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

**using namespace std;**

**#define M 1000000007**

**struct compare { // Inheritance templating scoping**

**bool operator() (int&a, int&b) {**

**return a>b;**

**}**

**};**

**//Min Heap**

**// Comparator for soldier question**

**bool c2 (std::pair<int, int>&a, std::pair<int, int>&b) {**

**if(a.first != b.first) {**

**return a.first < b.first;**

**} else {**

**return a.second > b.second;**

**}**

**// true: b has a more priority**

**}**

**int main() {**

**/\* \*//\* std::vector<int>v;**

**cout<<"Yes"<<endl;**

**v.push\_back(5);**

**v.push\_back(10);**

**v.push\_back(6);**

**v.push\_back(3);**

**std::make\_heap(v.begin(), v.end(), compare());**

**cout<<"Hi"<<endl;**

**cout<<v.front()<<endl;**

**std::pop\_heap(v.begin(), v.end(), compare());**

**v.push\_back(9);**

**std::push\_heap(v.begin(), v.end(), compare());\*//\***

**int n, k;**

**cin>>n>>k;**

**std::vector<std::pair<int, int>>vec(n);**

**for(int i=0;i<n;i++){**

**cin>>vec[i].first;**

**vec[i].second = i;**

**}**

**std::make\_heap(vec.begin(), vec.end(), c2);**

**for(int i=0;i<k;i++) {**

**auto ans = vec.front();**

**cout<<ans.second + 1<<" ";**

**// Removal operation**

**std::pop\_heap(vec.begin(), vec.end(), c2);**

**vec.resize(vec.size() - 1); // capacity**

**if(ans.first != 1) {**

**// Addition Operation**

**vec.push\_back({ans.first - 1, ans.second});**

**std::push\_heap(vec.begin(), vec.end(), c2);**

**}**

**}**

**cout<<endl;\*/**

**int numberBuildings;**

**cin>>numberBuildings;**

**// Increasing order;**

**int bricks, ropes;**

**cin>>bricks>>ropes;**

**vector<int>arr(numberBuildings);**

**for(int i=0;i<numberBuildings;i++){**

**cin>>arr[i];**

**}**

**vector<int>diffArr;**

**for(int i=1;i<numberBuildings;i++) {**

**diffArr.push\_back(arr[i] - arr[i-1]);**

**}**

**// Defined answer space here**

**int low = 0, high = numberBuildings-1;**

**int ans = 0;**

**while(low<high) {**

**int mid = (low + high) >> 1;**

**priority\_queue<int>pq;**

**for(int i = 0; i < mid;i++) {**

**pq.push(diffArr[i]);**

**}**

**int ropeTemp = ropes;**

**while(!pq.empty() and ropeTemp > 0) {**

**pq.pop();**

**ropeTemp--;**

**}**

**int mxBrickCountPossible = 0;**

**while(!pq.empty()) {**

**mxBrickCountPossible += pq.top();**

**pq.pop();**

**}**

**if(mxBrickCountPossible > bricks) {**

**high = mid - 1;**

**} else {**

**ans = mid;**

**low = mid + 1;**

**}**

**}**

**cout<<ans<<endl;**

**return 0;**

**}**