1 задание

1.3 Даны четыре неотрицательных числа a, b, c и d. Сложить две рациональные дроби a/b и c/d, а их результат представить в виде несократимой дроби m/n. Вывести числа m и n. a, b, c, d ≤ 1000.

#include <iostream>

//сумма дробей

using namespace std;

int gcd (int b, int d){

if (d==0)

return b;

else

return gcd (d, b%d);

}

int main()

{

int a, b, c, d;

cin >> a >> b >> c >> d;

int m, n;

n = b\*d/gcd (b, d);

m = a\*n/b+c\*n/d;

while (gcd (m, n)!=1) {

int gcd1 = gcd (m, n);

m = m/gcd1;

n = n/gcd1;

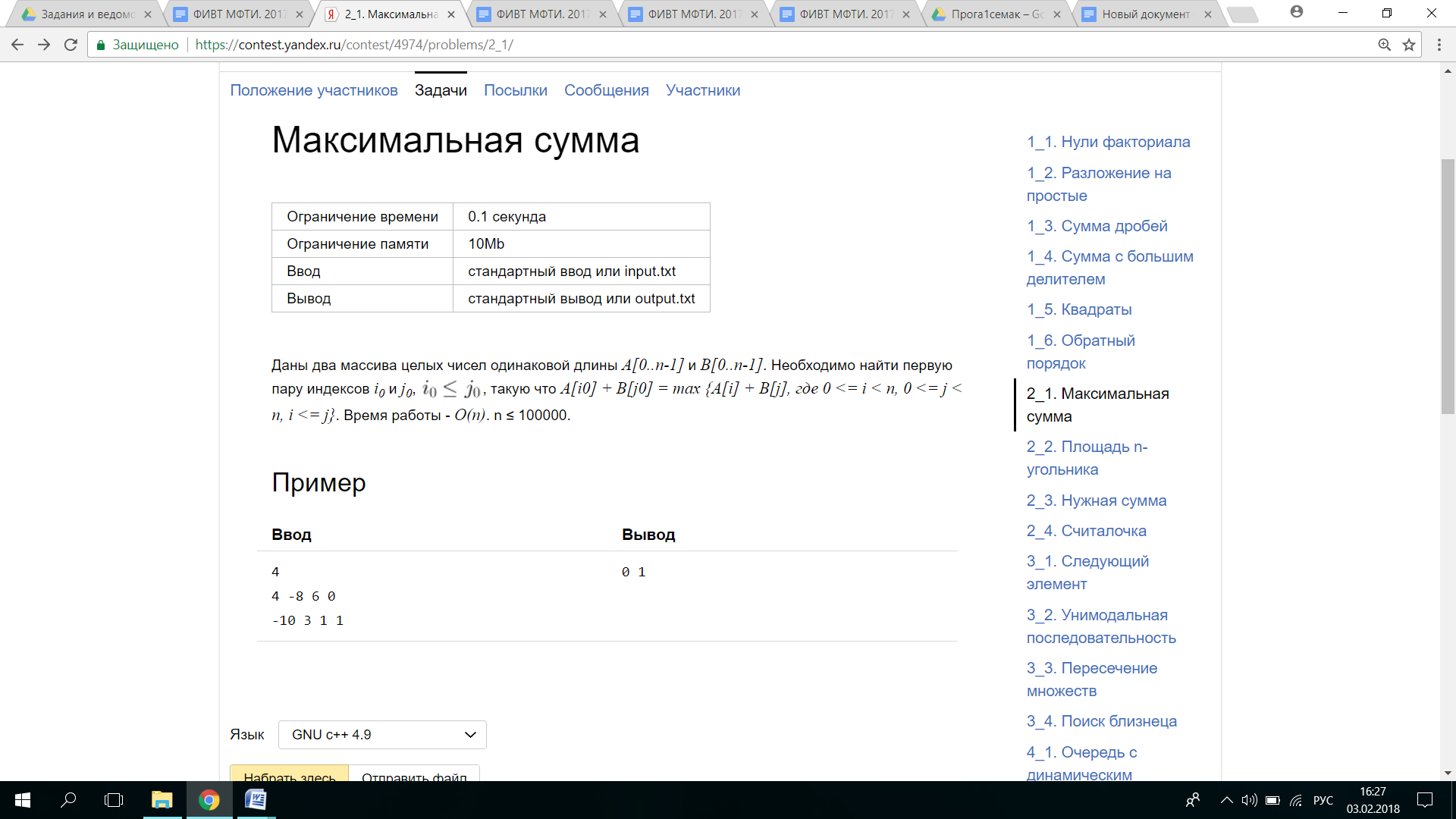
}

cout << m << " " << n;

return 0;

}

2.1



#include <iostream>

#include <climits>

using namespace std;

int main() {

int n, m;

cin >> n;

int \*A = new int[n];

int \*B = new int[n];

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

cin >> A[i];

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

cin >> B[i];

int maxSum = INT\_MIN, maxB = B[n - 1];

int summandAIndex, summandBIndex, maxBIndex;

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

if (B[i] >= maxB) {

maxB = B[i];

maxBIndex = i;

}

if (A[i] + maxB >= maxSum) {

maxSum = A[i] + maxB;

summandAIndex = i;

summandBIndex = maxBIndex;

}

}

cout << summandAIndex << " " << summandBIndex;

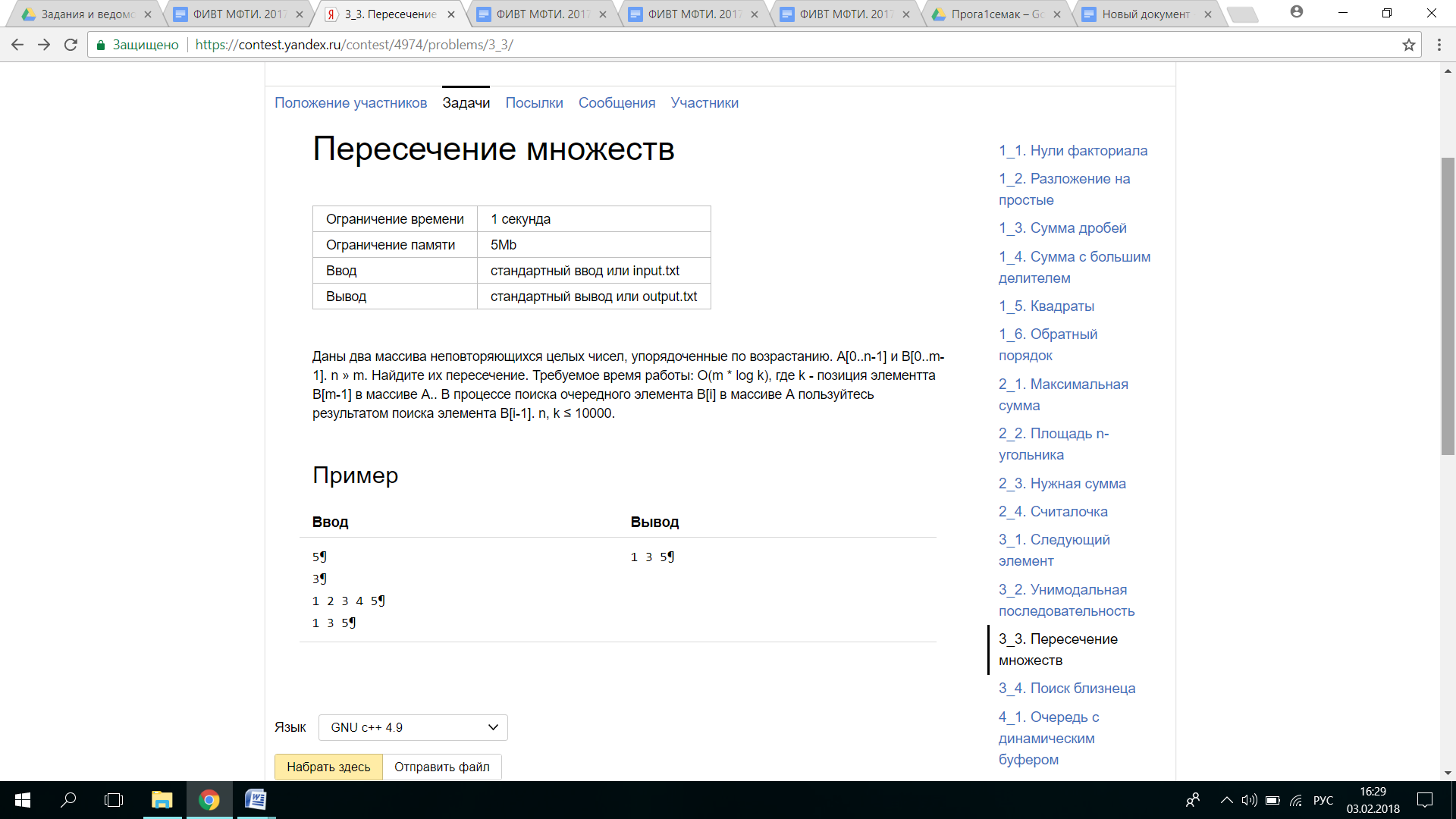
delete [] A;

delete [] B;

return 0;

}

3.4



#include <iostream>

using namespace std;

int elementSearch (const int \*A, int n, int element) {

if (n == 0 || (A[0] > element)) {

return -1;

}

if (A[0] == element) {

return 0;

}

int left = 0;

int right = n - 1;

while (left + 1 < right) {

const int middle = (left + right) / 2;

if (A[middle] < element) {

left = middle;

}

else

right = middle;

}

if (A[right] == element)

return right;

else

return -1;

}

int main() {

int n, m;

cin >> n;

int \*A = new int[n];

cin >> m;

int \*B = new int[m];

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

cin >> A[i];

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

cin >> B[i];

int border = 0;

for (int j = 0; j < m; j++) {

int t = elementSearch (A + border, n - border, B[j]);

if (t > -1) {

cout << B[j] << " ";

border = t + 1;

}

}

delete [] A;

delete [] B;

return 0;

}

4.2 ДЕК

#include <iostream>

using namespace std;

class Deque {

public:

Deque();

~Deque();

int Size();

int left();

int right();

void push\_Back(int element);

void push\_Front(int element);

void pop\_Back();

void pop\_Front();

private:

int head, tail, realSize, arraySize;

int \*Array;

static int const ArrayOneSize = 2;

void initBuffer() {

arraySize = Deque::ArrayOneSize;

Array = new int[arraySize];

}

void NewBuffer() {

int newBufferSize = arraySize \* 2;

int \*newBuffer = new int[newBufferSize];

int j = 0;

if (head == 0) {

for (int i = 0; i <= tail; i++) {

newBuffer[j] = Array[i];

j++;

}

}

else if (head > tail) {

for (int i = head; i < arraySize; i++)

newBuffer[j++] = Array[i];

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

newBuffer[j++] = Array[i];

}

else {

for (int i = head; i >= 0; i--)

newBuffer[j++] = Array[i];

for (int i = arraySize-1; i >= tail; i--)

newBuffer[j++] = Array[i];

}

tail = j - 1;

head = 0;

delete[] Array;

Array = newBuffer;

arraySize = newBufferSize;

}

};

Deque::Deque() {

head = tail = realSize = arraySize = 0;

Array = NULL;

}

Deque::~Deque() {

delete[] Array;

}

int Deque::left() {

return Array[head];

}

int Deque::right() {

return Array[tail];

}

int Deque::Size() {

return realSize;

}

void Deque::pop\_Back() {

if (tail == 0) {

tail = arraySize;

}

tail--;

realSize--;

}

void Deque::pop\_Front() {

realSize--;

head = (head + 1) % arraySize;

}

void Deque::push\_Back(int element) {

if (arraySize == 0) {

initBuffer();

}

if (realSize + 1 > arraySize) {

NewBuffer();

}

if (realSize == 0) {

tail = -1;

head = 0;

}

realSize++;

tail = (tail + 1) % arraySize;

Array[tail] = element;

}

void Deque::push\_Front(int element) {

if (arraySize == 0) {

initBuffer();

}

if (realSize + 1 > arraySize) {

NewBuffer();

}

if (head == 0) {

head = arraySize;

}

if (realSize == 0) {

tail = 0;

head = 1;

}

realSize++;

Array[--head] = element;

}

int main() {

Deque deque;

int n, error = 0;

cin >> n;

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

int a, b, test = 0;

cin >> a >> b;

if (a == 1) {

deque.push\_Front(b);

}

else if (a == 2) {

if (deque.Size() == 0) {

test = -1;

}

else {

test = deque.left();

deque.pop\_Front();

}

if (test != b) {

error = -1;

}

}

else if (a == 3) {

deque.push\_Back(b);

}

else {

if (deque.Size() == 0) {

test = -1;

}

else {

test = deque.right();

deque.pop\_Back();

}

if (test != b) {

error = -1;

}

}

}

if (error == 0) {

cout << "YES";

}

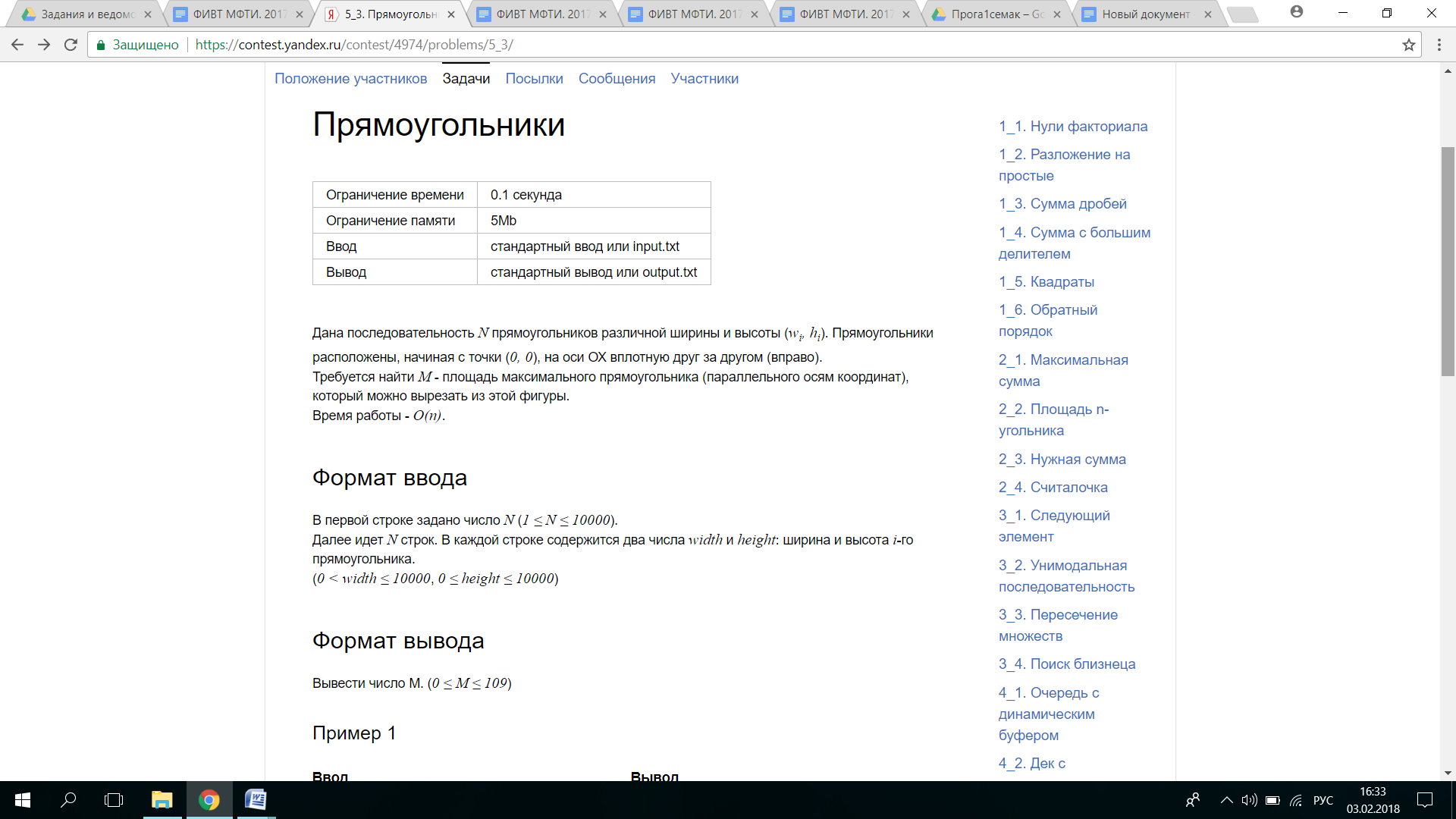
else {

cout << "NO";

}

return 0;

}

#include <iostream>

using namespace std;

struct rectangle

{

int width = 0;

int height = 0;

};

class Stack {

public:

Stack();

~Stack();

//bool empty();

void push(rectangle e);

void pop();

rectangle top();

int bufferSize;//размер буфера

rectangle\* buffer;//буфер типа прямоугольник, то есть у него будет высота и ширина

int front;//указывает на занятый

private:

void expand();

};

Stack::Stack() {

bufferSize = 32;

front = 0;// в нулевом лежит 0,-1

buffer = new rectangle[bufferSize];

buffer[0].height = -1;

}

Stack::~Stack() {

delete [] buffer;

}

void Stack::expand() {

int newBufferSize = bufferSize \* 2;

rectangle \*newBuffer = new rectangle[newBufferSize];

for(int i = 0; i < bufferSize; i++) {

newBuffer[i] = buffer[i];

}

delete [] buffer;

buffer = newBuffer;

bufferSize = newBufferSize;

}

rectangle Stack::top() {

return buffer[front];

}

void Stack::pop() {

front--;

}

void Stack::push(rectangle e){

if(front + 1 >= bufferSize) {

expand();

}

buffer[++front] = e;//!!!!

}

/\*

bool Stack::empty() {

return front == -1;

}\*/

int main()

{ int n;

cin >> n;

int square = 0;

Stack stack;

for (int i = 1; i < n + 2; i++ ){

int width0 = 0;

rectangle element;

if (i != n + 1){

cin >> element.width >> element.height;

}

////

if (element.height < stack.top().height) { //сравниваем с верхним в стеке элементом

while (element.height < stack.top().height){

width0 = width0 + stack.top().width; //общая длина

if (width0 \* stack.top().height > square){

square = width0 \* stack.top().height;

}

stack.pop();

}

element.width = element.width + width0;

stack.push(element);

}

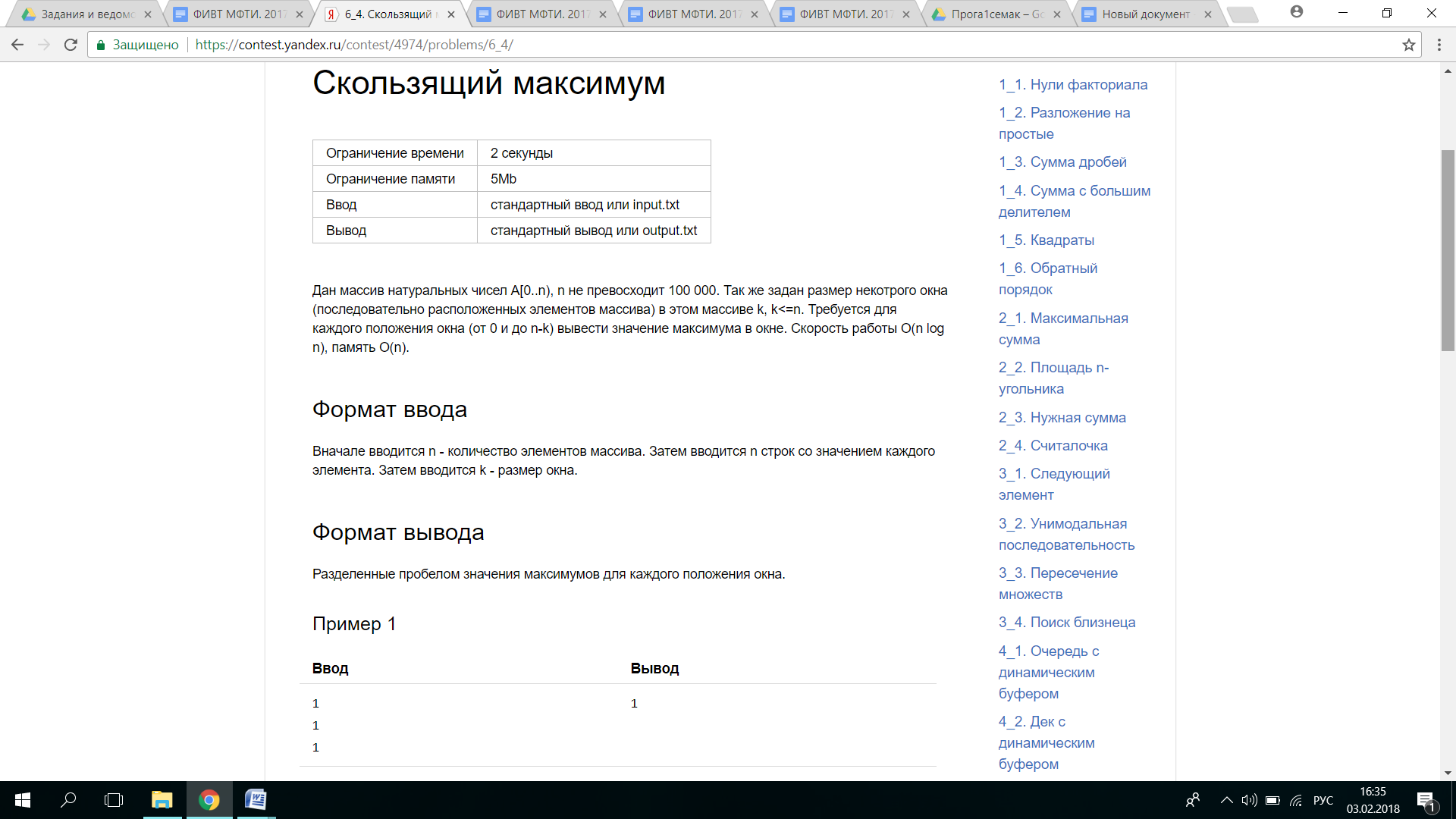
else stack.push(element);

}

cout << square;

return 0;

}

RE!!!

#include <iostream>

using namespace std;

struct couple

{

int value = 0;

int index = 0;

};

class Heap {

public:

Heap();

~Heap();

couple\* buffer;

int bufferSize;

int num;

int SizeHeapNow;

int window;

int\* Array;

void TakeArray(int n);

void SiftUp( int index );

void del();

void TakeBuffer (int i);

void BuildHeap(int k);

void SiftDown( int i );

void shift(int j);

void expand();

int maxheap ();

void AddArray(int i, int element);

// int cou(int g);

void initBuffer(int k) {

bufferSize=k;

buffer = new couple[bufferSize];

}

};

Heap::Heap() {

SizeHeapNow = 0;

}

Heap::~Heap() {

delete [] buffer;

delete [] Array;

}

void Heap::TakeArray(int n){

num = n;

Array = new int [num];

}

void Heap:: del(){

delete [] buffer;

delete [] Array;

}

void Heap::AddArray( int i, int element ){

Array[i] = element;

}

void Heap::TakeBuffer (int i){

buffer[i].value = Array[i];

buffer[i].index = i;

}

// Проталкивание элемента вниз. CArray – целочисленный массив.

void Heap::SiftDown( int i)

{

int left = 2 \* i + 1;

int right = 2 \* i + 2;

// Ищем большего сына, если такой есть.

int largest = i;

if( left < SizeHeapNow && buffer[left].value > buffer[i].value )

largest = left;

if( right < SizeHeapNow && buffer[right].value > buffer[largest].value )

largest = right;

// Если больший сын есть, то проталкиваем корень в него.

if( largest != i ) {

swap( buffer[i], buffer[largest] ); //меняю местами индексы, чтобы знать, какой элемент я должна заменить

SiftDown( largest);

}

}

// Построение кучи.

void Heap::BuildHeap( int k )

{ SizeHeapNow = k;

window = k;

for (int t = SizeHeapNow/2-1; t>-1; --t){

SiftDown( t);

}

}

int Heap::maxheap(){

return buffer[0].value;

}

// Проталкивание элемента наверх.

void Heap::SiftUp( int index)

{

while( index > 0 ) {

int parent = (index - 1)/2;

if( buffer[index].value <= buffer[parent].value )

return;

swap(buffer[index], buffer[parent] );

index = parent;

}

}

void Heap::expand() {

int newBufferSize = bufferSize +1;

couple \*newBuffer = new couple[newBufferSize];

for(int i = 0; i < bufferSize; i++) {

newBuffer[i] = buffer[i];

}

delete [] buffer;

buffer = newBuffer;

bufferSize = newBufferSize;

}

void Heap::shift(int j) {

if (buffer[0].index > j){

// expand();

buffer[SizeHeapNow].value = Array [j + window];

buffer[SizeHeapNow].index = j + window;

SiftUp ( SizeHeapNow );

SizeHeapNow++;

}

else {

buffer[0].value = Array [window+j];

buffer[0].index = window+j;

SiftDown(0);

}

}

int main()

{ ios\_base::sync\_with\_stdio(false);

cin.tie(NULL);

int n,k;

cin >> n;

Heap heap;

heap.TakeArray(n);

couple element;

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

cin>>element.value;

heap.AddArray( i, element.value);

}

cin>>k;

heap.initBuffer(n);

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

heap.TakeBuffer(i);

}

heap.BuildHeap(k);

cout<<heap.maxheap()<<" ";

for (int j = 0; j<n-k;j++){

heap.shift(j);

cout<<heap.maxheap()<<" ";

}

heap.del();

return 0;

}