j) // range ta node range er che boro
        return tree[node].first + carry*(en-be+1);
    int left = 2*node;
    int right = (2*node) +1;

    int mid = be+en;
    mid/=2;

    int left_subtree_result =
query(left,be,mid,i,j,carry+tree[node].second);
    int right_subtree_result =
query(right,mid+1,en,i,j,carry+tree[node].second);
    return left_subtree_result+right_subtree_result;
}
```

```
void update(int node,int be, int en, int i, int j, int val)
{
    if(i>en or j<be)    return ; //out of range of the node
    if(be>=i and en<=j)
    {
        tree[node].first+=((en-be+1)*val);        //makes the sum according to the range
        tree[node].second+=val;                   //generates the propagation value
        return;
    }
    int left = 2*node;
    int right = (2*node)+1;
    int mid = be+en;
    mid/=2;
    update(left,be,mid,i,j,val);
    update(right,mid+1,en,i,j,val);
    tree[node].first=tree[left].first+tree[right].first +
((en-be+1)*tree[node].second);
}
```