1 Последовательности

1. Написать функцию, получающую в качестве аргумента имя файла, содержащего последовательность битов неизвестной длины (в файле битовая последовательность задается как последовательность целых чисел, двоичное представление которых (без ведущих 0) образует битовую последовательность), а также целое число k, и возвращающую целое число, равное минимальному расстоянию между соседними участками этой последовательности, составленными из единичных битов и имеющими длину не менее k. Функция должна возвращать -1, -2 и т.д., если она не смогла открыть файл, прочитать элемент и т.д.. Основная программа должна выводить на экран исходную последовательность (как битовую), вызывать эту функцию и отображать результат ее работы.
2. Написать функцию, получающую в качестве аргументов имена двух файлов, содержащих последовательности $a_1, a_2, \ldots$ и $b_1, b_2, \ldots$ вещественных чисел неизвестной длины. Функция должна возвращать целое число, равное 1, если каждый элемент второй последовательности (кроме первого и последнего) равен полусумме элементов первой последовательности с соседними ему номерами: $b_i = (a_{i+1} + a_{i-1})/2$, и 0 в противном случае. Функция должна возвращать -1, -2 и т.д., если она не смогла открыть файл, прочитать элемент и т.д.. Основная программа должна вызывать эту функцию и выводить на экран результат ее работы.
3. Написать функцию, получающую в качестве аргументов имя файла, содержащего последовательность вещественных чисел неизвестной длины, и адрес вещественного числа d, и возвращающую в d вещественное число, равное среднему квадратическому отклонению чисел из этого файла от их среднего арифметического. Возвращаемое значение функции равно длине последовательности в случае успешного завершения и равно -1, -2 и т.д., если она не смогла открыть файл, прочитать элемент и т.д.. Основная программа должна вызывать эту функцию и выводить на экран результат ее работы.
4. Написать функцию, получающую в качестве аргумента имя файла, содержащего последовательность вещественных чисел неизвестной длины, и возвращающую целое число, равное 1, если эта последовательность является арифметической прогрессией, 2, если она является геометрической прогрессией, 3, если постоянна, 4, если в ней недостаточно элементов для принятия решения, и 0 в противном случае. Функция должна возвращать -1, -2 и т.д., если она не смогла открыть файл, прочитать элемент и т.д.. Основная программа должна вызывать эту функцию и выводить на экран результат ее работы.

2 Массивы

1. Написать функцию, получающую в качестве аргументов имя файла, массив $a[n]$ вещественных чисел и целое число n, и возвращающую целое число, равное количеству вхождений подпоследовательности $a[n]$ в последовательность, содержащуюся в заданном файле. Основная программа должна заполнять данными массив (из файла или случайными числами), выводить его на экран, вызывать эту функцию и выводить на экран результат ее работы.
2. Написать функцию, получающую в качестве аргументов массив $a[n]$ вещественных чисел, целое число n, и вещественное число x, и выбрасывающую из массива все элементы, меньшие x (при выбрасывании элемента $a[i]$ все элементы с номером, большим i, сдвигаются на одну позицию к началу массива и длина массива уменьшается на 1), произведя при этом не более $n + O(1)$ перемещений элементов. Функция возвращает длину получившегося в результате массива. Основная программа должна заполнять данными массив (из файла или случайными числами), выводить его на экран, вызывать эту функцию и выводить на экран результат ее работы.
3. Написать подпрограмму, получающую в качестве аргументов массив вещественных чисел, целое число n, являющееся длиной этого массива и целое число k, и циклически сдвигающую элементы массива на k позиций вправо произведя не более $n + O(1)$ перемещений элементов. Основная программа должна заполнять данными массив (из файла или случайными числами), выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.

3 Сортировки

1. Написать функцию, получающую в качестве аргументов неубывающий массив $a[n]$ вещественных чисел, целое число n, являющееся длиной этого массива и вещественное число x, и возвращающую место, где в этот массив можно вставить число x (т.е. целое число i такое, что $a[i] \le x \le a[i+1]$). Место определяется двоичным поиском по следующему алгоритму (*алгоритм деления пополам*).

   Взять первоначально 0 и n в качестве границ поиска элемента; далее, до тех пор, пока границы не совпадут, шаг за шагом сдвигать эти границы следующим образом: сравнить x с $a[s]$, где s – целая часть среднего арифметического границ; если $a[s] < x$, то заменить прежнюю нижнюю границу на $s+1$, а верхнюю оставить без изменений, иначе оставить без изменения нижнюю границу, а верхнюю заменить на s; когда границы совпадут, став равными некоторому числу t, выполнение закончится с результатом t.

   Общее количество сравнений и перестановок элементов в наихудшем случае не должно превышать $\log_2 n + O(1)$. Основная программа должна заполнять данными массив (из файла), выводить его на экран, вводить с клавиатуры число x, вызывать эту функцию и выводить на экран результат ее работы.

2. Написать подпрограмму, получающую в качестве аргументов три массива $a[n]$, $b[m]$, $c[n+m]$ вещественных чисел, где $a[n]$, $b[m]$ не убывают, и целые числа n и m, и строящую по неубывающим массивам $a[n]$, $b[m]$ неубывающий массив $c[n+m]$ слиянием первых двух за $n+m+O(1)$ сравнений и $n+m+O(1)$ пересылок элементов по следующему алгоритму

   Просматриваем очередные элементы $a[i]$, $i = 0,\ldots,n-1$ и $b[j]$, $j = 0,\ldots,m-1$ массивов a и b. Если $a[i] < b[j]$, то $c[k] = a[i]$ и увеличиваем i на 1, иначе $c[k] = b[j]$ и увеличиваем j на 1. Здесь k, $k = 0,\ldots,n+m-1$ – очередной элемент массива c (здесь всегда равен $i+j$).

   Основная программа должна заполнять данными массивы $a[n]$ и $b[m]$ (из файла), выводить их на экран, вызывать эту подпрограмму и выводить на экран результат ее работы – массив $c[n+m]$.

3. Написать функцию, получающую в качестве аргументов неубывающий массив $a[n]$ вещественных чисел, целое число n, являющееся длиной этого массива и вещественное число x, и переставляющую элементы массива так, чтобы вначале шли все элементы, меньшие x, а затем элементы, большие x. Функция возвращает место, где в этот массив можно вставить число x (т.е. целое число i такое, что $a[i-1] \le x \le a[i]$, если x меньше всех элементов массива a, то $i = 0$, если больше — то $i = n$). Решение задачи производится следующим алгоритмом:

   Просматривая массив с начала (линейным поиском), находим i такое, что $a[i] \ge x$. Просматривая массив с конца (линейным поиском), находим j такое, что $a[j] < x$. Если $i \le j$, то меняем $a[i]$ и $a[j]$ местами, i увеличиваем на 1, j уменьшаем на 1. Повторяем эту процедуру пока $i \le j$. После этого все элементы $a[0],\ldots,a[i]$ будут не больше всех элементов $a[j],\ldots,a[n-1]$. Возвращаемым значением функции будет $i = j$.

   Общее количество сравнений элементов в наихудшем случае не должно превышать $n/2 + O(1)$, а пересылок элементов – $3n/2 + O(1)$. Основная программа должна заполнять данными массив (из файла), выводить его на экран, вводить с клавиатуры число x, вызывать эту функцию и выводить на экран результат ее работы: массив $a[n]$ и значение $i = j$.

4. Написать подпрограмму, получающую в качестве аргументов массив $a[n]$ вещественных чисел и целое число n, и переставляющую элементы массива $a[n]$ в неубывающем порядке: $a[0] \le a[1] \le \cdots \le a[n-1]$ по следующему алгоритму (*сортировка двоичной вставкой*):

   Просматривать последовательно $a[1],\ldots,a[n-1]$ и каждый новый элемент $a[i]$ вставлять на подходящее место в уже упорядоченную совокупность $a[0],\ldots,a[i-1]$. Это место определяется алгоритмом деления пополам (см. задачу 1).

   Общее количество сравнений в наихудшем случае не должно превышать $n\log_2 n + O(n)$, а пересылок элементов – $n^2/2 + O(n)$. Основная программа должна заполнять данными массив (из файла или случайными числами), выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.

5. Написать подпрограмму, получающую в качестве аргументов массив $a[n]$ и вспомогательный массив $b[n]$ вещественных чисел, и целое число n, и переставляющую элементы массива $a[n]$ в неубывающем порядке: $a[0] \le a[1] \le \cdots \le a[n-1]$ по следующему алгоритму (*сортировка Неймана*):

   Вначале весь массив рассматривается как совокупность упорядоченных групп по одному элементу в каждом. Слиянием соседних групп (см. задачу 2) получаются упорядоченные группы, каждая из которых содержит два элемента (кроме, может быть, последней группы, которой не нашлось парной). Далее упорядоченные группы укрупняются тем же способом и т.д. Используется вспомогательный массив b[n].

   Общее количество сравнений и пересылок элементов в наихудшем случае не должно превышать $n\log_2 n + O(n)$ (каждое). Основная программа должна заполнять данными массив (из файла или случайными числами), выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.

6. Написать подпрограмму, получающую в качестве аргументов массив $a[n]$ вещественных чисел и целое число n, и переставляющую элементы массива $a[n]$ в неубывающем порядке: $a[0] \le a[1] \le \cdots \le a[n-1]$ по следующему алгоритму (*"быстрая"сортировка*):

   Полагаем $x = a[n/2]$. Алгоритмом задачи 3 находим место, где в этот массив можно вставить число x, т.е. такое i, что все элементы $a[0],\ldots,a[i-1]$ будут меньше или равны всех элементов $a[i],\ldots,a[n-1]$. Затем сортируем каждую из частей (т.е. $a[0],\ldots,a[i-1]$ и $a[i],\ldots,a[n-1]$) этим же алгоритмом. Для уменьшения глубины рекурсии вначале сортируем более короткий массив. Затем вместо повторного вызова процедуры переходим к ее началу с новыми значениями указателя на начало массива a и его длины n.

   Общее количество сравнений в наихудшем случае не должно превышать $n^2/2 + O(n)$, а пересылок элементов – $3n^2/2 + O(n)$. Основная программа должна заполнять данными массив (из файла или случайными числами), выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.

7. Написать подпрограмму, получающую в качестве аргументов массив $a[n]$ вещественных чисел и целое число n, и переставляющую элементы массива $a[n]$ в неубывающем порядке: $a[0] \le a[1] \le \cdots \le a[n-1]$ по следующему алгоритму (*"турнирная"сортировка или алгоритм heapsort*):

   Массив $a[n]$ рассматриваем как бинарное дерево с корнем $a[0]$. Для элемента с номером $a[k]$ потомками являются элементы $a[2*k+1]$ и $a[2*k+2]$, а родителем – элемент $a[(k-1)/2]$. На первом этапе алгоритма превращаем наше бинарное дерево в т.н. упорядоченную пирамиду, т.е. в дерево, в котором всякая цепочка от корня до любого конечного элемента выстроена по убыванию. Для этого для всех $k = 1,\ldots,n-1$ идем от элемента $a[k]$ по цепочке его родителей, продвигая его на свое место (подобно тому, как это делалось в алгоритме сортировки линейной вставкой). После этого этапа алгоритма в корне дерева (т.е. в $a[0]$) будет находиться максимальный элемент массива. На втором этапе алгоритма для всех $k = n-1,\ldots,1$: меняем корень (т.е. $a[0]$) и элемент $a[k]$ местами и рассматриваем массив a как имеющий длину k. В этом массиве восстанавливаем структуру упорядоченной пирамиды, нарушенную помещением элемента $a[k]$ в $a[0]$. Для этого идем от элемента $a[0]$ по цепочкам его потомков, продвигая его на свое место (подобно тому, как это делалось в алгоритме сортировки линейной вставкой, с одним отличием: элемент пирамиды должен быть больше обоих своих потомков).

   Общее количество сравнений в наихудшем случае не должно превышать $4n\log_2 n + O(n)$, а пересылок элементов – $2n\log_2 n + O(n)$. Основная программа должна заполнять данными массив (из файла или случайными числами), выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.

4 Строки

1. Написать функцию

   ```
   int strcspn_ (const char *string1, const char *string2);
   ```

   которая возвращает индекс первого символа в строке, адрес которой определяется значением аргумента `string1`, который принадлежит набору символов, содержащихся в строке, адрес которой определяется значением аргумента `string2` (например, если строка `string1` не содержит ни одного символа из `string2`, то возвращается значение, равное длине `string1`, а если строка `string1` начинается с символа из `string2`, то возвращается 0). Основная программа должна выделять память под строки, вводить с клавиатуры строки `string1` и `string2`, вызывать эту подпрограмму и выводить на экран результат ее работы.
2. Написать функцию, получающую в качестве аргументов массив a строк, целое число n, являющееся длиной этого массива, целое число k и строку s, и выбрасывающую из массива все подряд идущие элементы количеством не менее k, в которые входит строка s; функция возвращает длину получившегося в результате массива (число пересылок элементов массива не должно превосходить $n+O(1)$).
3. Написать функцию, получающую в качестве аргументов массив a строк, целое число n, являющееся длиной этого массива, целое число k и строку s, и выбрасывающую из массива все подряд идущие элементы количеством не менее k, которые состоят только из символов строки s; функция возвращает длину получившегося в результате массива (число пересылок элементов массива не должно превосходить $n+O(1)$).
4. Для текстового файла a.txt назовем матрицей слов матрицу, i-я строка которой состоит из слов i-й строки этого файла. Словом назовем последовательность символов, не содержащую пробельных символов, пробельным символовом назовем символ, содержащийся в заданной строке t. Если слов матрице не достаточно, чем элементов в строке матрице, то слова полагаются равными пустым словам. Если слов больше, чем элементов в строке матрице, то "лишние" не используются. Написать подпрограмму, получающую в качестве аргументов $m \times n$ массив a слов и целые числа m, n, и переставляющую строки в массиве a так, чтобы они шли в порядке невозрастания максимального элемента в строке $\max_{j=1,\ldots,n} a_{ij}$. Основная программа должна вводить t, числа m, n (аргументы командной строки) и массив a (из файла), вызывать эту функцию и выводить на экран результат ее работы.