Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Сортировки массивов»**

**Выполнила**:

студентка группы 382003-1

Семенова В.А.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2020

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

Требуется написать программу на языке C, которая сортирует в порядке возрастания случайный или введенный пользователем массив элементов типа float следующими алгоритмами:

* Сортировка пузырьком;
* Сортировка Шелла;
* Сортировка слиянием;
* Поразрядная сортировка.

В ходе каждой сортировки необходимо подсчитать количество сравнений и перестановок элементов в массиве и провести тесты, показывающие теоретическую сложность алгоритмов.

# Метод решения

1. **Сортировка пузырьком.**

Шаг сортировки состоит в проходе по массиву сверху вниз, сравнении каждых двух соседних элементов и их перестановке в случае неправильного порядка. При каждом проходе по массиву минимальный элемент из еще не отсортированной части массива «всплывает, словно пузырек» и ставится на границу между отсортированной и не отсортированной частями исходного массива. После сравнения и перестановки происходят в части массива с индекса на единицу больше. Время исполнения сортировки можно сократить, введя «переменную-флаг», отвечающую за наличие перестановок в цикле. В случае, когда перестановок не происходит, программа заканчивает работу преждевременно, так как массив уже отсортирован.

Алгоритм имеет сложность O(), очень медленен и малоэффективен. Сортировка пузырьком устойчива.

1. **Сортировка Шелла.**

Сортировка Шелла является модификацией сортировки простыми вставками. Идея сортировки состоит в сравнении и возможной перестановке не соседних, а стоящих на некотором изменяющемся расстоянии элементов, что позволяет сократить время работы программы. Шаг после каждого прохода по циклу уменьшается (в представленной реализации в два раза), пока не станет равным единице. Сортировка при единичном шаге и есть сортировка вставками. Сложность алгоритма зависит от выбранной последовательности шагов, в худшем случае – это O(), в лучшем - O(),

1. **Сортировка слиянием.**

Сортировка основана на идее объединения двух отсортированных массивов. Исходный массив разделяется на две части примерно равного размера, каждая из которых сортируется отдельно. Затем они объединяются в один.

Сортировка подразумевает использование рекурсивной функции, разбивающей один массив на два до тех пор, пока размерность массивов не достигнет единицы. Единичный массив по определению отсортирован.

При соединении двух отсортированных массивов в один результирующий на каждом шаге в него записывается наименьший из двух первых элементов подмассивов, после чего инкримируются счетчики результирующего массива и подмассива, откуда был взят последний элемент. Когда из одного подмассива «взяли» все элементы, все оставшиеся элементы второго записываются в результирующий.

Сортировка устойчивая, однако требует дополнительную память того же объема, что и размер исходного массива, и одинаково долго работает в лучшем и среднем случаях.

1. **Поразрядная сортировка.**

Сортировка основана на сравнении разрядов элементов массива. Сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда и происходит упорядочивание элементов по результатам этого сравнения, затем сравниваются следующие разряды и выполняются дальнейшие перестановки, при этом сохраняющие порядок меньших разрядов.

В качестве одного разряда берется 1 байт, так же учитывается, что байты записи числа в памяти хранятся в обратном порядке. В записи отрицательных чисел старший бит равен 1, в то время как положительных – 0. Поэтому после побайтовой сортировки необходимо пройти по массиву, переставив отрицательные числа в начало массива.

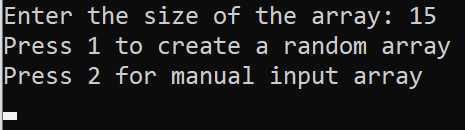
# Руководство пользователя

Пользователь вправе самостоятельно выбрать размер сортируемого массива, способ его заполнения и вид сортировки.

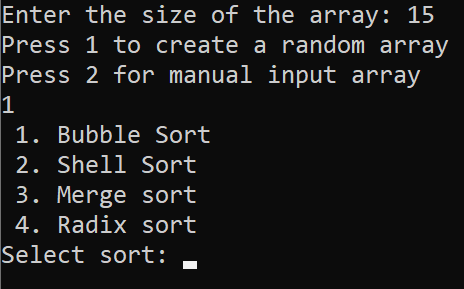
При запуске программы пользователь должен ввести натуральное число – размер сортируемого массива.

размер.png

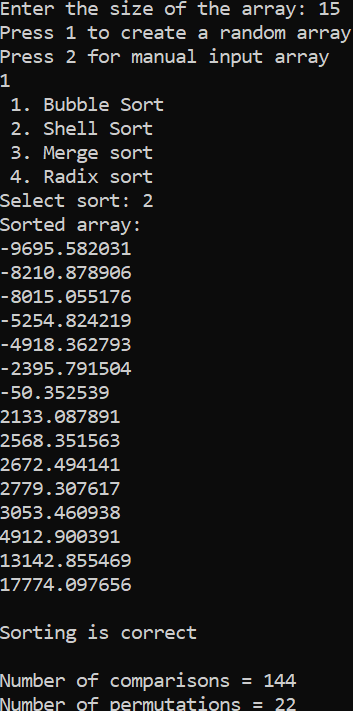
Затем, чтобы выбрать способ заполнения массива, он должен ввести «1» для создания случайного массива или «2» для самостоятельного ввода каждого элемента.



Далее пользователю доступны на выбор 4 сортировки. Номер выбранной сортировки необходимо ввести с помощью клавиатуры.

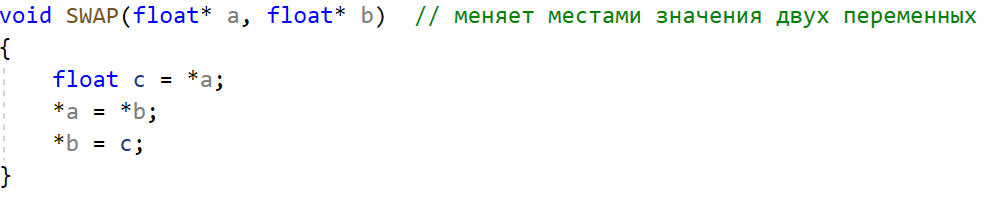


В результате на экран выводится упорядоченный выбранной сортировкой в порядке возрастания массив заданной длины и количество выполненных в процессе сортировки сравнений и перестановок элементов. Надпись «Sorting is correct» означает, что сортировка выполнена успешно. «Something went wrong» выводится в случае какой-либо ошибки, произошедшей в процессе сортировки.

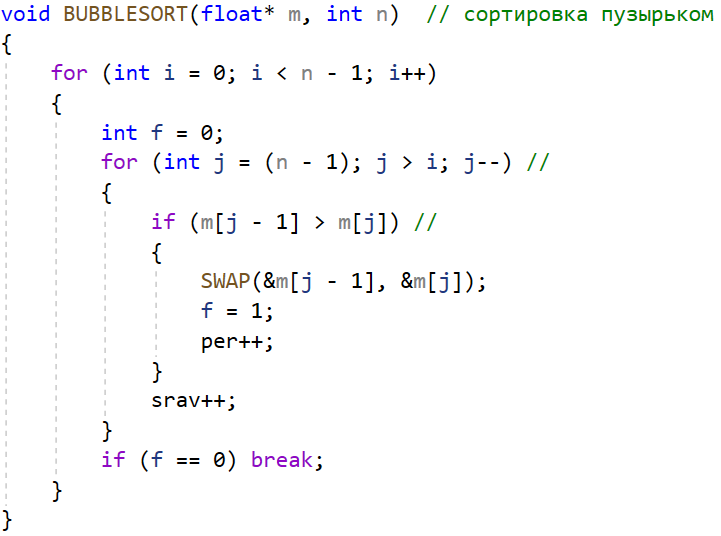


# Описание программной реализации

Функция **SWAP** меняет местами два элемента массива. Указатели позволяют проводить операцию над глобальными переменными. SWAP используется в сортировке пузырьком и сортировке Шелла.

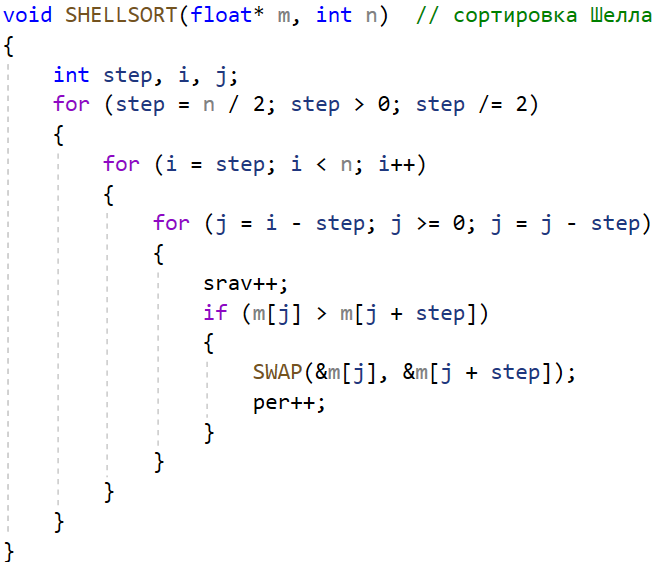


**Сортировка пузырьком** принимает в качестве входных данных массив типа float и его целочисленный размер. Для каждого элемента массива создается цикл, проходящий сверху вниз от конца массива до элемента индексом больше его, в котором происходят сравнения каждых соседних элементов и их перестановка в случае расположения в порядке убывания. При этом в случае вызова функции SWAP инкримируется значение переменной, отвечающей за количество перестановок, количество сравнений увеличивается на единицу при каждом шаге внутреннего цикла. Сортировка пузырьком может быть модифицирована переменной-флагом, которая принимает значение 1, останавливающее функцию, в случае, если перестановок во внешнем цикле не производилось.



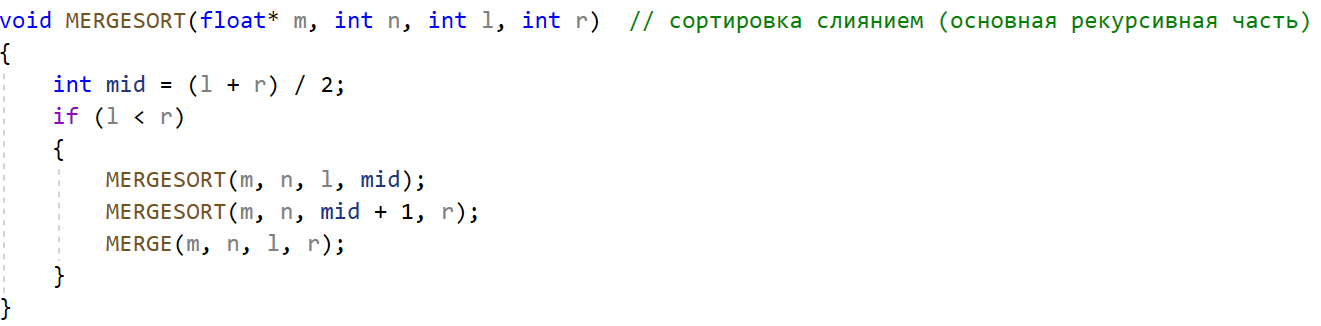
**Сортировка Шелла** принимает в качестве входных данных массив типа float и его целочисленный размер. Создаются три локальные целочисленные переменные - счетчики циклов и шаг сортировки.

Внешний цикл целочисленно уменьшает шаг сортировки вдвое. Для каждого значения шага для каждого элемента от индекса размера шага до конца массива происходит сравнение его значения со значениями элементов, стоящих от него левее на расстояние, кратное шагу, и увеличение счетчика, отвечающего за количество сравнений. В случае неправильного порядка двух элементов для них вызывается функция SWAP и происходит увеличение счетчика, отвечающего за количество перестановок.



**Сортировка слиянием** принимает в качестве входных данных массив типа float, его целочисленный размер, номера первого и последнего элемента. Целочисленным делением находится середина массива. Если левая и правая граница не совпадают (то есть в массиве больше одного элемента), функция MERGESORT рекурсивно вызывается для двух «разбитых» подмассивов (для элементов левее и правее середины исходного массива). При этом переменные l и r принимают другие значения (для первого подмассива – l и mid , для второго – mid+1 и r). Далее функция MERGE производит слияние двух отсортированных подмассивов в один упорядоченный.

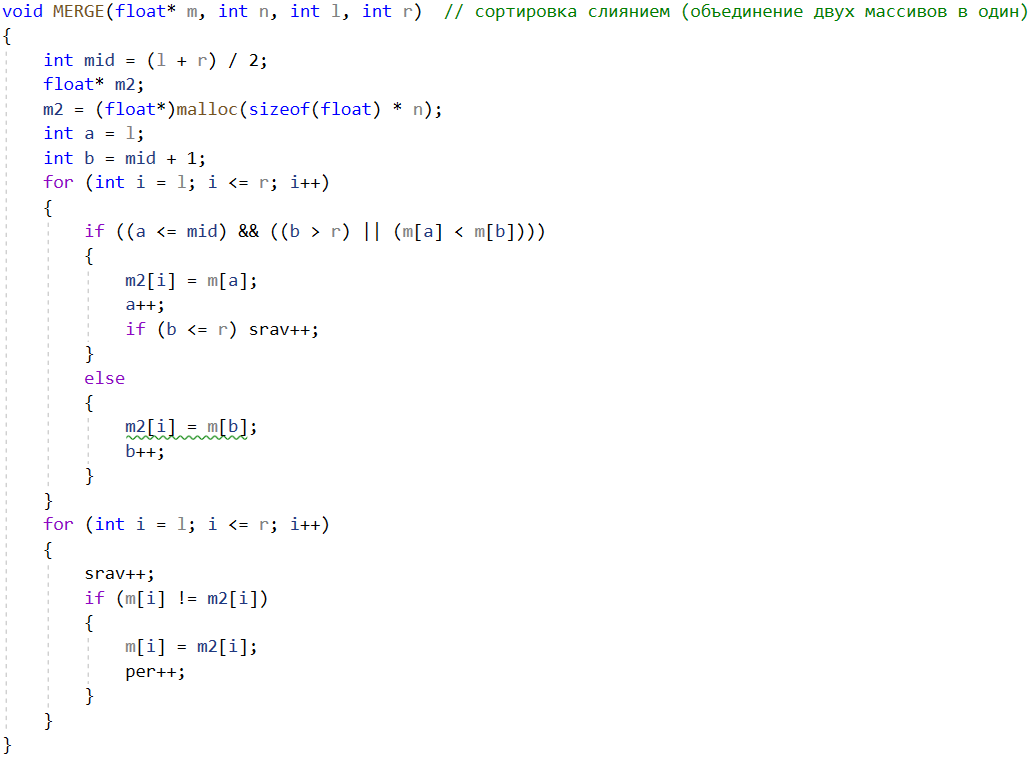
Функция вызывает саму себя до тех пор, пока l не будет больше или равно r, то есть пока рассматриваемый массив содержит в себе больше 1 элемента. Массив из 1 элемента по определению считается упорядоченным.

 Задача функции MERGE – объединять отсортированные подмассивы в один результирующий. В качестве входных данных она принимает массив типа float, его целочисленный размер, номера первого и последнего элемента. Так же целочисленным делением находится индекс середины массива. Затем объявляются указатель на дополнительный массив того же типа данных и размера, что и исходный, и две целочисленные переменные, принимающие значения индексов начал первого и второго отсортированных подмассивов.

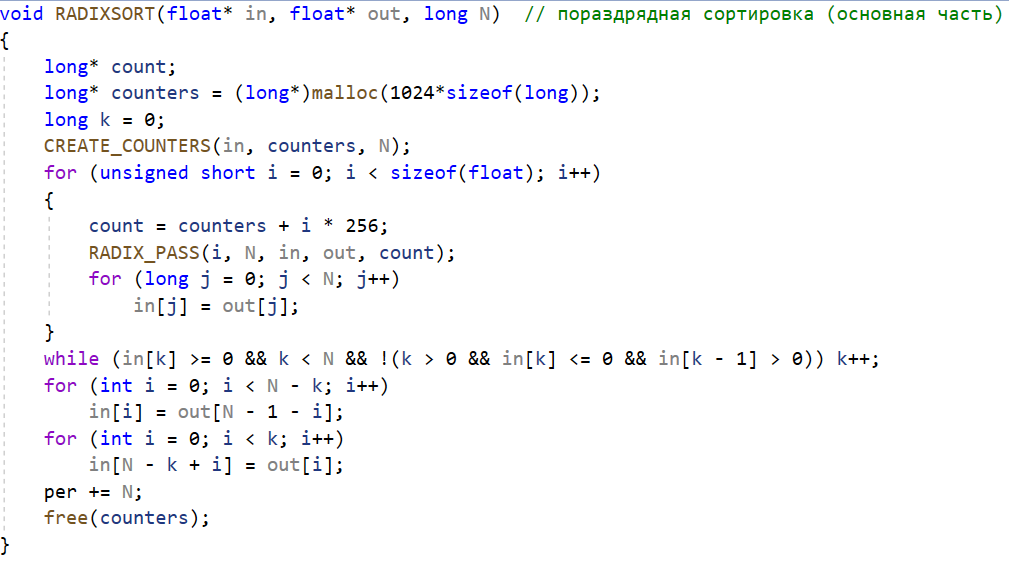
Чтобы заполнить дополнительный массив элементами из двух подмассивов в порядке возрастания необходимо для каждого индекса результирующего массива делать проверку на выход за границы подмассивов и в случае, если в обоих подмассивах «остались» элементы, производить сравнение их первых элементов. В результирующий массив заносится наименьший элемент из сравнимых в случае строгого неравенства или элемент из первого подмассива в случае равенства. Так как элементы из первого подмассива в исходном массиве стояли раньше элементов второго подмассива, выбор элемента именно из первого обеспечивает устойчивость сортировки. При каждом переносе элементов из подмассива в массив увеличивается счетчик его начала.

Сравнения элементов происходят только в случае, если оба подмассива «еще содержат» в себе элементы. В случае если в каком-либо подмассиве «закончатся» элементы, все элементы из другого в том же порядке, что расположены в подмассиве, заносятся в результирующий.

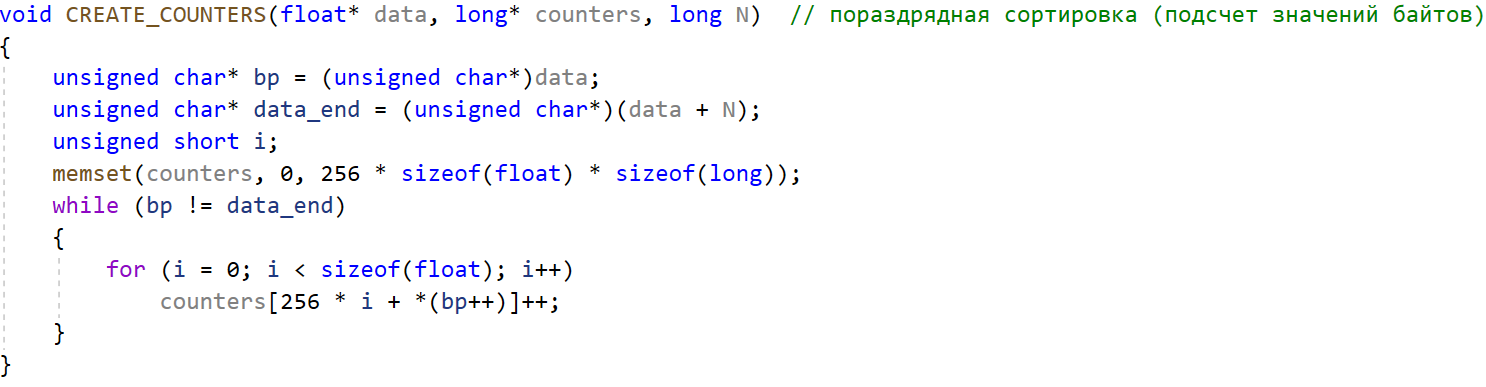
Наконец, чтобы отсортировать требуемый массив, каждый его элемент сравнивается с элементом из дополнительного с тем же индексом. В случае неравенства значений элементов в требуемый массив в рассматриваемую ячейку заносится значение элемента из дополнительного и увеличивается счетчик перестановок. При этом на каждом шаге последнего цикла увеличивается счетчик сравнений.

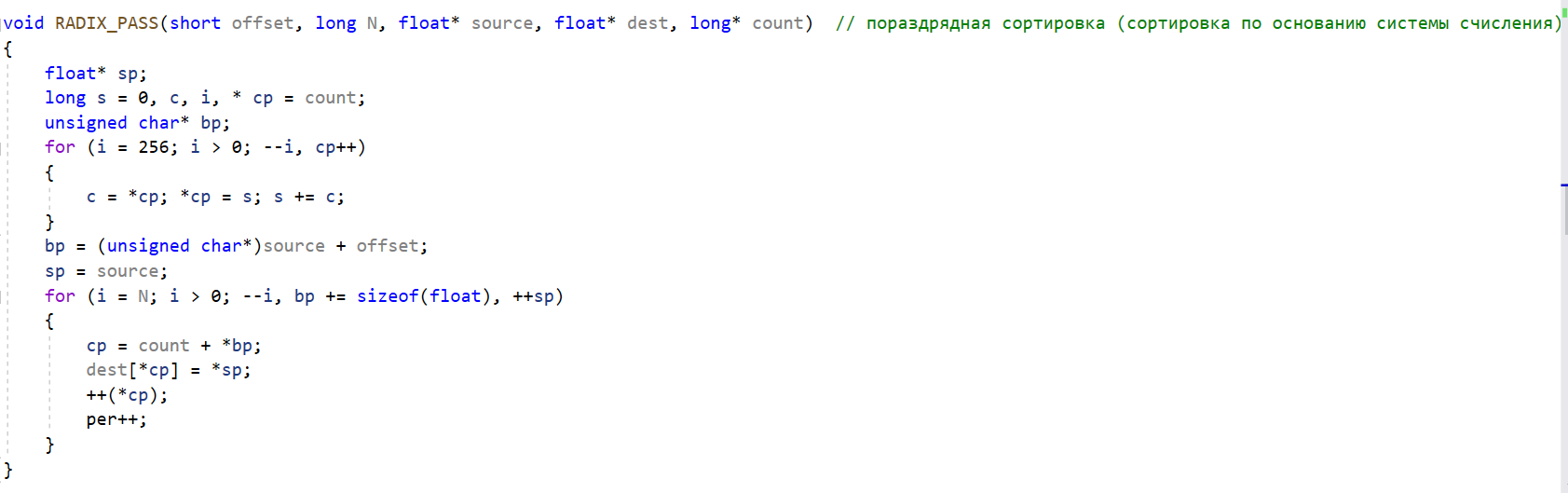


**Поразрядная сортировка** принимает в качестве входных данных основной и дополнительный массивы равных размеров и их целочисленный размер.



Вспомогательный массив размера (1024\*sizeof(long)) заполняется в функции CREATE\_COUNTERS, принимающей в качестве входных данных исходный массив, вспомогательный массив и целочисленный размер исходного массива. Функция записывает во вспомогательный массив элементы из исходного в побитовом виде.

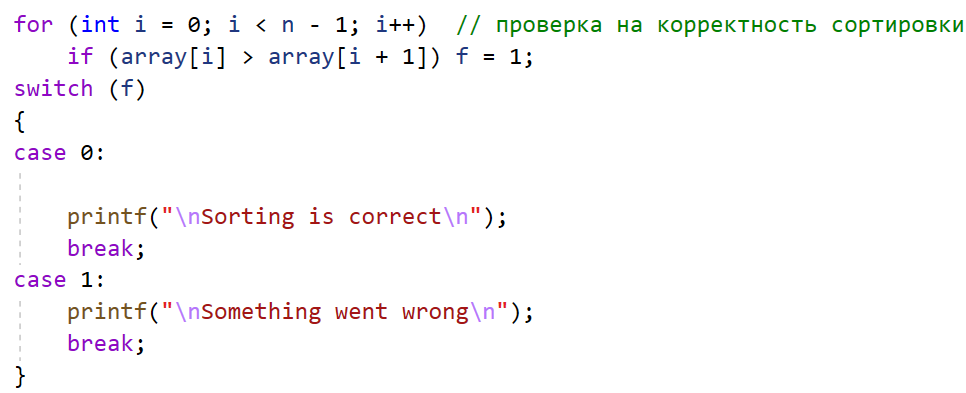


После перезаписи элементов массива вызывается функция RADIX\_PASS для сортировки подсчетом вспомогательного массива. При этом учитывается, что в памяти байты записи числа хранятся в обратном порядке, а так же то, что старший бит отрицательных чисел равен 1, в то время как положительных – 0. Поэтому в массиве будут расположены сначала положительные числа, затем отрицательные в порядке возрастания. 

В функции RADIX\_PASS сортировка проходит по одному байту, результат заносится в дополнительный массив размера исходного, после чего сортировка проходит по следующему байту.

Чтобы сортировка выдавала корректно упорядоченный массив, в конце работы функции находится индекс первого отрицательного числа, с помощью чего все отрицательные числа, сохраняя их порядок, переставляются в начало массива.

# Подтверждение корректности

Для подтверждения корректности в программе используется переменная-флаг f, которая в случае неправильного порядка элементов в результирующем массиве принимает в ходе цикла прохода по массиву значение 1. В цикле сравниваются все соседние элементы и проверяется их расположение в порядке возрастания. Оператор switch-case выводит на экран «Sorting is correct», если флаг равен 0 и сортировка прошла успешно, и «Something went wrong», если флаг равен 1 и в ходе сортировки произошли какие-либо ошибки. 

# Результаты экспериментов

**Сортировка пузырьком.** Лучшим случаем для данной сортировки является уже упорядоченный массив. Худшим – массив, элементы которого расположены в порядке убывания. Средним – случайный массив. Для эксперимента возьмем три массива длины 10: отсортированный, отсортированный в обратном порядке и случайный.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лучший случай | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| Худший случай | 100 | 90 | 80 | 70 | 60 | 50 | 40 | 30 | 20 | 10 |

|  |  |  |
| --- | --- | --- |
|  | Количество сравнений | Количество перестановок |
| Лучший случай | 9 | 0 |
| Худший случай | 45 | 45 |
| Случайный массив | 45 | 25 |

Таким образом, количество сравнений и перестановок зависит от степени упорядоченности исходного массива.

Сложность сортировки O(). Число сравнений и перестановок в худшем случае = . Число сравнений в лучшем случае = N-1, число перестановок = 0.

**Сортировка Шелла.** Лучшим случаем для данной сортировки является уже упорядоченный массив. Худший случай – неправильный выбор шага. Лучшая сложность O(), худшая и средняя зависят от выбранной последовательности шагов. Сортировка деградирует до сложности O() при неудачном выборе данной последовательности.

Примером неудачного выбора последовательности шагов может быть сортировка следующего массива:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 12 | 14 | 16 | 18 | 11 | 13 | 15 | 17 | 19 |

|  |  |  |
| --- | --- | --- |
|  | Количество сравнений | Количество перестановок |
| Лучший случай | 70 | 0 |
| Неудачный случай | 70 | 8 |
| Случайный массив | 70 | 6 |

Количество перестановок зависит от степени упорядоченности массива.

**Сортировка слиянием.** Требует O() дополнительной памяти. Массив длины n делится пополам за O() до тех пор, пока размер части не станет равен 1. В любом случае работает за время O(.

|  |  |  |
| --- | --- | --- |
|  | Количество сравнений | Количество перестановок |
| Лучший случай | 53 | 0 |
| Худший случай | 34 | 34 |
| Случайный массив | 41 | 29 |

**Поразрядная сортировка.** Сложность O( по времени и O() по дополнительной памяти, где k – количество бит, требуемых для хранения каждого ключа. Используется сортировка подсчетом и ее свойство устойчивости.

|  |  |  |
| --- | --- | --- |
|  | Количество сравнений | Количество перестановок |
| Случайный массив 1 длины 10 | 0 | 50 |
| Случайный массив 2 длины 20 | 0 | 100 |
| Случайный массив 3 длины 10000 | 0 | 50000 |

# Заключение

В процессе выполнения лабораторной работы были написаны корректно работающие реализации для четырех сортировок массивов, посчитаны количества сравнений и перестановок в каждом случае, определена сложность алгоритмов. По результатам экспериментов было выявлено, что самыми быстрыми сортировками являются сортировка слиянием (для почти упорядоченного массива) и поразрядная сортировка, сложность которых линейно зависит от размера массива при данных условиях.

# Приложениемаин 1.pngмаин 2 2.png