# LCA & BIT

```
#include <bits/stdc++.h>

#define mx 200007

using namespace std;

typedef long long int LL;

int n, sparse_table[mx][18], parent[mx], level[mx], dist[mx], ara[mx], st[mx], en[mx], Timer = 0;

vector<int> graph[mx];

void dfs(int u,int par,int dis)

{

   st[u] = ++Timer;

   parent[u] = par;

   dist[u] = level[u] = dis;

   for(int v: graph[u])

      if(par!=v)

         dfs(v,u,dis+1);

   en[u] = Timer;

}

void ini()

{

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

   {

      for(int j = 0; j<18; j++)

      {
```

```
            sparse_table[i][j] = -1;
        }
    }
}
void make_table()
{
    for(int i = 1; i<=n; i++)
    {
        sparse_table[i][0] = parent[i];
    }
    for(int j = 1; j<18; j++)
    {
        for(int i = 1; i<=n; i++)
        {
            if(sparse_table[i][j - 1] != -1)
            sparse_table[i][j] = sparse_table[sparse_table[i][j-1]][j-1];
        }
    }
}


int query(int p,int q)
{
    if(level[p]<level[q]) swap(p,q);
    for(int j = 17; j>=0; j--)
```

```
    {
        if(level[p] - (1<<j) >=level[q])
        {
            p = sparse_table[p][j];
        }
    }
    if(p == q) return p;
    for(int j = 17; j>=0; j--)
    {
        if(sparse_table[p][j] != -1 && sparse_table[p][j] != sparse_table[q][j])
        {
            p = sparse_table[p][j];
            q = sparse_table[q][j];
        }
    }
    return parent[p];
}
int calculate_dist(int a, int b)
{
    int lca = query(a,b);
    return dist[a] + dist[b] - 2*dist[lca];
}
LL getSum(LL BITree[], LL index)
{
```

```
    LL sum = 0;
    index = index + 1;
    while (index>0)
    {
        sum += BITree[index];
        index -= index & (-index);
    }
    return sum;
}
void updateBIT(LL BITree[], LL n, LL index, LL val)
{
    index = index + 1;
    while (index <= n)
    {
        BITree[index] += val;
        index += index & (-index);
    }
}
LL sum(LL x, LL BITTree1[], LL BITTree2[])
{
    return (getSum(BITTree1, x) * x) - getSum(BITTree2, x);
}
void updateRange(LL BITTree1[], LL BITTree2[], LL n,
                 LL val, LL l, LL r)
```

```
{
   updateBIT(BITTree1,n,l,val);
   updateBIT(BITTree1,n,r+1,-val);
   updateBIT(BITTree2,n,l,val*(l-1));
   updateBIT(BITTree2,n,r+1,-val*r);
}

LL rangeSum(LL l, LL r, LL BITTree1[], LL BITTree2[])
{
   return sum(r, BITTree1, BITTree2) -
         sum(l-1, BITTree1, BITTree2);
}



LL *constructBITree(LL n)
{
   LL *BITree = new LL[n+1];
   for (LL i=1; i<=n; i++)
      BITree[i] = 0;

   return BITree;
}
int find_curLca(int a, int b, int c)
{
```

```
    vector< pair<int,int> > vv;
    int lca = query(a,b);
    int dis = dist[a] + dist[b] - 2*dist[lca] + calculate_dist(lca,c);
    vv.push_back({dis,lca});
    lca = query(b,c);
    dis = dist[b] + dist[c] - 2*dist[lca] + calculate_dist(lca,a);
    vv.push_back({dis,lca});
    lca = query(c,a);
    dis = dist[c] + dist[a] - 2*dist[lca] + calculate_dist(lca,b);
    vv.push_back({dis,lca});
    sort(vv.begin(), vv.end());
    return vv[0].second;
}
bool isAncestor(int root, int u)
{
    return (st[root] <= st[u] && en[u] <= en[root]);
}
int up(int till, int cur)
{
    for(int i = 17; i >= 0; i--)
    {
        if(level[cur] - (1<<i) > level[till])
            cur = sparse_table[cur][i];
    }
```

```
    return cur;
}
int main(int argc, char const *argv[])
{
    ios_base::sync_with_stdio(false);
    int q;
    cin >> n >> q;
    for(int i = 1; i<=n; i++) cin >> ara[i];
    for(int i = 1; i<n; i++)
    {
        int u,v;
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
    ini();
    dfs(1,-1,1);
    make_table();
    LL *BITTree1, *BITTree2;
    BITTree1 = constructBITree(n);
    BITTree2 = constructBITree(n);
    for(int i = 1; i<=n; i++)
    {
        updateRange(BITTree1, BITTree2, n, ara[i], st[i] - 1, st[i] - 1);
```

```
}
int gRoot = 1, curRoot = 1;
while(q--)
{
   int t;
   cin >> t;
   if(t == 1)
   {
      cin >> curRoot;
   }
   else if(t == 2)
   {
      int u, v, x;
      cin >> u >> v >> x;
      int lca = find_curLca(curRoot, u, v);
      if(lca == curRoot)
      {
         updateRange(BITTree1,BITTree2, n, x, 0, Timer - 1);
      }
      else if(isAncestor(curRoot, lca))
      {
         updateRange(BITTree1, BITTree2, n, x, st[lca] - 1, en[lca] - 1);
      }
      else
```

```
        {
            if(!isAncestor(lca, curRoot))
            {
                updateRange(BITTree1, BITTree2, n, x, st[lca] - 1, en[lca] - 1);
            }
            else
            {
                int up_till_child_of_lca = up(lca, curRoot);
                updateRange(BITTree1, BITTree2, n, x, 0, Timer - 1);
                updateRange(BITTree1, BITTree2, n, -x, st[up_till_child_of_lca] - 1,
en[up_till_child_of_lca] - 1);
            }

        }
    }
    else
    {
        int v;
        cin >> v;
        if(v == curRoot)
        {
            cout << rangeSum(0, Timer - 1, BITTree1, BITTree2) << endl;
        }
        else if(isAncestor(curRoot, v) || !isAncestor(v, curRoot))
        {
```

```
            cout << rangeSum(st[v] - 1, en[v] - 1, BITTree1, BITTree2) << endl;
        }
        else
        {
            int up_till = up(v, curRoot);
            cout << rangeSum(0, Timer - 1, BITTree1, BITTree2) - rangeSum(st[up_till] - 1,
en[up_till] - 1, BITTree1, BITTree2) << endl;
        }
      }
   }
   return 0;
}
```

# Dynamic Diameter

```
#include <bits/stdc++.h>
using namespace std;
/*author :: humayan kabir,sust*/

int const MAX = 100000 + 7;
int const MAXM = 10000 + 7;
int const LOGN = 17;

int n, ara[MAX];
vpii e[MAX];
vi divisors[MAXM];
```

```
ll ans;


void init()
{
  for(int i = 1; i<MAXM; i++) {
    for(int j = i; j<MAXM; j+=i) {
      divisors[j].push_back(i);
    }
  }
}


namespace Lca {
  int lca[MAX][LOGN];
  int inTime[MAX];
  int outTime[MAX];
  ll h[MAX];
  int Timer;


  void dfs(int u, int p, int dist) {
    h[u] = dist;
    inTime[u] = ++Timer;
    for(int i = 0; i<LOGN; i++) {
      lca[u][i] = -1;
    }
```

```
  lca[u][0] = p;
  for(int i = 1; i<LOGN; i++) {
    int pp = lca[u][i - 1];
    if(pp == -1) continue;
    lca[u][i] = lca[pp][i - 1];
  }
  for(auto v : e[u]) {
    if(v.first == p) continue;
    dfs(v.first, u, dist + v.second);
  }
  outTime[u] = Timer;
}

void build() {
  Timer = 0;
  dfs(1, -1, 0);
}

int isancestor(int a, int b) {
  return (inTime[a] <= inTime[b] && outTime[a] >= outTime[b]);
}

int getlca(int a,int b)
{
```

```
  if(isancestor(a,b)) return a;
  if(isancestor(b,a)) return b;
  for(int i = LOGN - 1; i>=0; i--)
  {
    if(lca[a][i] == -1 or isancestor(lca[a][i], b)) continue;
    a = lca[a][i];
  }
  return lca[a][0];
 }

 ll getdist(int a,int b) {
  return h[a] + h[b] - 2 * h[getlca(a,b)];
 }
};

vpii edge[MAX];

void pushedge(int x, int y)
{
 if(ara[x] > ara[y]) swap(x,y);
 int g = __gcd(ara[x], ara[y]);
 for(int d : divisors[g]) {
  edge[d].push_back(make_pair(x, y));
 }
```

```
}

namespace Dsu {
 int par[MAX];
 ll diameter[MAX];
 int furthestA[MAX];
 int furthestB[MAX]; //here furthestA & B represent
 int sizeT[MAX];    //diameter two point of tree root at varible x

 void init(int u) {
  par[u] = u;
  diameter[u] = 0;
  furthestA[u] = u;
  furthestB[u] = u;
  sizeT[u] = 1;
 }

 int find(int u)
 {
  if(par[u] == u) return u;
  return par[u] = find(par[u]);
 }

 ll unionTree(int a,int b)
```

```
{
  a = find(a);
  b = find(b);
  if(sizeT[a] < sizeT[b]) swap(a,b);
  sizeT[a] += sizeT[b];
  int a1 = Dsu::furthestA[a];
  int a2 = Dsu::furthestB[a];
  int b1 = Dsu::furthestA[b];
  int b2 = Dsu::furthestB[b];
  if(diameter[a] < Lca::getdist(a1,b1)) {
    Dsu::diameter[a] = Lca::getdist(a1,b1);
    furthestA[a] = a1;
    furthestB[a] = b1;
  }
  if(diameter[a] < Lca::getdist(a1,b2)) {
    Dsu::diameter[a] = Lca::getdist(a1,b2);
    furthestA[a] = a1;
    furthestB[a] = b2;
  }
  if(diameter[a] < Lca::getdist(a2,b1)) {
    Dsu::diameter[a] = Lca::getdist(a2,b1);
    furthestA[a] = a2;
    furthestB[a] = b1;
  }
```

```
    if(diameter[a] < Lca::getdist(a2,b2)) {
      Dsu::diameter[a] = Lca::getdist(a2,b2);
      furthestA[a] = a2;
      furthestB[a] = b2;
    }
    if(diameter[a] < Lca::getdist(b1,b2)) {
      Dsu::diameter[a] = Lca::getdist(b1,b2);
      furthestA[a] = b1;
      furthestB[a] = b2;
    }
    par[b] = a;
    return diameter[a];
  }

}

bool cmp(pii a, pii b) {
  return ara[a.first] < ara[b.first];
}

void clear() {
  for(int i = 1; i<=n; i++) {
    e[i].clear();
  }
```

```
  for(int i = 1; i<MAXM; i++) {
    edge[i].clear();
  }
}
void solve() {
  Lca::build();
  for(int i = 1; i<=n; i++) {
    for(pii v : e[i]) {
      if(i < v.first) pushedge(i, v.first);
    }
  }
  ans = 0;
  for(int i = 1; i<MAXM; i++) {
    sort(edge[i].begin(), edge[i].end(), cmp);
    for(pii v : edge[i]) {
      Dsu::init(v.first);
      Dsu::init(v.second);
    }
    ll cur = 0;
    for(int j = edge[i].size() - 1; j>=0; j--) {
      ll d = Dsu::unionTree(edge[i][j].first, edge[i][j].second);
      cur = max(cur, d * ara[edge[i][j].first]);
    }
    ans = max(ans, cur * i);
```

```
  }
  printf("%lld\n", ans);
  clear();
}
int main(int argc, char const *argv[]) {
  init();

  int t;
  scanf("%d", &t);
  while(t--)
  {
    scanf("%d", &n);
    for(int i = 1; i<=n; i++) scanf("%d", &ara[i]);
    for(int i = 1; i<n; i++) {
      int u, v, w;
      scanf("%d %d %d", &u, &v, &w);
      e[u].push_back(make_pair(v, w));
      e[v].push_back(make_pair(u, w));
    }
    solve();
  }
  return 0;
}
```

# CENTROID DCMP

```
#include <bits/stdc++.h>

using namespace std;


#define error(args...) { string _s = #args; replace(_s.begin(), _s.end(), ',', ' '); stringstream _ss(_s);
istream_iterator<string> _it(_ss); err(_it, args); }


void err(istream_iterator<string> it) {}

template<typename T, typename... Args>

void err(istream_iterator<string> it, T a, Args... args) {

   cerr << *it << " = " << a << endl;

   err(++it, args...);

}


typedef long long ll;

typedef pair<int, int> pii;

typedef pair<ll, ll> pll;

typedef vector<int> vi;

typedef vector<pii> vpii;

typedef vector<pll> vpll;

typedef tuple<int,ll,int,int> data;

typedef tuple<ll,int,ll,int> data2;

typedef long double ld;
```

```
const int MAXM = 10000 + 7;
const int MAXN = 100000 + 7;

/*author :: humayan kabir,sust*/

vi divisors[MAXM];
void calculatediviors() {
  for(int i = 1; i < MAXM; i++) {
    for(int j = i; j < MAXM; j += i) {
      divisors[j].push_back(i);
    }
  }
}
struct CentroidTree
{
    vector < vector<pii> > graph;
    vector <int> subtree;
    vector <bool> IsDeleted;
    vector <data> vt;
    data2 trackTwoMax[MAXM];
    int vis[MAXM], ara[MAXN];
    const int inf = 1E9;
    int n;
    int Root = 0, Timer = 0;
```

```
ll ans = 0;
CentroidTree(int n): n(n), graph(n), subtree(n), IsDeleted(n, false)
{
   memset(vis, 0, sizeof vis);
   for(int i = 0; i < n; i++) {
     scanf("%d", &ara[i]);
   }
   for(int i = 1; i < n; i++)
   {
      int u, v, w;
      scanf("%d %d %d",&u, &v, &w);
      graph[u - 1].push_back(make_pair(v - 1, w));
      graph[v - 1].push_back(make_pair(u - 1, w));
   }
   Decompose(Root, -1);
}
///Centroid Decomposition Part Start
int SubTreeDFS(int u, int p)
{
   subtree[u] = 1;
   int ret = 1;
   for(auto v : graph[u]) {
      if(v.first != p && !IsDeleted[v.first]) {
         SubTreeDFS(v.first, u);
```

```
            subtree[u] += subtree[v.first];
            ret += subtree[v.first];
        }
    }
    return ret;
}
int CentroidDFS (int u, int p, int &cnt) {
    for (auto v: graph[u]) {
        if (v.first != p && !IsDeleted[v.first] && subtree[v.first] > cnt/2)
            return CentroidDFS(v.first, u, cnt);
    }
    return u;
}
void gotochild(int u, int p,int mn, ll dist, int g,int childno) {
  mn = min(mn, ara[u]);
  g = __gcd(g, ara[u]);
  vt.emplace_back(mn, dist, g, childno);
  for(auto v : graph[u]) {
    if(v.first != p && !IsDeleted[v.first]) {
      gotochild(v.first, u, mn, dist + v.second, g, childno);
    }
  }
}
void trackBestTwo(int g, ll dist, int childno) {
```

```
  ll v1, v2;
  int t1, t2;
  tie(v1, t1, v2, t2) = trackTwoMax[g];
  if(t1 == childno) {
    if(dist > v1) {
      v1 = dist;
    }
    trackTwoMax[g] = tie(v1,t1,v2,t2);
    return;
  }
  if(dist > v1) {
    v2 = v1; t2 = t1;
    v1 = dist; t1 = childno;
    trackTwoMax[g] = tie(v1,t1,v2,t2);
    return;
  }
  if(dist > v2) {
    trackTwoMax[g] = tie(v1,t1,dist,childno);
    return;
  }
}
ll calculate(int mn, int g, ll dist1, ll dist2)
{
   return 1LL * mn * g * (dist1 + dist2);
```

```
}
void solve() {
  Timer++;
  sort(vt.rbegin(), vt.rend());
  for(data x : vt) {
    int mn, g, childno;
    ll dist;
    tie(mn, dist, g, childno) = x;
    for(int d : divisors[g]) {
      if(vis[d] == Timer) {
        int t1, t2;
        ll v1, v2;
        tie(v1,t1,v2,t2) = trackTwoMax[d];
        if(t1 != childno) {
          ans = max(ans, calculate(mn, d, dist, v1));
        }
        if(t2 != childno) {
          ans = max(ans, calculate(mn, d, dist, v2));
        }
      }
      else {
        vis[d] = Timer;
        trackTwoMax[d] = data2(-1, -1, -1, -1);
      }
```

```
      trackBestTwo(d, dist, childno);
    }
  }
}
void Decompose(int root, int p)
{
    vt.clear();
    int childno = 0;
    int cnt = SubTreeDFS(root, root);
    int centroid = CentroidDFS(root, root, cnt);
    IsDeleted[centroid] = true;
    vt.emplace_back(ara[centroid], 0, ara[centroid], childno);
    for(auto v : graph[centroid]) {
      if(v.first != p && !IsDeleted[v.first]) {
        gotochild(v.first, centroid, ara[centroid], v.second, ara[centroid], ++childno);
      }
    }
    solve();
    for(auto v : graph[centroid]) {
        if(v.first != p && !IsDeleted[v.first]) {
            Decompose(v.first, centroid);
        }
    }
    return;
```

```
    }
    ///Centroid Decomposition Part Ends
    /// Particular Problem Part
};
int main() {
    //ios::sync_with_stdio(false);
    calculatediviors();
    int t, n;
    scanf("%d",&t);
    while(t--) {
      scanf("%d", &n);
      CentroidTree T(n);
      printf("%lld\n", T.ans);
    }
    return 0;
}
```

# DSU ON TREE

```
#include <bits/stdc++.h>
#define mx 1000007
using namespace std;
vector<int> graph[mx];
int tree[5 * mx], sz[mx], st[mx], ft[mx], height[mx], ans[mx], cnt[mx], Timer = 0;
```

```
void getsz(int u, int par, int level)
{
   st[u] = ++Timer;
   height[Timer] = level;
   sz[u] = 1;
   for(int v : graph[u])
      if(v != par)
         getsz(v, u, level + 1), sz[u] += sz[v];

  ft[u] = Timer;
}
void update(int node,int l,int r, int x, int val)
{
   if(r < x || l > x || l > r) return;
   if(l >= x && r <= x)
   {
      tree[node] += val;
      return;
   }
   int mid = (l + r) / 2;
   update(node + node, l, mid, x, val);
   update(node + node + 1, mid + 1, r, x, val);
   tree[node] = max(tree[node + node], tree[node + node + 1]);
}
```

```
int query(int node,int l,int r, int val)
{
   int mid = (l + r) / 2;
   if(l == r) return l;
   if(tree[node + node] >= val) return query(node + node, l, mid, val);
   else return query(node + node + 1, mid + 1, r, val);
}
void dfs(int u, int par, int level,int keep)
{
   int MX = -1, bigChild = -1;
   for(int v : graph[u])
      if(v != par && sz[v] > MX)
         MX = sz[v], bigChild = v;

   for(int v : graph[u])
      if(v != par && v != bigChild)
         dfs(v, u, level + 1, 0);

   if(bigChild != -1) dfs(bigChild, u, level + 1, 1);

   for(int v : graph[u])
      if(v != par && v != bigChild)
         for(int i = st[v]; i<=ft[v]; i++)
            update(1,1,mx-7,height[i], 1);
```

```
    ///cout << "level check "<<level << endl;
    update(1,1,mx - 7, level, 1);
    ans[u] = query(1,1,mx-7,tree[1]) - level;
    if(!keep)
    {
        for(int i = st[u]; i<=ft[u]; i++)
            update(1,1,mx - 7, height[i], -1);
    }

}
int main(int argc, char const *argv[])
{
    int n;
    scanf("%d",&n);
    for(int i = 1; i<n; i++)
    {
        int u, v;
        scanf("%d %d",&u,&v);
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
    getsz(1, -1, 1);
    dfs(1, -1, 1, 0);
```

```
  /// cout << "fa " << endl;
   for(int i = 1; i<=n; i++) printf("%d\n",ans[i]);
   return 0;
}
```

# PERSISTENT TRIE

```
#include <bits/stdc++.h>
#define mx 300007
using namespace std;
int root[mx], nodeNo = 0;
struct tree
{
        int ara[2];
} trie[mx * 32];
inline void insert(int prevNode, int &curNode, int k)
{
        curNode = ++nodeNo;
        int curRoot = curNode;
        for(int i = 30; i>=0; i--)
        {
                bool bit = k & (1 << i);
                trie[curRoot].ara[bit] = ++nodeNo;
                trie[curRoot].ara[!bit] = trie[prevNode].ara[!bit];
                curRoot = nodeNo;
```

```
                prevNode = trie[prevNode].ara[bit];
        }
}
inline int querymx(int curRoot, int k)
{
        int ans = 0;
        for(int i = 30; i>=0; i--)
        {
                bool bit = k & (1 << i);
                if(trie[curRoot].ara[!bit] > 0)
                {
                        ans = ans | (1 << i);
                        curRoot = trie[curRoot].ara[!bit];
                }
                else if(trie[curRoot].ara[bit] > 0)
                {
                        curRoot = trie[curRoot].ara[bit];
                }
                else
                {
                        break;
                }
        }
        return ans;
```

```
}
inline int querymn(int curRoot, int k)
{
	int ans = 0;
	for(int i = 30; i>=0; i--)
	{
		bool bit = k & (1 << i);
		if(trie[curRoot].ara[bit] > 0)
		{
			curRoot = trie[curRoot].ara[bit];
		}
		else if(trie[curRoot].ara[!bit] > 0)
		{
			ans = ans | (1 << i);
			curRoot = trie[curRoot].ara[!bit];
		}
		else
		{
			break;
		}
	}
	return ans;
}
int main(int argc, char const *argv[])
```

```
{
	ios_base::sync_with_stdio(false);
	int n, q, id = 0;
	cin >> n >> q;
	map<int,int> Maping;
	int r, key;
	cin >> r >> key;
	Maping[r] = ++id;
	insert(root[0], root[Maping[r]], key);
	for(int i = 1; i<n; i++)
	{
		int u, v, k;
		cin >> u >> v >> k;
		if(Maping.find(u) == Maping.end()) Maping[u] = ++id;
		if(Maping.find(v) == Maping.end()) Maping[v] = ++id;
		insert(root[Maping[v]], root[Maping[u]], k);
	}
	int last_answer = 0;
	for(int i = 0; i<q; i++)
	{
		int t;
		cin >> t;
		t =t^last_answer;
		if(!t)
```

```
        {
            int u, v, k;
            cin >> v >> u >> k;
            u = u ^ last_answer;
            v = v ^ last_answer;
            k = k ^ last_answer;
            //cout<<u<<" ... "<<v<<" "<<k<<endl;
          if(Maping.find(u) == Maping.end()) Maping[u] = ++id;
          if(Maping.find(v) == Maping.end()) Maping[v] = ++id;
          insert(root[Maping[v]], root[Maping[u]], k);
        }
        else
        {
            int v, k;
            cin >> v >> k;
            v = v ^ last_answer;
            k = k ^ last_answer;
            //cout<<" .... "<<v<< ' '<<k<<endl;
            int MN = querymn(root[Maping[v]], k);
            int MX = querymx(root[Maping[v]], k);
            last_answer = MN ^ MX;
            cout << MN << " " << MX << endl;
        }
}
```

```
	return 0;
}
```

# Dynamic ConvexHullTrick

```
#include <bits/stdc++.h>

using namespace std;


#define error(args...) { string _s = #args; replace(_s.begin(), _s.end(), ',', ' '); stringstream _ss(_s); istream_iterator<string> _it(_ss); err(_it, args); }


void err(istream_iterator<string> it) {}

template<typename T, typename... Args>

void err(istream_iterator<string> it, T a, Args... args) {

   cerr << *it << " = " << a << endl;

   err(++it, args...);

}


typedef long long ll;

typedef pair<int, int> pii;

typedef pair<ll, ll> pll;

typedef vector<int> vi;

typedef vector<pii> vpii;

typedef vector<pll> vpll;

typedef long double float128;
```

```
/*author :: humayan kabir,sust*/

const ll is_query = -(1LL<<62), inf = 1e18;

struct Line {
   ll m, b;
   mutable function<const Line*()> succ;
   bool operator<(const Line& rhs) const {
      if (rhs.b != is_query) return m < rhs.m;
      const Line* s = succ();
      if (!s) return 0;
      ll x = rhs.m;
      return b - s->b < (s->m - m) * x;
   }
};

struct HullDynamic : public multiset<Line> { // // will maintain lower hull for maximum
   bool bad(iterator y) {
      auto z = next(y);
      if (y == begin()) {
         if (z == end()) return 0;
         return y->m == z->m && y->b <= z->b;
      }
      auto x = prev(y);
```

```
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (float128)(x->b - y->b)*(z->m - y->m) >= (float128)(y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });//for maxi
        //   auto y = insert({ -m, -b });// for here for minimum
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    //for query, Line can't be empty
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return (l.m * x + l.b);//here for maxi
      //   return -(l.m * x + l.b);////  here for minimum
    }
};
ll sum[200007], p[200007];
void solve(int n) {
  HullDynamic H;
  H.insert_line(0, 0);
  ll ans = 0;
  for(int i = 1; i <= n; i++) {
```

```
    ll num;
    scanf("%lld", &num);
    sum[i] = i * num + sum[i - 1];
    p[i] = num + p[i - 1];
    ans = max(ans, sum[i] + H.eval(p[i]));
    H.insert_line(-i*1LL, 1LL * i * p[i] - sum[i]);
  }
  printf("%lld\n", ans);
}
int main(int argc, char const *argv[]) {
  int n;
  scanf("%d", &n);
  solve(n);
  return 0;
}
```

# DIVIDE AND CONQUER

```
#include <bits/stdc++.h>
#define infinity 1111111111111111111ll
using namespace std;
typedef long long int LL;
LL n, g, ara[8002], dp[8002][802], sum[8002];
LL cost(int i,int j)
```

```
{
	return (sum[j] - sum[i - 1]) * (j - i + 1);
}
void rec(int g,int l,int r,int kl,int kr)
{
	if(l > r) return;
	int mid = (l + r) / 2;
	dp[mid][g] = infinity;
	int bestk = -1;
	for(int i = kl; i<=kr; i++)
	{
		if(i > mid) break;
		LL now = dp[i][g - 1] + cost(i + 1,mid);
		if(now < dp[mid][g])
		{
			dp[mid][g] = now;
			bestk = i;
		}
	}
	rec(g, l, mid - 1, kl, bestk);
	rec(g, mid + 1, r, bestk, kr);
}

int main(int argc, char const *argv[])
```

```
{
	ios_base::sync_with_stdio(false);
	cin >> n >> g;
	for(int i = 1; i<=n; i++) cin >> ara[i];
	for(int i = 1; i<=n; i++) sum[i] = sum[i - 1] + ara[i];
	for(int i = 1; i<=n; i++) dp[i][1] = cost(1,i);
	for(int i = 2; i<=g; i++)
	{
		rec(i,1,n,1,n);
	}
	cout << dp[n][g] << endl;
	return 0;
}
```