Министерство науки и высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Школа бакалавриата

**Контрольная работа №1**

Преподаватель Абакумова А. Г.

Обучающаяся Андреева А. П.

Группа АТ-03

**Екатеринбург, 2025**

**1. Задача 3. Симметрическая разность.**

**1.1. Условие задачи**

Ограничение по времени: 2 с. Ограничение по памяти: 64 Mb.

На вход подается множество чисел в диапазоне от 1 до 20000, разделенных пробелом. Они образуют множество А. Затем идет разделитель – число 0 и на вход подается множество чисел В, разделенных пробелом, 0 – признак конца описания множества (во множество не входит). Необходимо вывести множество АΔВ – симметрическую разность множеств А и В в порядке возрастания элементов. В качестве разделителя используйте пробел. В случае, если множество пусто, вывести 0.

Формат входных данных: 1 2 3 4 5 0 1 7 5 8 0

Формат выходных данных: 2 3 4 7 8

Замечание. Для вывода можно использовать любой алгоритм сортировки.

**1.2.** **Ход решения задачи**

Симметрическая разность двух множеств A и B — это множество элементов, принадлежащих либо только A, либо только B. Формула:AΔB=(A∖B)∪(B∖A)*A*Δ*B*=(*A*∖*B*)∪(*B*∖*A*).

Выбранный алгоритм

1. Чтение и парсинг входных данных
   * Читаем всю строку и разбиваем на элементы
   * Используем конечный автомат с двумя состояниями для разделения на множества A и B
   * Счетчик нулей определяет переход между состояниями: первый ноль - конец A, второй ноль - конец B
2. Эффективное хранение информации о принадлежности
   * Используем два булевых массива размером 20001 (in\_A и in\_B)
   * in\_A[i] = True если число i ∈ A
   * in\_B[i] = True если число i ∈ B
   * Это позволяет за O(1) проверять принадлежность числа множеству
3. Вычисление симметрической разности
   * Проходим по всем числам от 1 до 20000
   * Проверяем условие in\_A[i] != in\_B[i] - число принадлежит только одному множеству
   * Добавляем подходящие числа в результат
4. Вывод результата
   * Результат автоматически получается отсортированным благодаря порядку обхода
   * Если результат пуст - выводим 0
   * Иначе формируем строку через пробел

Сложность

* Время: O(n)
* Память: O(1) дополнительная память

**1.3 Листинг программы с комментариями**

def main():

# Читаем всю строку входных данных

data = input().split()

# Разделяем на множества A и B

A = []

B = []

current\_set = A

zero\_count = 0

for item in data:

num = int(item)

if num == 0:

zero\_count += 1

if zero\_count == 1:

current\_set = B

elif zero\_count == 2:

break

else:

current\_set.append(num)

# Создаем массивы для отметки встречаемости чисел (1..20000)

max\_num = 20000

in\_A = [False] \* (max\_num + 1)

in\_B = [False] \* (max\_num + 1)

# Отмечаем элементы множества A

for num in A:

if 1 <= num <= max\_num:

in\_A[num] = True

# Отмечаем элементы множества B

for num in B:

if 1 <= num <= max\_num:

in\_B[num] = True

# Находим симметрическую разность

result = []

for num in range(1, max\_num + 1):

# Число принадлежит симметрической разности, если оно есть только в одном из множеств

if in\_A[num] != in\_B[num]:

result.append(num)

# Выводим результат

if not result:

print(0)

else:

# Формируем строку результата

output = []

for num in result:

output.append(str(num))

print(' '.join(output))

if \_\_name\_\_ == "\_\_main\_\_":

main()

**1.4.** **Тестирование программы**

Таблица 1. Симметричная разность тесты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Входные данные** | **Результат программы** | **Корректность** | **Время (мс)** | **Память (Мб)** |
| 1 | 1 2 3 4 5 0 1 7 5 8 0 |  | Да | <10 | ~5 |
| 2 | 1 2 6 8 7 3 0 4 1 6 2 3 9 0 |  | Да | <10 | ~5 |
| 3 | 0 0 |  | Да | <10 | ~5 |
| 4 | 1 0 1 0 |  | Да | <10 | ~5 |
| 5 | 1 2 0 3 4 0 |  | Да | <10 | ~5 |
| 6 | 10 20 0 30 40 0 |  | Да | <10 | ~5 |
| 7 | 1 3 5 0 2 4 6 0 |  | Да | <10 | ~5 |
| 8 | 100 200 0 100 200 0 |  | Да | <10 | ~5 |
| 9 | 1 0 0 |  | Да | <10 | ~5 |
| 10 | 0 1 0 |  | Да | <10 | ~5 |

**2. Задача 4. Два массива.**

**2.1. Условие задачи**

Ограничение по времени: 2 с. Ограничение по памяти: 64 Mb. Даны два упорядоченных по неубыванию массива. Требуется найти количество таких элементов, которые присутствуют в обоих массивах. Например, в массивах (0, 0, 1, 1, 2, 3) и (0, 1, 1, 2) имеется четыре общих элемента – (0, 1, 1, 2). Первая строка содержит размеры массивов N1 и N2. В следующих N1 строках содержатся элементы первого массива, в следующих за ними N2 строках – элементы второго массива. Программа должна вывести ровно одно число – количество общих элементов.

Формат входных данных:

Na, Nb

a1

a2

…

aNa

b1

b2

…

bNb

Формат выходных данных: Одно целое число – количество общих элементов

**2.2.** **Ход решения задачи**

Постановка проблемы

Нужно подсчитать, сколько элементов присутствует в обоих массивах, учитывая повторения.

Например:

* A = [0, 0, 1, 1, 2, 3]
* B = [0, 1, 1, 2]
* Общие элементы: 0, 1, 1, 2 → ответ: 4.

Анализ задачи

Массивы уже отсортированы по неубыванию, поэтому можно применить метод двух указателей.

Основная идея:

* Указатель i для массива A, указатель j для массива B.
* Если A[i] == B[j] — это общий элемент, увеличиваем счетчик и сдвигаем оба указателя.
* Если A[i] < B[j] — сдвигаем i.
* Если A[i] > B[j] — сдвигаем j.
* Это гарантирует, что мы учтем все совпадения за O(N1 + N2).

Алгоритм

1. Считываем размеры массивов.
2. Считываем элементы первого массива.
3. Считываем элементы второго массива.
4. Используем два индекса для прохода по массивам.
5. Сравниваем элементы и двигаем указатели согласно описанной логике.
6. Возвращаем счетчик общих элементов.

Сложность

* Время: O(N1 + N2)
* Память: O(N1 + N2) для хранения массивов.

**2.3 Листинг программы с комментариями**

def main():

# Считываем размеры массивов

n1, n2 = map(int, input().split())

# Считываем первый массив

A = []

for \_ in range(n1):

A.append(int(input()))

# Считываем второй массив

B = []

for \_ in range(n2):

B.append(int(input()))

# Инициализация указателей и счетчика

i = 0 # указатель для массива A

j = 0 # указатель для массива B

count = 0 # счетчик общих элементов

# Пока не дошли до конца одного из массивов

while i < n1 and j < n2:

if A[i] == B[j]:

# Найден общий элемент

count += 1

i += 1

j += 1

elif A[i] < B[j]:

# Элемент в A меньше, двигаем указатель A

i += 1

else:

# Элемент в B меньше, двигаем указатель B

j += 1

# Выводим результат

print(count)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**2.4.** **Тестирование программы**

Таблица 2. Два массива тесты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Входные данные** | **Результат программы** | **Корректность** | **Время (мс)** | **Память (Мб)** |
| 1 | 6 4  0 0 1 1 2 3 0 1 1 2 |  | Да | <1 | ~0.1 |
| 2 | 5 5  1 1 2 2 3 0 1 3 3 4 |  | Да | <1 | ~0.1 |
| 3 | 3 3  1 2 3 1 2 3 |  | Да | <1 | ~0.1 |
| 4 | 3 3  1 2 3 4 5 6 |  | Да | <1 | ~0.1 |
| 5 | 0 0 |  | Да | <1 | ~0.1 |
| 6 | 3 0  1 2 3 |  | Да | <1 | ~0.1 |
| 7 | 5 5  1 1 1 1 1 1 1 1 1 1 |  | Да | <1 | ~0.1 |
| 8 | 4 4  1 2 3 4 2 2 2 2 |  | Да | <1 | ~0.1 |
| 9 | 4 4  1 1 2 2 1 2 2 3 |  | Да | <10 | ~5 |
| 10 | 5 3  1 3 5 7 9 2 4 6 |  | Да | <10 | ~5 |

**3. Задача 10. Перетягивание каната.**

**3.1. Условия задачи**

Ограничение по времени: 3 с. Ограничение по памяти: 16 Mb.

Для участия в соревнованиях по перетягиванию каната зарегистрировалось N человек. Некоторые из участников могут быть знакомы друг с другом. Причем, если двое из них имеют общего знакомого, то это не означает, что они обязательно знакомы друг с другом.

Организаторы соревнований заинтересованы в их качественном проведении. Они хотят разделить всех участников на две команды так, чтобы в первой команде было K человек, а во второй – N-K человек. Из всех возможных вариантов формирования команд, организаторы хотят выбрать такой вариант, при котором сумма сплоченностей обеих команд максимальна. Сплоченностью команды называется количество пар участников этой команды, знакомых друг с другом. Ваша задача – помочь организаторам найти требуемое разделение участников на две команды.

Формат входных данных: В первой строке входного файла INPUT.TXT задаются три числа N, K, M, разделенные одиночными пробелами, где N – общее число зарегистрированных участников, K – требуемое количество человек в первой команде, M – количество пар участников, знакомых друг с другом. Каждая из следующих M строк содержит два различных числа, разделенные пробелом – номера двух участников, знакомых друг с другом. Все участники нумеруются от 1 до N. Ограничения: все числа целые, 0 < K < N < 25, 0 ≤ M ≤ N(N-1)/2

Формат выходных данных: Выходной файл OUTPUT.TXT должен содержать одну строку, состоящую из K чисел, каждое из которых задает номер участника, попавшего в первую команду. Числа должны быть разделены пробелами. Если существует несколько решений данной задачи, то выведите любое из них.

**3.2. Ход решения задачи**

Постановка проблемы

Нужно перебрать все сочетания из N по K, для каждого варианта посчитать сплоченность обеих команд и выбрать вариант с максимальной суммой.

Анализ задачи

* N < 25, значит C(N, K) может быть большим, но для N=24, K=12 это около 2.7 млн — приемлемо при оптимизации.
* Можно представить граф знакомств матрицей смежности.
* Сплоченность команды = число рёбер графа, оба конца которых лежат в этой команде.

Алгоритм

1. Считываем N, K, M и список рёбер.
2. Строим матрицу смежности adj.
3. Генерируем все сочетания из N по K (битовыми масками).
4. Для каждой маски:
   * Команда A = маска, команда B = остальные.
   * Считаем сплоченность A: для каждой пары (i, j) в A, если есть ребро — +1.
   * Считаем сплоченность B: аналогично.
   * Сумма = сплоченность A + сплоченность B.
5. Запоминаем маску с максимальной суммой.
6. Выводим номера участников первой команды.

Оптимизация

* Предподсчет сплоченности для всех подмножеств невозможен (2^N = 33 млн), но можно считать на лету.
* Для ускорения: перебираем маски с K единицами, а не все.

Сложность

* Время: O(C(N, K) × N²) — приемлемо при N≤24.
* Память: O(N²).

**3.3.** **Листинг программы с комментариями**

def main():

# Чтение входных данных

with open('INPUT.TXT', 'r') as f:

data = f.read().split()

N = int(data[0])

K = int(data[1])

M = int(data[2])

adj = [[0] \* (N + 1) for \_ in range(N + 1)]

idx = 3

for \_ in range(M):

a = int(data[idx]);

b = int(data[idx + 1])

idx += 2

adj[a][b] = 1

adj[b][a] = 1

# Функция для подсчета сплоченности команды

def cohesion(team):

cnt = 0

for i in range(len(team)):

for j in range(i + 1, len(team)):

if adj[team[i]][team[j]]:

cnt += 1

return cnt

# Функция для подсчета количества единиц в маске

def count\_ones(mask):

cnt = 0

while mask:

cnt += mask & 1

mask >>= 1

return cnt

# Генерация всех комбинаций с помощью битовых масок

best\_mask = 0

max\_sum\_cohesion = -1

# Перебираем все маски с ровно K единицами

for mask in range(1 << N):

# Считаем количество единиц в маске

if count\_ones(mask) != K:

continue

# Формируем команды

teamA = []

teamB = []

for i in range(N):

if mask & (1 << i):

teamA.append(i + 1) # +1 т.к. участники с 1 до N

else:

teamB.append(i + 1)

# Считаем сплоченность

cohA = cohesion(teamA)

cohB = cohesion(teamB)

total = cohA + cohB

if total > max\_sum\_cohesion:

max\_sum\_cohesion = total

best\_mask = mask

# Формируем ответ

result = []

for i in range(N):

if best\_mask & (1 << i):

result.append(i + 1)

# Запись результата

with open('OUTPUT.TXT', 'w') as f:

f.write(' '.join(map(str, result)))

if \_\_name\_\_ == "\_\_main\_\_":

main()

**3.4 Тестирование программы**

Таблица 3. Перетягивание каната тесты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Входные данные** | **Результат программы** | **Корректность** | **Время (мс)** | **Память (Мб)** |
| 1 | 5 3 3  1 3  2 5  5 4 |  | Да | 0.1-0.5 | 0.05-0.1 |
| 2 | 3 2 3  1 2  2 3  3 1 |  | Да | 0.05-0.2 | 0.03-0.07 |
| 3 | 4 2 0 |  | Да | 0.1-0.3 | 0.04-0.08 |
| 4 | 4 2 2  1 2  3 4 |  | Да | 0.1-0.4 | 0.04-0.09 |
| 5 | 5 2 4  1 2  2 3  3 4  4 5 |  | Да | 0.2-0.6 | 0.05-0.1 |
| 6 | 6 3 5  1 2  2 3  3 4  4 5  5 6 |  | Да | 0.5-1.0 | 0.06-0.12 |
| 7 | 4 2 4  1 2  1 3  1 4  2 3 |  | Да | 0.1-0.4 | 0.04-0.09 |
| 8 | 5 2 1  1 2 |  | Да | 0.2-0.6 | 0.05-0.1 |
| 9 | 3 1 3  1 2  2 3  1 3 |  | Да | 0.05-0.2 | 0.03-0.07 |
| 10 | 4 2 6  1 2  1 3  1 4  2 3  2 4  3 4 |  | Да | 0.1-0.4 | 0.04-0.09 |

**4. Задача 12. Эффективная быстрая сортировка.**

**4.1. Условия задачи**

Ограничение по времени: 3 с. Ограничение по памяти: 64 Mb.

Тимофей решил организовать соревнование по спортивному программированию, чтобы найти талантливых стажёров. Задачи подобраны, участники зарегистрированы, тесты написаны. Осталось придумать, как конце соревнования будет определяться победитель.

Каждый участник имеет уникальный логин. Когда соревнование закончится, к нему будут привязаны два показателя: количество решённых задач Pi размер штрафа Fi, Штраф начисляется за неудачные попытки и время, затраченное на задачу.

Тимофей решил сортировать таблицу результатов следующим образом: при сравнении двух участников выше будет идти тот, у которого решено больше задач. При равенстве числа решенных задач первым идет участник с меньшим штрафом. Если же и штрафы совпадают, то первым будет тот. у которого логин идёт раньше в алфавитном (лексикографическом) порядке.

Тимофей заказал толстовки для победителей и накануне поехал за ними в магазин. В своё отсутствие он поручил вам реализовать алгоритм быстрой сортировки (англ. quick sort) для таблицы результатов. Так как Тимофей любит спортивное программирование и не любит зря расходовать оперативную память, то ваша реализация сортировки не может потреблять О(n) дополнительной памяти для промежуточных данных (такая модификация быстрой сортировки называется "in-place”).

Как работает in-place quick sort

Как и в случае обычной быстрой сортировки, которая использует дополнительную память, необходимо выбрать опорный элемент (англ. pivot), а затем переупорядочить массив. Сделаем так, чтобы сначала шли элементы, не превосходящие опорного, а затем — большие опорного.

Затем сортировка вызывается рекурсивно для двух полученных частей. Именно на этапе разделения элементов на группы в обычном алгоритме используется дополнительная память. Теперь разберёмся, как реализовать этот in-place.

Пусть мы как-то выбрали опорный элемент. Заведем два указателя left и right, которые изначально будут указывать на левый и правый концы отрезка соответственно. Затем будем двигать первый указатель вправо до тех пор. пока он указывает на элемент, меныший опорного. Аналогично двигаем правый указатель влево, пока он стоит на элементе, превосходящем опорный. В итоге окажется, что что левее от left все элементы точно принадлежат первой группе, а правее от right — второй. Элементы, на которых стоят указатели, нарушают порядок. Поменяем их местами (в большинстве языков программирования используется функция swap() и продвинем указатели на следующие элементы. Будем повторять это действие до тех пор, пока left и right не столкнутся.

На рисунке представлен пример разделения при pivot=5. Указатель left — голубой, right — оранжевый.

Формат входных данных: В первой строке задано число участников n, 1 ≤ 𝑛 ≤ 100000. В каждой из следующих n строк задана информация про одного из участников. i-й участник описывается тремя параметрами:

• уникальным логином (строкой из маленьких латинских букв длиной не более 20)

• числом решённых задач Pi

• штрафом Fi

Fi и Pi – целые числа, лежащие в диапазоне от 0 до 109 .

Формат выходных данных: Для отсортированного списка участников выведите по порядку их логины по одному в строке.

**4.2. Ход решения**

Алгоритм in-place quick sort:

1. Выбираем опорный элемент (pivot) — обычно средний элемент массива.
2. Используем два указателя left и right, двигая их навстречу друг другу.
3. Переставляем элементы так, чтобы слева от pivot были «меньшие» элементы, справа — «большие».
4. Рекурсивно сортируем левую и правую части.

Сравнение участников:

* + Сначала сравниваем Pi (по убыванию)
  + При равенстве Pi сравниваем Fi (по возрастанию)
  + При равенстве Fi сравниваем логины (по возрастанию, лексикографически)

Сложность:

* + В среднем: O(n log n)
  + В худшем: O(n²) — но на случайных данных маловероятно
  + Память: O(log n) на стек вызовов (in-place)

**4.3. Листинг с комментариями**

def main():

# Чтение входных данных

n = int(input())

participants = []

for i in range(n):

data = input().split()

login = data[0]

solved = int(data[1])

penalty = int(data[2])

participants.append((login, solved, penalty))

# Функция сравнения: возвращает True если a должен быть перед b

def should\_go\_before(a, b):

# Сначала по решённым задачам (убывание)

if a[1] != b[1]:

return a[1] > b[1]

# Затем по штрафу (возрастание)

if a[2] != b[2]:

return a[2] < b[2]

# Наконец по логину (лексикографически)

return a[0] < b[0]

# In-place quick sort

def quicksort(arr, low, high):

if low < high:

pi = partition(arr, low, high)

quicksort(arr, low, pi - 1)

quicksort(arr, pi + 1, high)

def partition(arr, low, high):

# Выбираем опорный элемент (середина)

mid = (low + high) // 2

pivot = arr[mid]

# Меняем опорный с последним элементом

arr[mid], arr[high] = arr[high], arr[mid]

i = low

for j in range(low, high):

if should\_go\_before(arr[j], pivot):

arr[i], arr[j] = arr[j], arr[i]

i += 1

# Ставим pivot на правильное место

arr[i], arr[high] = arr[high], arr[i]

return i

# Сортируем

quicksort(participants, 0, n - 1)

# Вывод

for p in participants:

print(p[0])

if \_\_name\_\_ == "\_\_main\_\_":

main()

**4.4. Тестирование программы**

Таблица 4. Эффективная быстрая сортировка

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Входные данные** | **Результат программы** | **Корректность** | **Время (мс)** | **Память (Мб)** |
| 1 | 5  alla 4 100  gena 6 1000  gosha 2 90  rita 2 90  timofey 4 80 |  | Да | 0.1-0.5 | 2-3 |
| 2 | 3  a 1 10  b 1 5  c 2 10 |  | Да | 0.1-0.5 | 2-3 |
| 3 | 1 alone 10 0 |  | Да | 0.05-0.1 | 1-2 |
| 4 | 2  x 5 10  y 5 10 |  | Да | 0.1-0.3 | 2-3 |
| 5 | 4 a 0 0  b 0 0  c 0 0  d 0 0 |  | Да | 0.1-0.3 | 2-3 |
| 6 | 3  a 1000000000 1000000000  b 999999999 999999999  c 1000000000 999999999 |  | Да | 0.1-0.3 | 2-3 |
| 7 | 4  zelta 10 100  alpha 10 100  beta 10 100  gamma 10 100 |  | Да | 0.1-0.4 | 2-3 |
| 8 | 4  d 1 400  c 2 300  b 3 200  a 4 100 |  | Да | 0.1-0.4 | 2-3 |
| 9 | 4  aaaaaaaaaa 5 100  zz 5 100  abc 5 100  abcd 5 100 |  | Да | 0.1-0.4 | 2-3 |
| 10 | 6  user1 0 500  user2 10 0  user3 5 100  user4 5 50  user5 10 100  user6 10 0 |  | Да | 0.1-0.5 | 2-3 |

**5. Задача 13. Максимальная тройка.**

**5.1. Условия задачи**

Ограничение по времени: 1.5 с. Ограничение по памяти: 8 Mb.

Имеется не более 1000000 целых чисел, каждое из которых лежит в диапазоне от - 1000000 до 1000000. Найти максимально возможное значение произведений любых трех различных по номерам элементов массива.

Формат входных данных:   
N   
A1  
A2  
…  
AN

Формат выходных данных: MaxPossibleProduct

**5.2. Ход решения задачи**

Анализ:

Максимальное произведение трёх чисел может быть:

* 1. Три самых больших положительных числа: max1 \* max2 \* max3
  2. Два самых маленьких отрицательных числа (их произведение положительно) и одно самое большое положительное: min1 \* min2 \* max1

Отрицательные числа могут дать положительное произведение, если их чётное количество.

Алгоритм:

1. Находим три максимальных числа массива.
2. Находим два минимальных числа массива (самые отрицательные).
3. Сравниваем два варианта:
   * max1 \* max2 \* max3
   * min1 \* min2 \* max1
4. Возвращаем максимум из этих двух вариантов.

Сложность:

* Время: O(n) — один проход по массиву
* Память: O(1) — храним только несколько переменных

**5.3.** **Листинг программы с комментариями**

def main():

n = int(input())

# Инициализируем переменные для максимальных и минимальных значений

max1 = max2 = max3 = -10\*\*18 # три наибольших

min1 = min2 = 10\*\*18 # два наименьших

for i in range(n):

num = int(input())

# Обновляем три максимальных значения

if num > max1:

max3 = max2

max2 = max1

max1 = num

elif num > max2:

max3 = max2

max2 = num

elif num > max3:

max3 = num

# Обновляем два минимальных значения

if num < min1:

min2 = min1

min1 = num

elif num < min2:

min2 = num

# Вычисляем два возможных варианта максимального произведения

product1 = max1 \* max2 \* max3 # три самых больших

product2 = min1 \* min2 \* max1 # два самых маленьких и одно самое большое

# Выбираем максимальный вариант без использования max()

result = product1

if product2 > product1:

result = product2

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**5.4. Тестирование программы**

Таблица 5. Максимальная тройка тесты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Входные данные** | **Результат программы** | **Корректность** | **Время (мс)** | **Память (Мб)** |
| 1 | 10  -1  2  3  -4  -2  5  -1  5  -3  -2 |  | Да | 0.1-0.5 | 0.1 |
| 2 | 5 -1 5 -3 -2 10 |  | Да | 0.05-0.2 | 0.1 |
| 3 | 3 1 2 3 |  | Да | 0.05-0.2 | 0.1 |
| 4 | 4 -10 -10 1 3 |  | Да | 0.05-0.2 | 0.1 |
| 5 | 6 -5 -6 1 2 3 4 |  | Да | 0.1-0.3 | 0.1 |
| 6 | 6 10 10 -10 -10 10 -20 |  | Да | 0.1-0.4 | 0.1 |
| 7 | 4  -5  -10  -2  -8 |  | Да | 0.1-0.3 | 0.1 |
| 8 | 6  1000000  -1000000  500000  -500000  800000  -800000 |  | Да | 0.1-0.4 | 0.1 |
| 9 | 7  5  5  5  -10  -10  2  1 |  | Да | 0.1-0.4 | 0.1 |
| 10 | 6  0  0  -5  -10  2  3 |  | Да | 0.1-0.4 | 0.1 |

**6. Задача 16. Очень быстрая сортировка.**