МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Сортировки

Студент гр. 9382 _____ Рыжих Р.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

На практике изучить быструю сортировку, а также сортировку вставками, реализовав программу при помощи языка C++.

Задание.

Вариант 11

Быстрая сортировка, рекурсивная реализация – отсечение малых подмассивов.

Быстрая сортировка выполняется не до конца. Когда сегменты становятся достаточно маленькими, они окончательно сортируются другим методом. Параметр, определяющий размер малого подмассива, задаётся пользователем.

Основные теоретические положения.

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке

Быстрая сортировка - один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O(nlogn) обменов при упорядочении элементов.

Сортировка вставками - алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов

Описание алгоритма быстрой сортировки (рекурсивная реализация):

Быстрая сортировка функционирует по принципу "разделяй и властвуй". Она разбивает сортируемый массив на две части, затем сортирует эти части независимо друг от друга.

1) Выбрать из массива медиану из первого, среднего и последнего элементов в качестве центрального элемента.

2) Центральный элемент помещается на свое место, а остальные элементы переупорядочиваются таким образом, что меньшие элементы находятся слева от центрального элемента, а большие - справа.

3) Выполнить рекурсивную сортировку левой и правой частей массива.

4) Отсечь рекурсии для небольших подмассивов (размер малого подмассива вводится пользователем).

5) В завершении использовать другой метод – сортировка вставками.

Выполнение работы:

В данной задаче требуется считать массив с элементами, который далее требуется отсортировать.

Функция read()

Данная функция считывает массив элементов.

Функция print()

Данная функция печатает на экран введенный массив.

Функция exch()

Функция меняет местами два элемента.

Функция compexch()

Функция меняет местами элементы (вызывает функцию *exch()*), если второй элемент меньше первого.

Функция insertion()

Данная функция выполняет сортировку вставками. Функция помещает в первую позицию наименьший элемент массива, чтобы использовать его как

сигнальный ключ; во внутреннем цикле выполняется одна операция присваивания; в завершении прекращается выполнение внутреннего цикла, когда вставляемый элемент уже находится в требуемой позиции. Для каждого *i* упорядочиваются элементы *arr[l], ..., arr[i]* с помощью сдвига на одну позицию вправо элементов *arr[l], ..., arr[i-1]* из отсортированного списка, которые больше *arr[i]*, и занесения *arr[i]* в освободившееся место.

Функции partition()

Данная функция реализовывает разбиение. Переменная v содержит значение среднего элемента *arr[r]*, а *i* и *j* - соответственно левый и правый индексы. Цикл разбиения увеличивает *i* и уменьшает *j* на 1, соблюдая условие, что ни один элемент слева от *i* не больше *v* и ни один элемент справа от *j* не больше *v*. После встречи указателей процедура разбиения завершается перестановкой *arr[r]* и *arr[i]*, при этом в *arr[i]* заносится значение *v*, и справа от *v* не останется меньших его элементов, а слева - больших. Цикл разбиения реализован в виде бесконечного цикла с выходом после перекрещивания указателей при помощи *break*. Проверка *j==1* написана на случай, если центральный элемент окажется наименьшим.

Функция quickSort()

Данная функция рекурсивно выполняет быструю сортировку. Выбирается медиана из первого, среднего и последнего элементов в качестве центрального элемента. Далее происходит отсечение рекурсии для небольших подмассивов. Все подмассивы, которые меньше *M* (размер малого подмассива, введенный пользователем) игнорируются при разбиениях. Затем для завершения сортировки используется другой метод – сортировка вставками (*Insertion()*).

Функция hybridSort()

Данная функция объединяет в себе сразу две сортировки – быстрая сортировка и сортировка вставками.

Функция main()

В данной функции происходит считывание количества элементов массива, размер малого подмассива, и сам массив элементов. Происходит динамическое выделение памяти, а далее вызывается функция *HybridSort()*. Во время работы программы на экран выводятся промежуточные результаты. В завершении печатается отсортированный массив.

Пример работы программы.

На рисунках 1-5 представлены примеры работы программы с различными исходными данными.

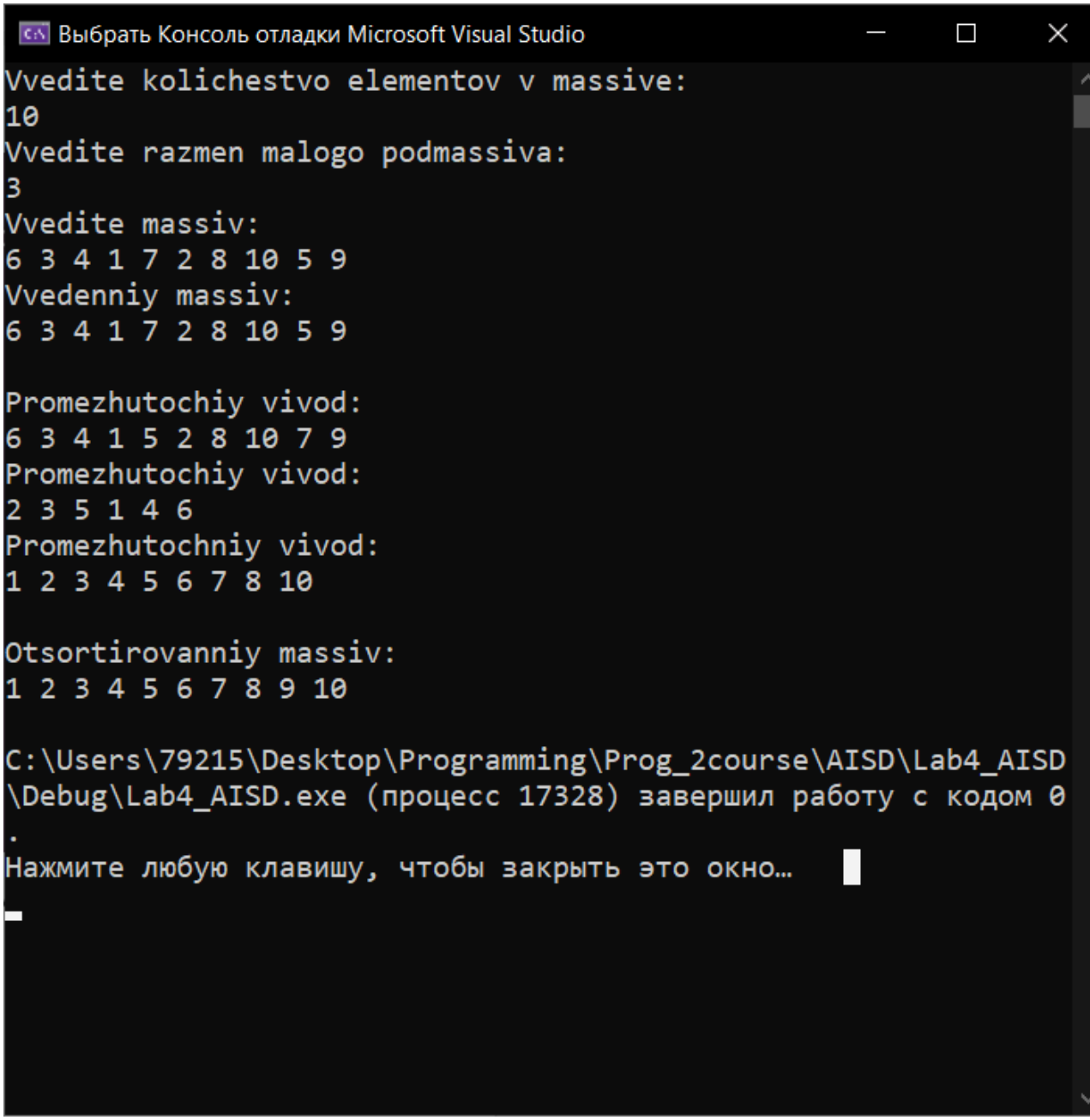

Рисунок 1 - Пример работы с входными данными № 1

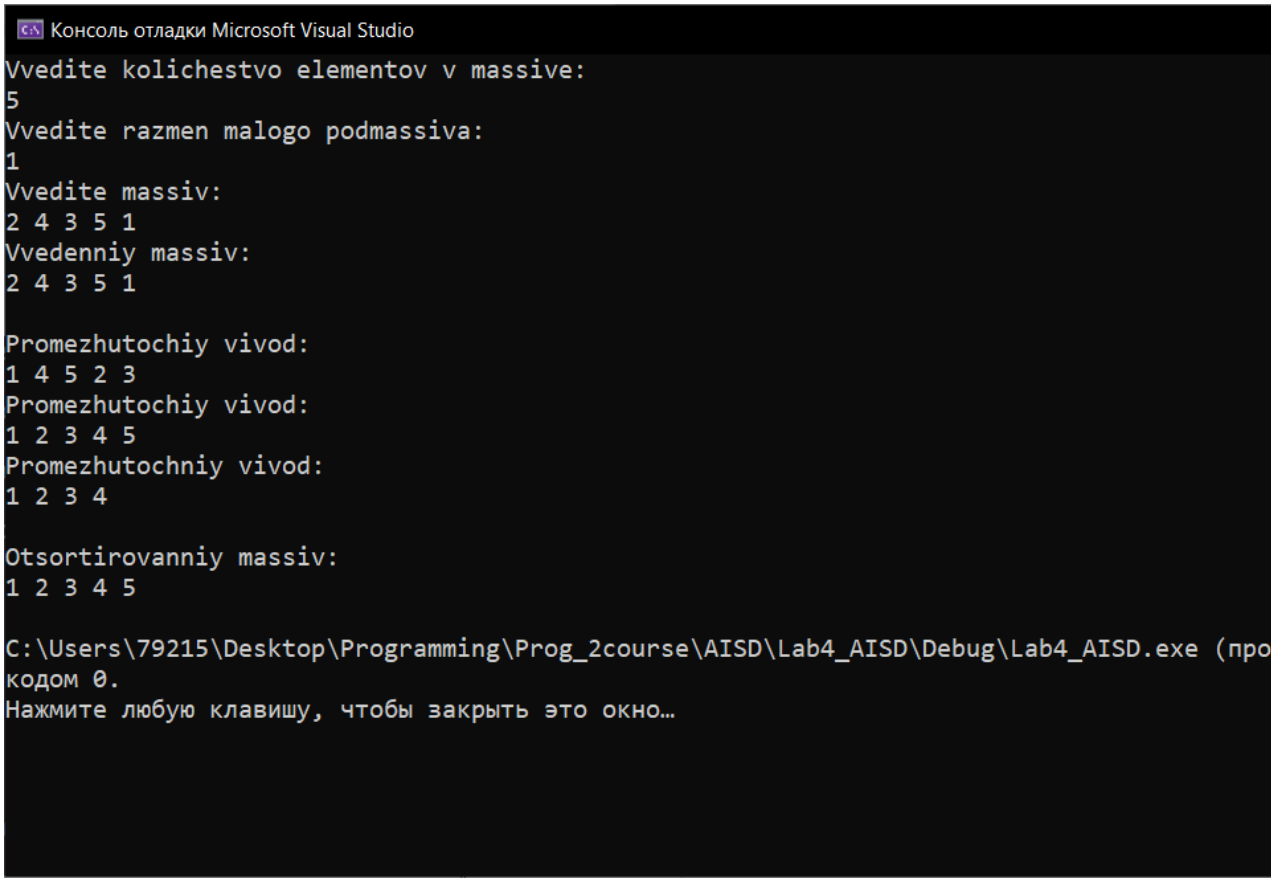

Рисунок 2 - Пример работы с входными данными № 2

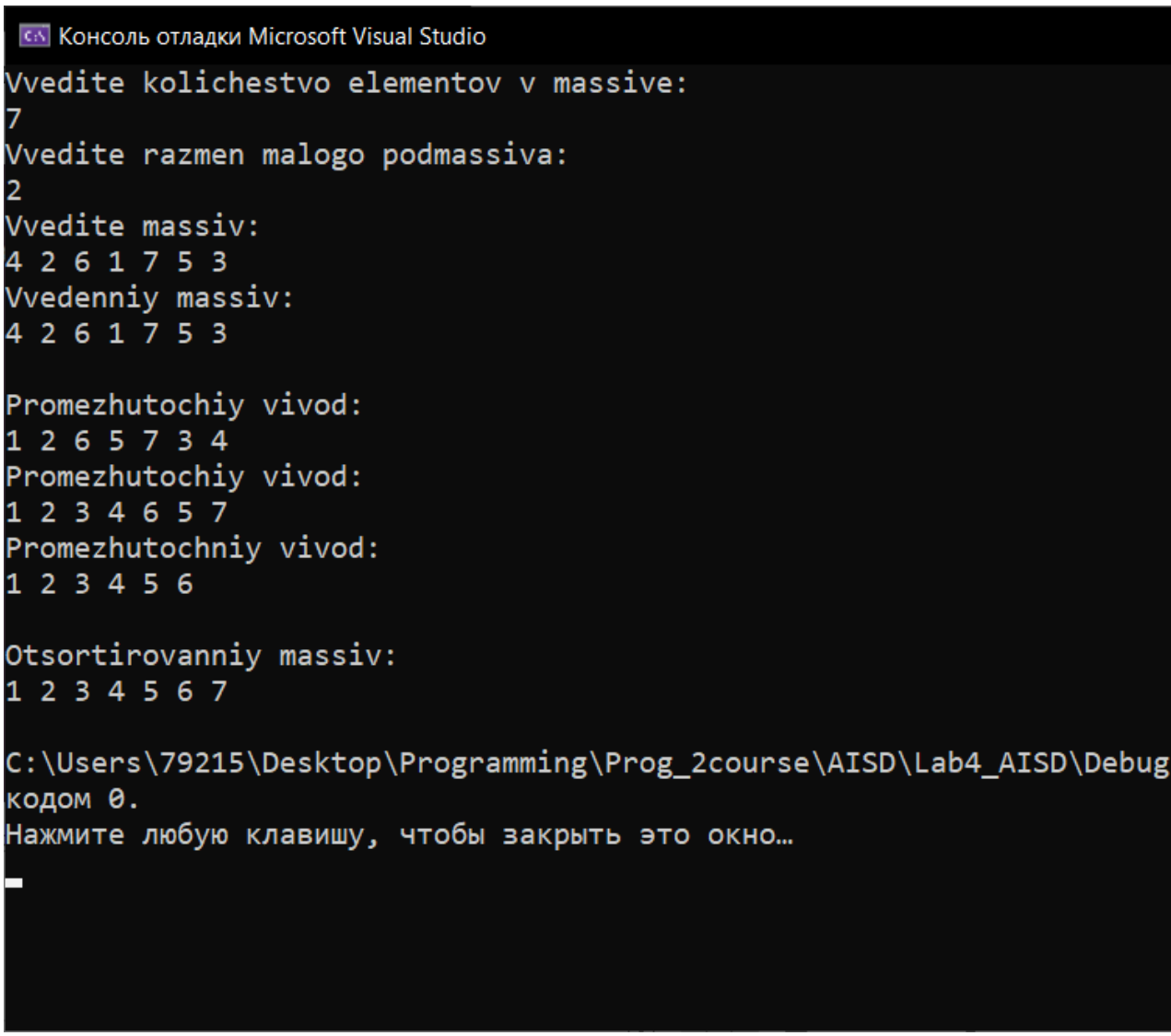

Рисунок 3 - Пример работы с входными данными № 3

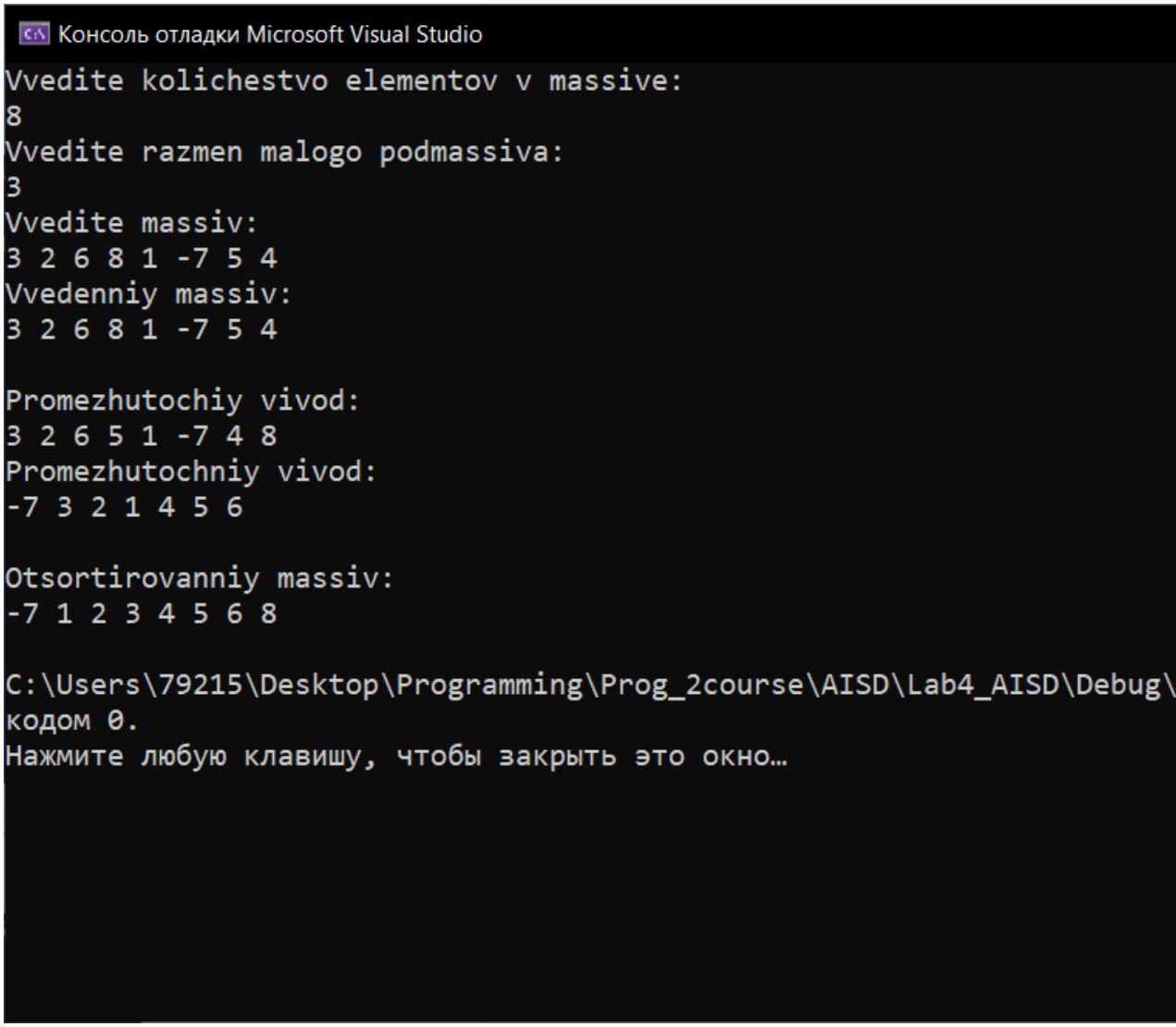

Рисунок 4 - Пример работы с входными данными № 4

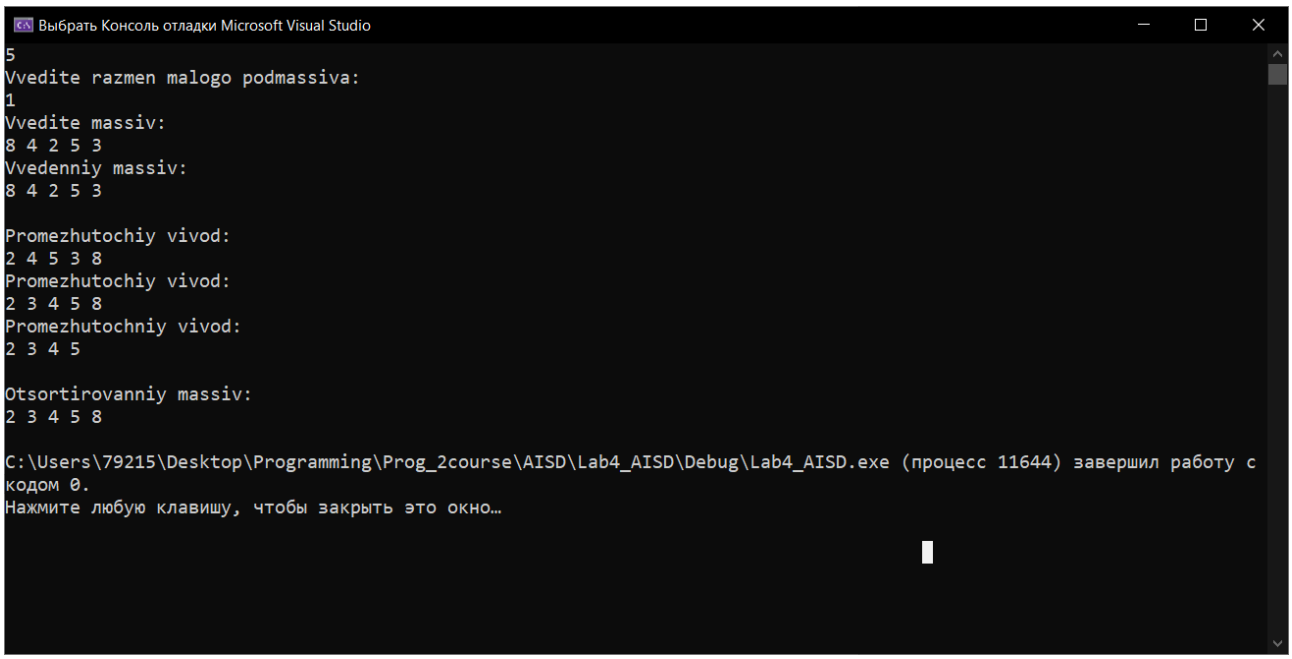

Рисунок 5 - Пример работы с входными данными № 5

Разработанный программный код см. в приложении А.

Выводы.

Были изучены быстрая сортировка и сортировка вставками, а так же реализована программа при помощи языка C++.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл qsort.cpp

```
#include <iostream>
using namespace std;


template <class Item>
void read(Item arr[], int n){    //считывание массива
	for(int i = 0; i < n; i++)
		cin >> arr[i];
}


template <class Item>
void print(Item arr[], int n){    //печать массива
	for(int i = 0; i < n; i++)
		cout << arr[i] << ' ';
	cout << "\n";
}


template <class Item>
void exch(Item &a, Item &b){   //перестановка элементов
	int tmp;
	tmp = a;
	a = b;
	b = tmp;
}


template <class Item>
void comepxch(Item &a, Item &b){
	if(b < a)
		exch(a, b);
}


template <class Item>
void insertion(Item arr[], int l, int r){    //сортировка вставками
	for(int i = r; i > l; i--)
		comepxch(arr[i-1], arr[i]);
	cout << "Promezhutochniy vivod: \n";
	print(arr, r);
	for(int i = l + 2; i <= r; i++){
		int j = i;
		Item v = arr[i];
		while(v < arr[j-1]){
			arr[j] = arr[j-1];
			j--;
		}
		arr[j] = v;
	}
}


template <class Item>
int partition(Item arr[], int l, int r){    //разбиение
```

```
	int i = l - 1;
	int j = r;
	Item v = arr[r];
    for (;;)
	{
		while (arr[++i] < v);
		while (v < arr[--j])
			if (j == l)
				break;
		if (i >= j)
			break;
		exch(arr[i], arr[j]);
    }
    exch(arr[i], arr[r]);
    return i;
}


template <class Item>
void quickSort(Item arr[], int l, int r, int M){   //быстрая сортировка
	if (r-l <= M)
		return;
	exch(arr[(l+r)/2], arr[r-1]);
	comepxch(arr[l], arr[r-1]);
	comepxch(arr[l], arr[r]);
	comepxch(arr[r-1], arr[r]);
	cout << "Promezhutochiy vivod:\n";
	print(arr, r+1);
	int i = partition(arr, l+1, r-1);
	quickSort(arr,l, i-1, M);
	quickSort(arr, i+1, r, M);
}


template <class Item>
void hybridSort(Item arr[], int l, int r, int M){   //быстрая сортировка,
совмещенная с сортировкой вставками
	quickSort(arr, l, r, M);
	insertion(arr, l, r);
}


int main(){
	int n, M;
	cout << "Vvedite kolichestvo elementov v massive:\n";
	cin >> n;
	cout << "Vvedite razmen malogo podmassiva:\n";
	cin >> M;
	cout << "Vvedite massiv:\n";
	int *arr = new int[n];
	read(arr, n);
	cout << "Vvedenniy massiv:\n";
	print(arr, n);
	cout << "\n";
	hybridSort(arr, 0, n-1, M);
	cout << "\nOtsortirovanniy massiv:\n";
	print(arr, n);
	delete[] arr;
	return 0;
}
```