Упражнения

Реализуйте различные алгоритмы сортировки в виде функций вида:

```
void namesort (vector<int>& arr);
```

Для тестирования полезно создать «тестовую оснастку», то есть небольшую программу для автоматического (или полуавтоматического) проведения тестов.

Например, эта программа может считывать по очереди файлы с различными входными данными (которые могут быть созданы отдельной программой или же вручную), и запускать её для каждой реализованной функции сортировки. Выходной массив нужно проверить на упорядоченность элементов. Для сравнения методов полезно выводить информацию о времени выполнения сортировки. Это можно сделать при помощи функции `clock()` из стандартной библиотеки `<ctime>`.

Можно воспользоваться специально заготовленными файлами. Каждый файл содержит определённый набор из 100 000 чисел. Представлены как неупорядоченные, так и упорядоченные наборы.

`const.in.txt`	целая константа
`seq.in.txt`	последовательность целых чисел в возрастающем порядке
`rseq.in.txt`	последовательность целых чисел в убывающем порядке
`rand.in.txt`	псевдослучайные целые числа в диапазоне $[-1000, 1000]$
`rand_2.in.txt`	псевдослучайные целые числа в диапазоне $[0, 1000]$

Вначале следует считывать небольшое количество чисел (например, 10), чтобы убедиться, что алгоритм запрограммирован верно, а затем это количество можно поэтапно увеличивать до максимального.

Тщательно отформатируйте вывод вашей программы. Пример вывода можно посмотреть в файле `sort.log`.

Простые алгоритмы сортировки

1. **Сортировка простым обменом (метод «пузырька»)** Просматривается неотсортированная часть массива, сравниваются два соседних элемента и, если необходимо, обмениваются. Каждая итерация приводит к «всплыванию», по крайней мере, одного максимального элемента.

2. Улучшите предыдущую версию метода «пузырька». Если за время текущего прохода не произошло ни одного «всплытия», или обмена, то, следовательно, массив отсортирован.

3. **Сортировка простым выбором** В неотсортированной части массива выбирается наименьшее число и обменивается с первым.

4. **Сортировка простыми вставками** В неотсортированной части массива берётся первый элемент и вставляется на соответствующее ему место в отсортированной части.

5. В предыдущем методе, рассмотрите вариант двоичного включения.

Сортировка за линейное время

Все упомянутые выше алгоритмы обладают одним общим свойством: *при сортировке используется только сравнение входных элементов*. При любой сортировке сравнением для обработки n элементов в наихудшем случае нужно произвести не менее $O(n \log n)$ сравнений. При сортировке за линейное время привлекаются дополнительные сведения о входных данных. Это, конечно же, ограничивает универсальность таких алгоритмов.

Далее, предполагается, что каждый из n входных элементов — целое число, принадлежащее интервалу от 0 до k, где k – некоторая целая константа. Если $k = O(n)$, то время работы алгоритма равно $O(n)$.

6. **Сортировка методом подсчёта** Создаётся массив k, в элементы которого k_i записывается число элементов исходного массива x, которые не превосходят i. Затем элемент сортируемого массива x_i помещается в позицию k_i.

7. Модифицируйте предыдущий алгоритм так, чтобы он работал при наличии повторяющихся элементов.

8. **Сортировка методом распределяющего подсчёта** Используется для массива x, в котором многие числа часто повторяются. Создаётся массив k длиной $\max x - \min x + 1$, в элементы которого записываются значения частоты повторения x_i. Затем значения x переопределяются.