Занятие №7. Сортировка массива

Отсортировать массив — значит расположить (поменять местами) его элементы так, чтобы они следовали в определенном порядке.

Собственно порядки бывают разные. Например, можно отсортировать лексикографически (по алфавиту, как в словаре — в этом случае число 31 должно идти после 251, потому что начинается с цифры три, а она «старше», чем два). Или сначала простые, потом составные, причём в каждой из них числа должны быть выстроены по длине, в случае равенства длины по сумме цифр, а если оба критерия одинаковы, то по значению… В любом случае мы имеем операцию сравнения, при помощи которой определяем, какое из чисел «левее». Поэтому для простоты будем говорить о сортировке по неубыванию, то есть порядок должен стать таким, чтобы следующий элемент был не меньше, чем предыдущий.

Существенных операций участвующих в сортировке выделим **две**:

1) сравнение двух элементов;
2) обмен двух элементов местами.

Для того чтобы разобраться в проблеме, а, возможно, и решить задачу сортировки самостоятельно, попробуйте сыграть в игру. Вы и ваш соперник загадываете последовательность чисел. Скажем, из пяти чисел (2, 5, 1, 6, 4). Каждый по очереди даёт сопернику команды типа «поменяй четвертый элемент со вторым» и «скажи, что больше, второй или третий элемент». Когда вы считаете, что массив уже отсортирован, говорите «стоп». Если массив соперника к этому моменту действительно стал отсортированным — вы выиграли раунд. Если же нет — проиграли. Для того чтобы разобраться в игре, придумать правильный и, возможно, быстрый алгоритм, нужно сыграть несколько раундов.

Рассмотрим два классических алгоритма.

Обмен двух элементов местами не единственный способ перестановки элементов при сортировке.

Единственная операция изменения, допустимая в алгоритме **блинной сортировки** — переворот элементов последовательности до какого-либо индекса. (Представим, что мы лопаточкой подцепляем часть стопки с блинами и всю ее переворачиваем)

Любопытно, что этой задачей занимался Билл Гейтс. В 1979 году онпредставил свой алгоритм и доказал достаточность $(5N + 5)/3$ переворотов и необходимость $17N/16$. Впоследствии этот результат был сильно улучшен.

Будем считать для определенности, что **массив называется `a` и в нём `N` элементов**.

Сортировка выбором (метод минимума)

Найдём минимальный элемент во всем массиве. Он должен стоять на нулевом месте — меняем его местами с нулевым. Теперь находим минимум, начиная с первого (без учёта нулевого — его можно не рассматривать, на нулевом уже всё хорошо) и меняем его с первым. И так далее до предпоследнего элемента.

(Почему до предпоследнего, а не до последнего?)

Опишем сказанное более формально.

Шаги алгоритма:

Для всех i от 0 до N-1

1) Находим номер минимального значения в части массива (переменная min), начиная с i-1-й ячейки до конца массива.
2) Производим обмен элемента с номером min с i-1-й ячейкой.

Таким образом, в программе будет два цикла — внешний, в котором изменяется i, и внутренний, в котором ищется индекс минимального.

Приведём фрагмент кода

```
for (int i = 0; i < N – 1; i++)
{
     min = __;                        // поиск индекса минимального,
     for (int j = i + 1; j < N; j++)    // в части массива,
     {                                // начиная с i-го элемента
          if (______)
          {

               _________
          }
     }
     int t = a[min];
     a[min] = a[i];
     a[i] = t;
}
```

Вычислим общее количество сравнений в алгоритме. На первом проходе делается N-1 сравнение, на втором N-2,..., на последнем одно. Всего получается $\frac{N(N-1)}{2}$ сравнений. Обменов алгоритм делает значительно меньше — в худшем случае $N - 1$.

Немного теории

В оценке эффективности алгоритмов нам важно не конкретное число действий, а «порядок», то есть насколько сильно растёт число действий при увеличении входных данных (для алгоритмов сортировки и многих других — количество чисел в массиве).

Допустим, что один алгоритм делает $100N + 200$ действий, а другой $0.001N^2 + 2$. Какой из них предпочесть? Для малых N, безусловно, второй. Но для больших N (начиная с какого N в данном случае?) второй алгоритм проигрывает в эффективности — количество действий в нём становится больше, а потом гораздо больше, чем в первом. Поэтому при оценке временной эффективности алгоритмов отбрасывают все, кроме самой большой степени многочлена (говорят «полинома»), которым выражается количество действий. Количество действий, которые делает первый алгоритм нашего примера, выражается полином первой степени, а второго — второй. Говорят, что первый алгоритм линейный, а второй квадратичный. Кратко это обозначают и $O(N)$ и$\mathrm{O}(N^2)$ и говорят «О от N», «О от N квадрат».

Общее количество основных действий в алгоритме сортировки выбором получается $\frac{N(N-1)}{2}$ — это полином второй степени. Значит, метод минимума — квадратичный, хотя и линейный по количеству обменов.

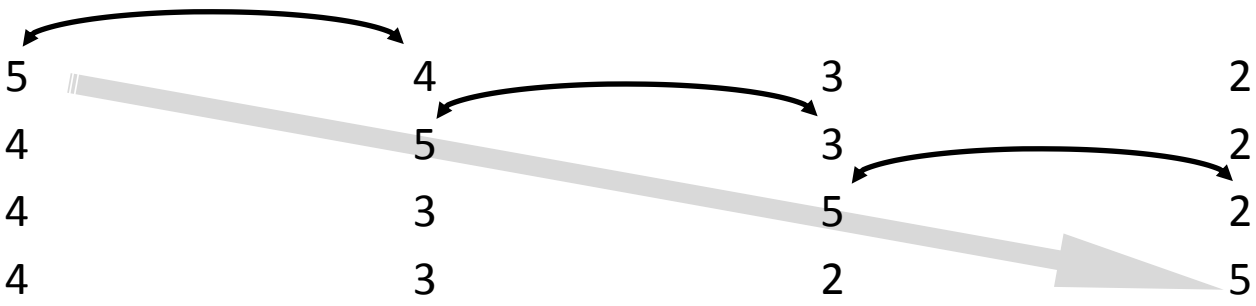
Метод сортировки обменами (метод пузырька)

Этот метод проще, чем метод минимума. Из названия следует, что он основан на постоянных обменах.

Посмотрим на нулевой и первый элементы, если они не в порядке — поменяем их местами. Дальше посмотрим первый и второй и тоже поменяем их в случае нарушения порядка. Так пройдём до конца массива. Ясно, что в результате этих действий самый большой элемент встаёт на своё (последнее) место. Если повторить проход — второй по величине элемент встанет на своё место, и т.д. Сделав *N* - 1 проход по массиву, мы отсортируем массив. От прохода к проходу можно сужать диапазон прохода. Действительно, второй раз можно идти не до предпоследнего элемента, третий до предпредпоследнего и так далее.

Проиллюстрируем сказанное примером. Пусть дан массив из 4 элементов 5, 2, 3, 4

Напишем цепочку обменов во время первого прохода



Пятёрка, как самый большой элемент, участвует во всех сравнениях — «всплывает» на своё место. На следующем проходе «всплывёт» четвёрка, потом тройка. Именно поэтому данный пузырёк на программистском сленге называется «метод пузырька», по-английски «BubbleSort».

Код гениально прост:

```
for (int i = 0; i< ___; i++)       // счетчик проходов
{
     for (int j = i + 1; j< ___;j++)    // проход по массиву
     {
          if (________)
          {
               int t = a[j];  // обмен в случае
               a[j] = a[j + 1];    // необходимости
               a[j + 1] = t;
          }
     }
}
```

Но работает этот метод достаточно медленно. Сравнений в нём получается столько же, сколько и в методе минимума, но обменов в худшем случае — столько же!

Таким образом, этот метод квадратичный и по количеству сравнений и по количеству обменов. А значит, и по общему количеству квадратичный (рассмотрите полином $\frac{N(N-1)}{2} + \frac{N(N-1)}{2}$ - если раскрыть скобки получится полином второй степени).

Кстати, посчитать количество обменов в пузырьковой сортировке, как это требуется, в одной из задач занятия, очень легко. Просто в сам обмен добавим увеличение переменной счетчика.

```
int t = a[j];  // обмен в случае
a[j] = a[j+1];      // необходимости
a[j+1] = t;
k++;
```

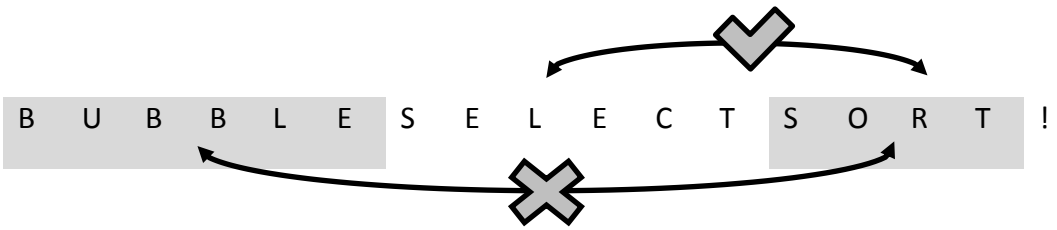
В начале, конечно, надо ее инициализировать нулем.

Тем не менее, несмотря на то, что «пузырек» обычно работает медленно, у этого метода есть достоинства. Он действительно очень прост и хорош для случая почти отсортированного массива. Действительно, если в массиве лишь несколько элементов нарушают порядок, за несколько проходов массив станет упорядоченным! А понять, что массив уже отсортирован, в методе пузырька просто — достаточно посчитать количество обменов, которые мы в ходе него сделали. То есть можно останавливаться тогда, когда на очередном проходе обменов не произошло. Конечно, это не повлияет время в худшем случае — на олимпиаде в тестах обязательно будет такой плохой случай, но для практических целей метод пузырька с этой оптимизацией используется часто.

Попробуйте написать эту оптимизацию самостоятельно или разберите код в Справочнике.

Больше того, бывают случаи, когда обмен соседних элементов — практически единственная допустимая операция при сортировке. Например, мы сортируем объекты в

памяти, у которых разный размер, например строки. В этом случае мы не сможем обменять два произвольных элемента — один из них «не влезет», а вот соседние всегда:



Немного о задачах

Очень важна в идейном плане задача «Результаты олимпиады». В ней даны сведения по участникам олимпиады – номер и количество набранных баллов. Нужно отсортировать участников по убыванию баллов, а в случае равенства баллов – по возрастанию номеров. Посмотрите примеры.

Понятно, что в ней надо завести два массива – один с номерами, другой с баллами участников (назовем их n и m). Но как их потом сортировать? Конечно, нельзя ни в коем случае сортировать их отдельно – номера «оторвутся» от баллов. Когда мы в сортировке меняем элементы первого массива, тут же, синхронно, надо менять и элементы второго.

Но как отсортировать «по баллам, а в случае равных баллов по номерам»? Просто надо сделать, ровно то, что написано: **менять пары в том случае, если или баллы в беспорядке или баллы равны, а участники в беспорядке**. Фрагмент пузырька для этой задачи выглядит так:

```
if (m[i] < m[i + 1] || m[i] == m[i + 1] && n[i] > n[i+1])
{
     // обмен i-ых и i+1-ых элементов массивов n и m
     // ...
}
```

Весьма интересна задача Злые свинки или Anti Angry Birds. Сначала придумайте алгоритм решения этой задачу и только потом отправляйтесь проходить очередной уровень на своем планшете!

Если в решении задачи требуется просто отсортировать массив по возрастанию, конечно, следует использовать стандартную статическую функцию **`sort`** определённую в классе **`Arrays`**.

Например, для сортировки массива с именем«a»мы можем написать

```
Arrays.sort(a);
```

или даже

```
Arrays.sort(a, 0, a.length/2);
```

чтобы отсортировать только первую его половину!

Задачи

Задача A. Пузырьковая сортировка: количество обменов

Определите, сколько обменов сделает алгоритм пузырьковой сортировки по возрастанию для данного массива.

Формат входного файла

На первой строке дано число N ($1 \le N \le 1000$) – количество элементов в массиве. На второй строке – сам массив. Гарантируется, что все элементы массива различны и не превышают по модулю 10^9.

Формат выходного файла

Выведите одно число – количество обменов пузырьковой сортировки.

Примеры

Ввод	Вывод
3 1 3 2	1
2 2 1	1
4 4 1 5 3	3

Задача B. Середина

Какое число окажется в середине, если расставить элементы массива по возрастанию?

Формат входного файла

На первой строке натуральное нечетное N (N < 1000, нечетное) – количество элементов.

На второй N элементов.

Формат выходного файла

Ответ на задачу

Примеры

Ввод	Вывод
5 6 2 7 4 2	4

Задача C. План обменов

Выведите план обменов его ячеек, который приводит массив в отсортированное по неубыванию состояние.

Формат входного файла

На первой строке натуральное нечетное *N* (*N* < 100, нечетное) – количество элементов.

На второй N элементов.

Формат выходного файла

Последовательность из не более 10000 обменов в формате *X*-Y (без пробелов), где X и Y индексы обмениваемых ячеек. Каждый обмен выводите на отдельной строке. После последнего обмена тоже должен следовать перевод строки. Если обменов не требуется – не выводите ничего. Если возможных ответов несколько – выведите любой.

Примеры

Ввод	Вывод	Комментарий
4 3 5 1 4	0-2 3-1	Достаточно обменять нулевую и вторую ячейку, а потом первую и третью местами. Возможны и другие варианты

Задача D. Путешествие первого элемента

Массив сортируется методом выбора по возрастанию (упорядочивание происходит слева направо). Сколько раз меняет свое место первый по порядку элемент?

Формат входного файла

На первой строке число (*N* < 1000) - количество элементов в массиве.

На второй - сам массив. Гарантируется, что все элементы массива различны.

Формат выходного файла

Одно число - количество перемещений первого элемента.

Примеры

Ввод	Вывод
3 1 3 2	0
4 4 1 5 3	3

Комментарий ко второму примеру: 4 перемещается сначала на второе место, потом на 4-е, а потом уже на 3-е.

Задача E. Результаты олимпиады

Во время проведения олимпиады каждый из участников получил свой идентификационный номер – натуральное число. Необходимо отсортировать список участников олимпиады по количеству набранных ими баллов.

Формат входного файла

На первой строке дано число $N (1 \le N \le 1000)$ – количество участников. На каждой следующей строке даны идентификационный номер и набранное число баллов соответствующего участника. Все числа во входном файле не превышают 10^5.

Формат выходного файла

В выходной файл выведите исходный список в порядке убывания баллов. Если у некоторых участников одинаковые баллы, то их между собой нужно упорядочить в порядке возрастания идентификационного номера.

Примеры

Ввод	Вывод
3 101 80 305 90 200 14	305 90 101 80 200 14
3 20 80 30 90 25 90	25 90 30 90 20 80

Задача F. Обувной магазин

Источник: Московская командная олимпиада 2006 года

В обувном магазине продается обувь разного размера. Известно, что одну пару обуви можно надеть на другую, если она хотя бы на три размера больше. В магазин пришел покупатель. Требуется определить, какое наибольшее количество пар обуви сможет предложить ему продавец так, чтобы он смог надеть их все одновременно.

Формат входного файла

Сначала вводится размер ноги покупателя (обувь меньшего размера он надеть не сможет), затем количество пар обуви в магазине и размер каждой пары. Размер — натуральное число, не превосходящее 100, количество пар обуви в магазине не превосходит 1000.

Формат выходного файла

Выведите единственное число — максимальное количество пар обуви.

Примеры

Входные данные	**Выходные данные**
60 2 60 63	2
35 5 30 40 35 42 35	2

Задача G. Субботник

Источник: Московская командная олимпиада 2009 года. Лига Б

В классе учатся *N* человек. Классный руководитель получил указание разбить их на *R* бригад по *C* человек в каждой и направить на субботник (*N* = *R*·*C*).

Все бригады на субботнике будут заниматься переноской бревен. Каждое бревно одновременно несут все члены одной бригады. При этом бревно нести тем удобнее, чем менее различается рост членов этой бригады.

Числом неудобства бригады будем называть разность между ростом самого высокого и ростом самого низкого членов этой бригады (если в бригаде только один человек, то эта разница равна 0). Классный руководитель решил сформировать бригады так, чтобы максимальное из чисел неудобства сформированных бригад было минимально. Помогите ему в этом!

Рассмотрим следующий пример. Пусть в классе 8 человек, рост которых в сантиметрах равен 170, 205, 225, 190, 260, 130, 225, 160, и необходимо сформировать две бригады по четыре человека в каждой. Тогда одним из вариантов является такой:

1 бригада: люди с ростом 225, 205, 225, 260

2 бригада: люди с ростом 160, 190, 170, 130

При этом максимальное число неудобства будет во второй бригаде, оно будет равно 60, и это наилучший возможный результат.

Формат входных данных

Сначала вводятся натуральные числа R и C — количество бригад и количество человек в каждой бригаде ($1 \le R \cdot C \le 1000$). Далее вводятся $N = R \cdot C$ целых чисел — рост каждого из N учеников. Рост ученика — натуральное число, не превышающее 1 000 000 000.

Формат выходных данных

Выведите одно число — наименьше возможное значение максимального числа неудобства сформированных бригад.

Примеры

Входные данные	Выходные данные
2 4 170 205 225 190 260 130 225 160	60

Задача H. Такси

Источник: Всероссийская заочная олимпиада 2006 года

После затянувшегося совещания директор фирмы решил заказать такси, чтобы развезти сотрудников по домам. Он заказал N машин — ровно столько, сколь у него сотрудников. Однако когда они подъехали, оказалось, что у каждого водителя такси свой тариф за 1 километр.

Директор знает, какому сотруднику сколько километров от работы до дома (к сожалению, все сотрудники живут в разных направлениях, поэтому нельзя отправить двух сотрудников на одной машине). Теперь директор хочет определить, какой из сотрудников на каком такси должен поехать домой, чтобы суммарные затраты на такси (а их несет фирма) были минимальны.

Формат входных данных

Сначала во входном файле записано натуральное число N ($1 \le N \le 1000$) — количество сотрудников компании (совпадающее с количеством вызванных машин такси). Далее записано N чисел, задающих расстояния в километрах от работы до домов сотрудников компании (первое число — для первого сотрудника, второе — для второго и т.д.). Все расстояния — положительные целые числа, не превышающие 1000. Далее записано еще N чисел — тарифы за проезд одного километра в такси (первое число — в первой машине такси, второе — во второй и т.д.). Тарифы выражаются положительными целыми числами, не превышающими 10000.

Формат выходных данных

В выходной файл выведите *N* чисел. Первое число — номер такси, в которое должен сесть первый сотрудник, второе число — номер такси, в которое должен сесть второй и т.д., чтобы суммарные затраты на такси были минимальны. Если вариантов рассадки сотрудников, при которых затраты минимальны, несколько, выведите любой из них.

Примеры

Входные данные	**Выходные данные**
3 10 20 30 50 20 30	1 3 2
5 10 20 1 30 30 3 3 3 2 3	1 2 3 5 4

Задача I. Злые свинки или Anti Angry Birds

Источник: Московская командная олимпиада, 2011 года, лига Б

Вы никогда не задумывались, почему в Angry Birds у птиц нет крыльев? Тем же вопросом задались разработчики новой игры. В их версии смысл игры прямо противоположен Angry Birds: зеленая свинка стреляет по злым птицам из лазерного ружья (завязка явно не хуже исходной игры).

Птицы в игре представляются точками на плоскости. Выстрел сбивает только ближайшую птицу находящуюся на линии огня. При этом сбитая птица падая сбивает всех птиц, находящихся ровно под ней. Две птицы не могут находиться в одной точке. По заданному расположению птиц необходимо определить, какое минимальное количество выстрелов необходимо, чтобы все птицы были сбиты.

Входные данные

Первая строка входного файла содержит единственное целое число N $1 \le N \le 1000$ — количество птиц.

Следующие N строк содержат по два натуральных числа каждая x_i, y_i — координаты i-ой птицы ($0 < x, y \le 10^9$). Свинка находится в точке с координатами (0, 0).

Выходные данные

Единственная строка выходного файла должна содержать одно целое число — минимальное количество выстрелов, необходимое для того, чтобы сбить всех птиц.

Примеры

Входные данные	**Выходные данные**

6 1 1 2 2 3 3 2 1 3 2 3 1	3
6 1 1 2 2 3 3 2 1 3 2 3 4	3

Задача J. Военная сортировка

Источник: Всероссийская командная олимпиада школьников 2002 года

Фирма *Macrohard* получила заказ от армии одной страны на реализацию комплекса программного обеспечения для нового суперсекретного радара. Одной из наиболее важных подпрограмм в разрабатываемом комплексе является процедура сортировки.

Однако в отличие от обычной сортировки, эта процедура должна сортировать не произвольный массив чисел, который передается ей на вход, а специальный заранее заданный массив из *N* чисел, в котором записана некоторая фиксированная перестановка чисел от 1 до *N*, и кроме того, ни одно число в нем изначально не находится на своем месте (то есть на позиции с номером *i* изначально не находится число *i*).

В связи с повышенными требованиями к надежности комплекса в процессе сортировки разрешается выполнять единственную операцию - менять местами два соседних элемента массива. Кроме того, в связи с необходимостью соответствия уставу, не разрешается изменять положение числа, которое уже находится на своем месте.

Например, если массив из 6 элементов в некоторый момент имеет вид <2, 1, 3, 6, 4, 5>, то можно поменять местами 1 и 2, 6 и 4 или 4 и 5, а менять местами 1 и 3 или 3 и 6 нельзя, поскольку число 3 находится на своем месте (на позиции с номером 3).

Вам дан входной массив и поставлено важное задание. Найти последовательность обменов (не обязательно кратчайшую), сортирующую массив и удовлетворяющую приведенным условиям.

Подсказка

Найти такую последовательность обменов всегда возможно.

Формат входных данных

В первой строке вводится целое число *N* - размер входного массива (1 <= *N* <= 100). Вторая строка содержит *N* целых чисел - исходную перестановку чисел от 1 до *N* в массиве. Изначально ни одно число не стоит на своем месте.

Формат выходных данных

Выведите *K* строк, где *K* - количество обменов в Вашей сортировке. На каждой строке выведите по два числа x_i и y_i, разделенных пробелом - позиции в массиве, числа на которых следует поменять местами на *i*-ом обмене. Помните, что должно выполняться условие $|x_i - y_i| = 1$ и что нельзя перемещать число, которое уже стоит на своем месте.

Пример

Входные данные	**Выходные данные**
6 2 3 1 6 4 5	2 3 1 2 4 5 5 6

В приведенном примере массив последовательно имеет следующий вид:

```
исходный вид массива                        2 3 1 6 4 5
поменяли местами числа на 2 и 3 позициях    2 1 3 6 4 5
поменяли местами числа на 1 и 2 позициях    1 2 3 6 4 5
поменяли местами числа на 4 и 5 позициях    1 2 3 4 6 5
поменяли местами числа на 5 и 6 позициях    1 2 3 4 5 6
```

Разбор. Занятие 7

7A. Пузырьковая сортировка: количество обменов

Разбирается в тексте занятия.

7B. Середина

Нужно просто отсортировать массив любым способом и вывести средний элемент (по условию он всегда есть), т.е. элемент с индексом N/2.

7C. План обменов

При сортировке при обмене просто выводим индексы обмениваемых ячеек. Например, в пузырьке:

```
...
if (a[j + 1] < a[j])
{
     int t = a[j];  // обмен в случае
     a[j] = a[j + 1];    // необходимости
     a[j + 1] = t;
     out.println(j + "-" + (j + 1));
}
```

7D. Путешествие первого элемента

Гарантия того, что все элементы различны сильно упрощает решение. Нужно сначала, до сортировки, запомнить значение первого элемента массива,

```
int first = a[0];
int moves = 0;
```

а при обмене проверять, участвует ли он в обмене. И, если да, то увеличивать счетчик перемещений:

```
if (a[i] == first || a[j] == first)
{
     moves++;
}
```

7E. Результаты олимпиады

Разбирается в тексте занятия.

7F. Обувной магазин

Отсортируем обувь по неубыванию размера и применим жадный алгоритм: как только размер очередной пары больше последней "надетой" хотя бы на 3 размера, увеличиваем счетчик количества пар и меняем последний использованный размер. В качестве первой

пары нам подходит минимальная из тех, которые больше или равны размеру ноги покупателя.

7G. Субботник

Отсортируем массив и разобьем по тройкам (по ограничениям в лиге Б разбивается хорошо, потому что $N = RC$). Понятно, что разбивать надо по порядку – если сделать тройки из несоседних элементов, число неудобства получится получится хуже. Дальше посчитаем для всех троек числа неудобства и из них выберем максимум.

7H. Такси

Нужно сажать самых «далекоживущих» в самое дешевое такси. Отсортируем массив тарифов такси по возрастанию, а массив расстояний по убыванию. Но ведь требуется вывести не просто сумму, а номера такси. Заведем еще два массива. Массив номеров такси и массив номеров пассажиров (заполним их последовательно). И про сортировке первых массивов будем менять элементы в соответствующих массивах номеров, подобно тому, как это делается в задаче «Результаты олимпиады». В конце просто выведем массив номеров пассажиров.

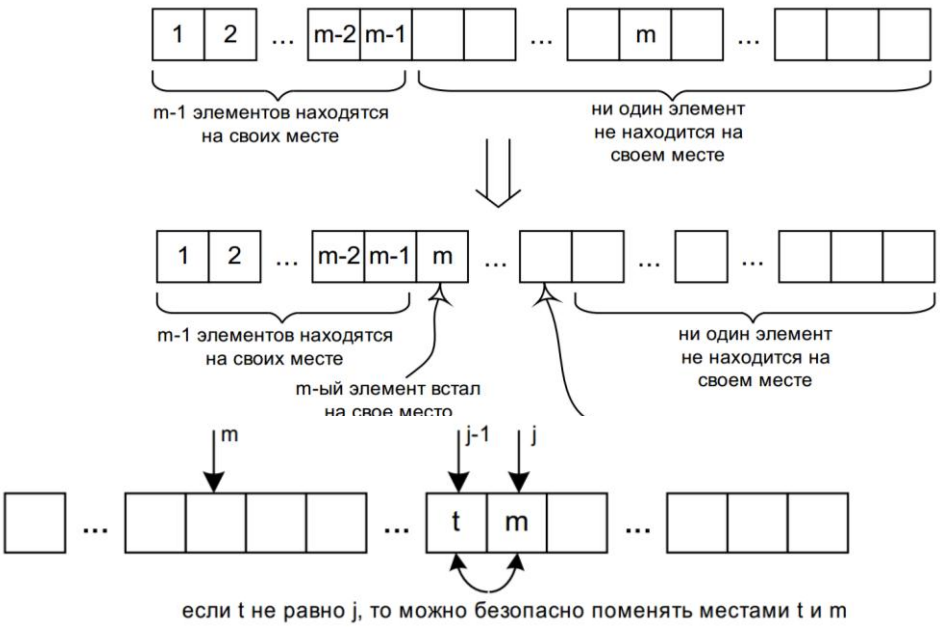
7I. Злые свинки или Anti Angry Birds

Весьма интересна в идейном плане. Мы будем должны сбить все вертикали, а это значит, что сколько и каких птиц, с какими ординатами (игреками) на каждой вертикали нам не важно. Можно даже не читать массив «игреков», а ограничится чтением массива «иксов». Дальше нужно узнать, сколько различных чисел в этом массиве. Для этого часто применяют сортировку. Отсортируем массив, а как проверить количество элементов в монотонном массиве, мы уже знаем из предыдущих занятий. Пройдемся по нему и посчитаем количество элементов, которые не совпадают со следующим. Ответ будет на единицу больше.

7J. Военная сортировка

Это задача с Всероссийской командной олимпиады. Она кажется достаточно сложной. Но такое впечатление создается в основном благодаря грамотному и очень подробному официальному разбору. Важно, что в нем описаны не только действия решающего алгоритма, но и доказано его правильность. Приведем его текст целиком.

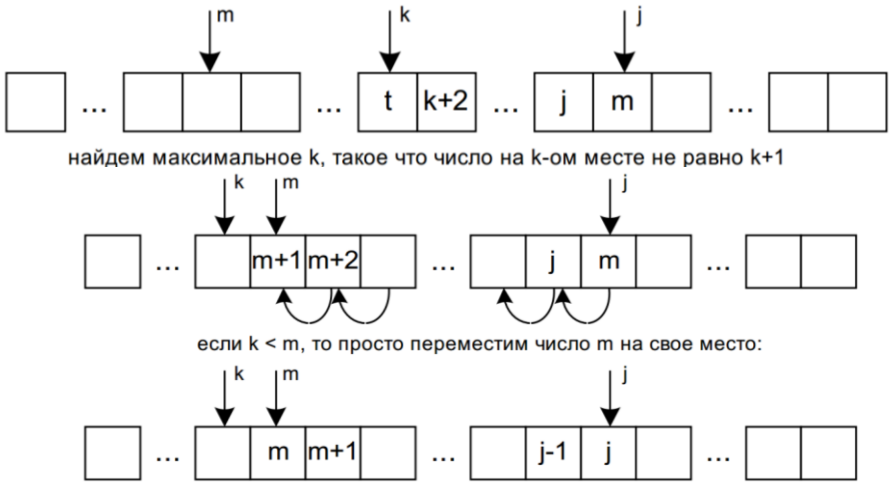
Будем устанавливать элементы массива на свои места по очереди. Пусть первое $m - 1$ число уже находится на своем месте. Рассмотрим оставшуюся часть массива. Предположим, что в ней ни одно число не находится на своем месте. Рассмотрим минимальное число в оставшейся части массива. Оно равно m. Поставим себе целью поставить его на свое место таким образом, чтобы при этом не возникла ситуация, когда некоторое число оказывается на своем месте и слева от него имеются числа, которые еще не находятся на своих местах.



Будем решать задачу индукцией по расстоянию от минимального числа до его места. Если минимальное число уже стоит на своей позиции в массиве, то подзадача решена. Пусть мы умеем решать поставленную подзадачу в случае если ни одно число (кроме, возможно, минимального) в массиве не находится на своем месте и минимальное число находится в массиве на i-ом месте, где $m \le i < j$. Научимся решать эту задачу в случае, если минимальное число находится на j-ом месте. Если число на j – 1-ой позиции не равно j, то его можно поменять с минимальным числом, поставленное нами условие не нарушится и минимальное число приблизится к своему месту, получится задача, которую мы уже умеем решать по индукционному предположению.

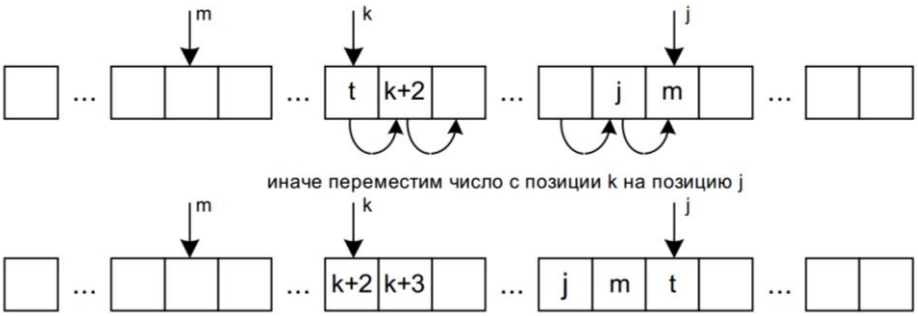
Пусть теперь число на позиции j – 1 равно j. Найдем максимальное число $k < j$ такое,

что число на k-ой позиции не равно k + 1 (положим k = 0, если таких нет).



Если k = m – 1 (то есть все числа, предшествующие минимальному в оставшейся части массива, находятся на одну позицию левее своего места), то просто последовательными обменами переместим число m на свое место. При этом числа с m + 1 по j также станут на свои места.

Если же k > m, то переместим число, стоящее на k-ой позиции на j-ую позицию последовательными обменами. При этом ни одно число не окажется на своем месте. После этой операции число m окажется на позиции с номером j – 1 ≤ j и следовательно



получится задача, которую мы умеем решать по индукционному предположению.

Таким образом мы можем последовательно поставить на свое место все числа в массиве. Итак, мы построили алгоритм решения задачи и параллельно доказали факт, указанный в условии в качестве подсказки – решение всегда существует.

Занятие 7.Справочник

Функция сортировки по неубыванию выбором

```
static void selectSort(int[] a)
{
    int N = a.length;
    for (int i = 0; i < N – 1; i++)
    {
        min = i;
        for (int j = i + 1; j < N; j++)
        {
            if (a[j] < a[min])
            {
                min = j;
            }
        }
        int t = a[min];
        a[min] = a[i];
        a[i] = t;
    }
}
```

Функция сортировки по неубыванию обменами с оптимизацией

```
static void bubbleSort(int[] a)
{
    int N = a.length;
    int count = 1; // чтобы войти в цикл первый раз
    for (int i = 0; i < N – 1 && count > 0; i++)
    {
        count = 0;
        for (int j = 0; j < N – 1 – i; j++) // проход по массиву
        {
            if (a[j] > a[j + 1])
            {
                int t = a[j];   // обмен в случае
                a[j] = a[j + 1];// необходимости
                a[j + 1] = t;
                count++;
            }
        }
    }

}
```

Встроенные функции сортировки класса **`Arrays`**

`static void`	**`sort`**`(int[] a)` Sorts the specified array of ints into ascending numerical order.
`static void`	**`sort`**`(int[] a, int fromIndex, int toIndex)` Sorts the specified range of the specified array of ints into ascending numerical order