1

На старт, внимание, сортировка!

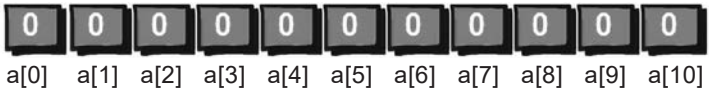
1.1. Быстро и просто — блочная сортировка

Мир, в котором мы живем, полон вещей, которые подлежат сортировке. Люди могут быть рассортированы по росту, доходам, оценкам, полученным на экзамене, и т. п. Покупки в интернете сортируются по цене, письма в почте — по времени получения... Короче говоря, можно сказать, что сортировка повсеместна. Рассмотрим пример для знакомства с алгоритмами сортировки.

После экзамена учитель хочет упорядочить оценки по убыванию. В классе всего пять учеников, которые набрали 5, 3, 5, 2 и 8 баллов, что очень плохо (высшая оценка — 10 баллов). Далее оценки сортируются от наибольшей к наименьшей и приобретают вид следующей последовательности: 8, 5, 5, 3, 2. Как будет работать программа, которая позволит компьютеру считать пять чисел и затем вывести их в порядке от наибольшего к наименьшему? Пожалуйста, подумайте над этим хотя бы 15 минут, прежде чем двигаться дальше (*^__^*).

Мы можем решить эту задачу с помощью одномерного массива. Хорошо, если вы самостоятельно пришли к этой идее.

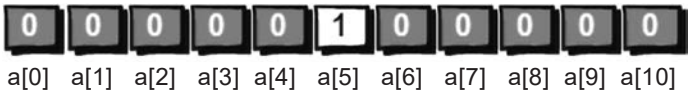
Сначала нужно объявить массив `int a[11]` размером 11. Итак, теперь у нас есть 11 переменных, пронумерованных от a[0] до a[10]. Вначале мы инициализируем каждую из них значением 0, поскольку ни одна оценка еще не проверена. Например, a[0] = 0 означает, что никто еще не набрал 0 баллов, и аналогично a[10] = 0 означает, что никто еще не набрал 10 баллов.



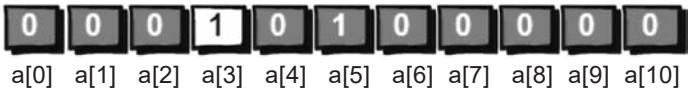
Индексы массива соответствуют оценкам от 0 до 10.

Значение элемента массива — число учеников, получивших определенную оценку

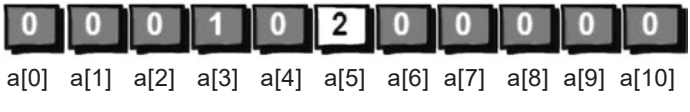
Далее начнем разбираться с каждой из оценок. Первый ученик получил 5 баллов, поэтому значение a[5] увеличивается на 1, то есть оценка 5 баллов встретилась один раз.



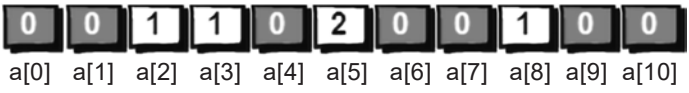
Второй ученик получил оценку 3. Следовательно, мы увеличиваем исходное значение a[3] на 1 — меняем с 0 на 1, указывая, что оценка 3 пока встретилась один раз.



Оценка третьего ученика также равна 5, поэтому значение a[5] необходимо увеличить на 1, то есть изменить с 1 на 2. Это указывает, что оценка 5 встретилась уже дважды.



Обработайте по той же методике четвертую и пятую оценки. Конечным результатом является следующая схема.



Как вы заметили, значения элементов массива a[0]~a[10] соответствуют количеству появлений каждой оценки от 0 до 10. Далее нам нужно вывести оценки, которые встретились один раз или более:

a[0] = 0 означает, что оценка 0 не встретилась и не выводится;

a[1] = 0 означает, что оценка 1 не встретилась и не выводится;

a[2] = 1 означает, что оценка 2 появилась один раз, поэтому нужно вывести 2;

a[3] = 1 означает, что оценка 3 встретилась один раз и нужно вывести 3;

a[4] = 0, означает, что оценка 4 не встретилась и не выводится;

a[5] = 2 означает, что оценка 5 встретилась два раза, поэтому нужно вывести 5 5;

a[6] = 0 означает, что оценка 6 не встретилась и не выводится;

a[7] = 0 означает, что оценка 7 не встретилась и не выводится;

a[8] = 1 означает, что оценка 8 встретилась один раз и нужно вывести 8;

a[9] = 0 означает, что оценка 9 не встретилась и не выводится;

a[10] = 0 означает, что оценка 10 не встретилась и не выводится.

Итоговый вывод на экран: 2 3 5 5 8. Полный код выглядит следующим образом:

```
#include <stdio.h>
int main()
{
    int a[11],i,j,t;
    for(i=0;i<=10;i++)
        a[i]=0; // инициализируется равной 0

    for(i=1;i<=5;i++) // цикл для считывания пяти чисел
    {
        scanf("%d",&t); // считываем каждое число в переменную t
        a[t]++; // заполняем массив
    }

    for(i=0;i<=10;i++) // поочередно оцениваем a[0]~a[10]
        for(j=1;j<=a[i];j++) // выводим результаты
            printf("%d ",i);

    getchar(); getchar();
    // здесь getchar(); используется для приостановки программы, чтобы
    // посмотреть, что она выводит
    return 0;
}
```

Исходные данные:

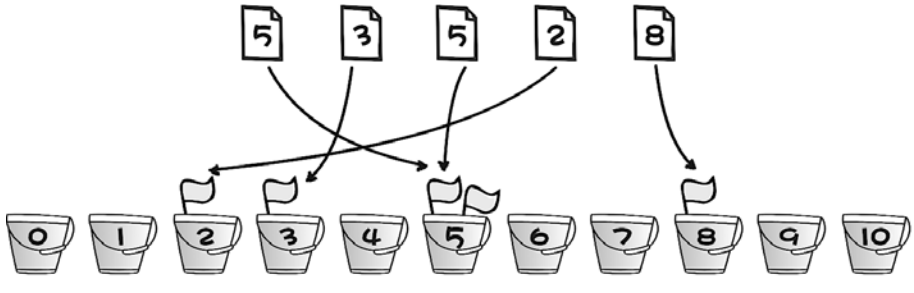
```
5 3 5 2 8
```

Однако здесь результаты сортируются по возрастанию, а требуется сортировать от большого к малому, как нам быть? Сначала подумайте над этим самостоятельно, а потом посмотрите решение ниже.

На самом деле все очень просто: замените `for(i=0;i<=10;i++)` на `for(i=10;i>=0;i--`, и все будет в порядке. Попробуйте!

Этот метод сортировки называется **блочным**. Иногда его образно называют также **сортировкой по ведрам**. Блочная сортировка может быть и более сложной, о чем мы подробно поговорим позже, но на данный момент нашим потребностям вполне удовлетворяет описанный выше алгоритм.

Итак, у нас есть 11 ведер, пронумерованных от 0 до 10. Для каждого числа, считанного из исходных данных, мы кладем флажок в соответствующее пронумерованное ведро, а в конце подсчитываем, сколько флажков находится в каждом ведре. Например, если в ведре 2 находится один флажок, значит, число 2 встретилось один раз; если в ведре 3 находится один флажок, значит, число 3 встретилось один раз; если в ведре 5 находятся два флажка, значит, число 5 встретилось дважды; если в ведре 8 находится один флажок, значит, число 8 встретилось один раз.



Теперь можно попробовать ввести *n* целых чисел в диапазоне от 0 до 1000 и отсортировать их от наибольшего к наименьшему. Обратите внимание, что если вы хотите отсортировать целые числа в диапазоне от 0 до 1000, то вам понадобится 1001 ведро. Роль каждого ведра фактически заключается в том, чтобы отмечать количество раз, когда встречается соответствующее число.

```
#include <stdio.h>

int main()
{
    int book[1001],i,j,t,n;
```

```
    for(i=0;i<=1000;i++)
        book[i]=0;
    scanf("%d",&n);     // n определяет число значений
    for(i=1;i<=n;i++)   // цикл считывания n чисел и их блочной сортировки
    {
        scanf("%d",&t);  // считываем каждое число в переменную t
        book[t]++;       // кладем флажок в ведро с номером t
    }
    for(i=1000;i>=0;i--) // перебираем ведра с номерами 1000~0
        for(j=1;j<=book[i];j++) // печатаем номер ведра столько раз,
                                // сколько флажков в нем лежит
             printf("%d ",i);

    getchar(); getchar();
    return 0;
}
```

Для проверки можно ввести следующие данные:

```
10
8 100 50 22 15 6 1 1000 999 0
```

Результат выполнения программы:

```
1000 999 100 50 22 15 8 6 1 0
```

Теперь поговорим о понятии под названием «**временна́я сложность**». Первый цикл `for` выполняется m раз (m — количество ведер), второй цикл `for` — n раз (n — количество сортируемых чисел), а третий цикл `for` — $m + n$ раз. То есть весь алгоритм сортировки выполняется $m + n + m + n$ раз. Временную сложность мы обозначаем заглавной буквой O (это называется нотация «O-большое»). Таким образом, временная сложность алгоритма равна $O(m + n + m + n)$ или $O(2 * (m + n))$. При обозначении временной сложности можно пренебречь константами, поэтому в итоге временная сложность нашего алгоритма будет равна $O(m + n)$. Еще один момент: при обозначении временной сложности обычно используются прописные буквы, то есть $O(M + N)$.

Это очень быстрый алгоритм. Блочная сортировка известна с 1956 года, а ее основная идея была разработана Э. Дж. Иссаком (E. J. Issac) и Р. К. Синглтоном (R. C. Singleton). Как уже было сказано, алгоритм блочной сортировки сложнее, чем использовавшийся выше. Однако, учитывая, что это наш первый алгоритм, ограничимся упрощенным вариантом, хотя на самом деле он даже не является алгоритмом сортировки в полном смысле этого слова. Почему? Объясним это на следующем примере.

Представим, что теперь у нас есть имена и оценки пяти человек: Сеня получил 5 баллов, Сева — 3 балла, Ксюша — 5 баллов, Костя — 2 балла, а Гоша — 8 баллов. Нужно вывести их имена в порядке убывания набранных ими баллов. Это означает, что вы должны вывести: Гоша, Сеня, Ксюша, Сева, Костя. Видите про-

блему? Если вы используете упрощенный алгоритм блочной сортировки, то можете только упорядочить оценки, но не получивших их учеников. Другими словами, мы не знаем, каким ученикам соответствуют отсортированные оценки! Что делать? Не волнуйтесь, читайте следующий раздел.

1.2. Рассказ о добрых соседях — сортировка пузырьком

Недостаток упрощенной версии блочной сортировки — не только проблема, про которую я сказал выше, но и, что еще хуже, огромные затраты памяти! Например, если нужно отсортировать числа в диапазоне от 0 до 2,1 миллиардов, то необходимо использовать массив размером 2 100 000 001, то есть объявить int a[2100000001]. Даже если вам нужно отсортировать только пять чисел, например 1, 1 912 345 678, 2 100 000 000, 18 000 000 и 91 2345 678, то все равно потребуется 2 100 000 001 ведро, а это лишняя трата памяти! А что делать, если нужно отсортировать не целые числа, а несколько дробей, например 5,56789; 2,12; 1,1; 3,123 и 4,1234? Давайте познакомимся еще с одним алгоритмом — **сортировка пузырьком**. Он может быть хорошим решением этих двух проблем.

Основная идея сортировки пузырьком заключается в сравнении двух соседних элементов и перестановке, если они расположены в неправильном порядке.

Например, нам нужно расставить числа 12, 35, 99, 18 и 76 от наибольшего к наименьшему. Поскольку они сортируются по убыванию, то чем меньше число, **тем дальше оно находится от начала**. Вы можете подумать, что это несущественно, но на самом деле это правило имеет решающее значение (∩_∩).

Вначале мы сравниваем первое и второе числа — 12 и 35. Так как мы хотим, чтобы меньшее число было дальше от начала, нам нужно поменять их местами. Теперь порядок стал таким: 35, 12, 99, 18, 76.

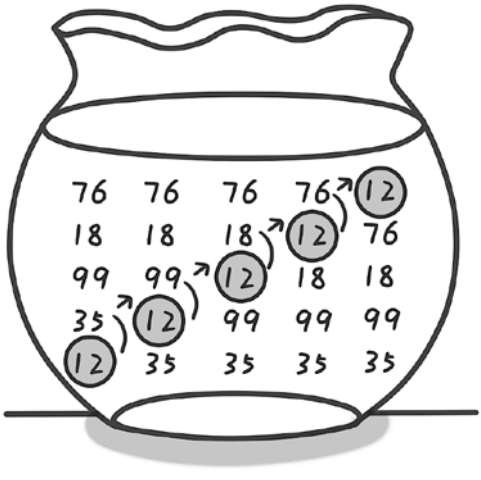
Продолжая действовать по этому правилу, сравниваем второе и третье числа — 12 и 99. Их также необходимо поменять местами. После этой замены порядок будет следующим: 35, 99, 12, 18, 76.

Далее сравним третье и четвертое числа. Поскольку 12 меньше 18, меняем их местами. Порядок чисел после обмена таков: 35, 99, 18, 12, 76.

Наконец, сравниваем четвертое и пятое числа. После четырех сравнений порядок следования чисел будет таким: 35, 99, 18, 76, 12.

После четырех сравнений мы обнаруживаем, что наименьшее число находится на последнем месте. Обратите внимание, как перемещается число 12. Разве это не удивительно? Теперь вспомним, как происходил процесс сравнения. Каждый

раз мы сравниваем два соседних числа, и если большее из них находится дальше от начала, то меняем их местами. Процесс продолжается до тех пор, пока в конце последовательности не окажется наименьшее число. Это похоже на пузырек в жидкости, который постепенно всплывает, пока не достигнет последней позиции (поверхности воды). Поэтому этот метод получил запоминающееся название «сортировка пузырьком».



Сейчас мы поставили на место только наименьшее из пяти чисел. Каждый этап, на котором мы помещаем число на место, называется «проходом». Мы продолжим процесс для оставшихся четырех чисел.

Итак, цель второго прохода — поставить на место второе по величине число. Снова начинаем со сравнения первого и второго чисел и, если первое меньше второго, то меняем их местами. Порядок чисел после этого такой: 99, 35, 18, 76, 12. Далее сравним второе и третье числа, затем — третье и четвертое. Обратите внимание, что на этом этапе нет необходимости сравнивать четвертое и пятое числа, так как после первого прохода можно быть уверенным, что наименьшее число находится на пятом месте. Порядок чисел после второго прохода таков: 99, 35, 76, 18, 12.

Аналогичным образом выполняется третий проход, после которого порядок таков: 99, 76, 35, 18, 12.

Настал черед последнего прохода. Вы спрашиваете: «А разве он нужен?» Действительно, в нашем случае правильный порядок установился уже на предыдущем этапе, но это случайное совпадение. С другими числами этого может не произойти. Можете ли вы найти пример таких данных? Пожалуйста, попробуйте.

Особенность сортировки пузырьком заключается в том, что каждый проход позволяет поставить на место только одно число. То есть на первом проходе мы определяем число, которое размещается на последнем месте (в нашем примере — пятом), на втором проходе — предпоследнее число (в нашем примере — четвертое) и т. п. После третьего прохода в нашем примере установилась правильная последовательность, но все равно нужно выполнить четвертый проход.

На последнем этапе нужно сравнить только первое и второе числа. В нашем примере первое — 99, второе — 76, и менять их местами не нужно. Порядок расположения чисел остается прежним: 99, 76, 35, 18, 12. Пятый проход не требуется, поскольку четыре из пяти чисел уже расставлены по местам и для последнего осталась только одна возможная позиция.

Подведем итог: для сортировки n чисел необходимо расставить по местам только $n - 1$ число, что означает $n - 1$ операций. При этом каждый проход нужно начинать с первой позиции, сравнивая два соседних числа, при необходимости выполнять перестановку и переходить к следующей позиции. Эти шаги повторяются до достижения последнего числа, позиция которого не была определена при предыдущих проходах.

Не правда ли, жесткий алгоритм? Не знаю, чем вдохновлялся человек, придумавший его. Но мы дошли до написания кода. Предлагаю сначала попробовать реализовать алгоритм самостоятельно, а потом посмотреть, как это сделал я.

```
#include <stdio.h>
int main()
{
    int a[100],i,j,t,n;
    scanf("%d",&n); // n -- количество чисел
    for(i=1;i<=n;i++) // цикл ввода n чисел в массив a
        scanf("%d",&a[i]);
    // Основная часть сортировки пузырьком
    for(i=1;i<=n-1;i++) // сортировка в n-1 проходов
    {
        for(j=1;j<=n-i;j++)
    // сравниваем с позиции 1 до n-i, то есть последнего числа,
    // позиция которого не определена
        {
            if(a[j]<a[j+1]) // сравниваем и меняем местами
            { t=a[j]; a[j]=a[j+1]; a[j+1]=t; }
        }
    }
    for(i=1;i<=n;i++) // вывод результатов
        printf("%d ",a[i]);

    getchar(); getchar();
    return 0;
}
```

Для проверки можно ввести следующие данные:

```
10
8 100 50 22 15 6 1 1000 999 0
```

Результат выполнения программы:

```
1000 999 100 50 22 15 8 6 1 0
```

Небольшая модификация приведенного выше кода позволит решить задачу из раздела 1.1.

```
#include <stdio.h>
struct student
{
    char name[21];
    int score;
}; // создается структура для хранения имени и оценки
int main()
{
    struct student a[100],t;
    int i,j,n;
    scanf("%d",&n);   // ввод числа n
    for(i=1;i<=n;i++) // цикл считывания n имен и оценок
        scanf("%s %d",a[i].name,&a[i].score);
    // сортировка оценок от наибольшей к наименьшей
    for(i=1;i<=n-1;i++)
    {
        for(j=1;j<=n-i;j++)
        {
            if(a[j].score<a[j+1].score) // сравнение оценок
            { t=a[j]; a[j]=a[j+1]; a[j+1]=t; }
        }
    }
    for(i=1;i<=n;i++) // вывод имени ученика
        printf("%s\n",a[i].name);

    getchar(); getchar();
    return 0;

}
```

Для проверки можно ввести следующие данные.

```
5
сеня 5
сева 3
ксюша 5
костя 2
гоша 8
```