**рекурсия** – это функция, которая сама вызывает себя.

У [хорошей](http://lurkmore.to/%D0%A5%D0%BE%D1%80%D0%BE%D1%88%D0%B8%D0%B9,_%D0%B3%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9) рекурсии должно быть и условие для выхода из рекурсии, иначе получается зацикливание

Типы:

Рекурсия, при которой рекурсивные вызовы на любом рекурсивном срезе, инициируют не более одного последующего рекурсивного вызова, называется *линейной.* Это наиболее простой и часто встречающийся тип рекурсии.

Частный случай линейной рекурсии с отсутствующими предварительными или отложенными вычислениями называется *повторительной* рекурсией.

тип рекурсии дополнительный к линейному - называют *каскадной*, если она не является линейной. При каскадной рекурсии, рекурсивные обращения, как правило, приводят к необходимости многократно решать одни и те же подзадачи. И поэтому, по возможности, от неё или следует избавляться или предпринимать шаги, освобождающие от необходимости производить повторные решения подзадач.

Циклическая последовательность вызовов нескольких функций F1, F2, …, Fk (процедур, алгоритмов) друг друга: F1 вызывает F2, F2 вызывает F3, …, Fk вызывает F1 (k>1), называют *косвенной* или *взаимной* рекурсией.

Рекурсия называется *удаленной*, если в теле функции при рекурсивных вызовах, в выражениях, являющихся фактическими параметрами, снова встречаются рекурсивные вызовы этой функции. Подобный пример дает классическая в теории рекурсии функция Аккермана.

[*жадный алгоритм*](http://ru.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) (greedy algorithm) — это алгоритм, который на каждом шагу делает локально наилучший выбор в надежде, что итоговое решение будет оптимальным.

[алгоритм Дейкстры](http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B) нахождения кратчайшего пути в графе вполне себе жадный, потому что мы на каждом шагу ищем вершину с наименьшим весом, в которой мы еще не бывали, после чего обновляем значения других вершин. При этом можно доказать, что кратчайшие пути, найденные в вершинах, являются оптимальными.

*[матроид](http://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%BE%D0%B8%D0%B4)* — это пара (X, I), где X — конечное множество, называемое носителем матроида, а I — некоторое множество подмножеств X, называемое семейством независимых множеств.

условия:

1. Множество I непусто. Даже если исходное множество X было пусто — X = Ø, то I будет состоять из одного элемента — множества, содержащего пустое. I = {{Ø}}
2. Любое подмножество любого элемента множества I также будет элементом этого множества.
3. Если множества A и B принадлежат множеству I, а также известно, что размер А меньше B, то существует какой-нибудь элемент x из B, не принадлежащий А, такое что объединение x и A будет принадлежать множеству I. Это свойство является не совсем тривиальным, но чаще всего наиважшейшим из всех остальных.

Матроид называется взвешенным, если на множестве X существует аддитивная весовая функция w.

теорема [Радо-Эдмондса](http://ru.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A0%D0%B0%D0%B4%D0%BE_%E2%80%94_%D0%AD%D0%B4%D0%BC%D0%BE%D0%BD%D0%B4%D1%81%D0%B0): если доказать, что объект является матроидом, то жадный алгоритм будет работать корректно и выдавать правильный результат.

#include <iostream>

#include <set>

#include <climits>

using namespace std;

int inf = INT\_MAX;

int main() {

cin.sync\_with\_stdio(false);

int n;

cin >> n;

int na;

cin >> na;

multiset<pair<int, int> > s;

int a[na];

int resa[n];

for(int i=0; i<na; i++)

{

cin >> a[i];

s.insert({a[i], a[i]});

}

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

{

auto it=s.begin();

int val=it->first;

int t=it->second;

s.erase(it);

s.insert({val+t, t});

resa[i]=val;

}

int nb;

cin >> nb;

int b[nb];

for(int i=0; i<nb; i++)

{

cin >> b[i];

}

multiset<pair<int, int> > q;

long long l=resa[n-1], r=10000100;

while(l<r)

{

int w=0;

int m=(l+r)/2;

for(int i=0; i<nb; i++)

{

q.insert({b[i], b[i]});

}

q.insert({inf, inf});

for(int i=n-1; i>=0; i--)

{

int y=m-resa[i];

auto x=q.begin();

if(y<(x->first))

{

w++;

break;

}

else{

auto it=q.lower\_bound({y, -1});

int z=it->first;

if(z>y)

it--;

int val=it->first;

int t=it->second;

q.erase(it);

q.insert({val+t, t});

}

}

if(w==0)

r=m;

else

l=m+1;

q.clear();

}

cout << l;

return 0;

}