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**:

студент группы 382003-1

Лапин Дмитрий

**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 случайно генерируемый массив элементов типа double следующими алгоритмами:

* Сортировка выбором;
* 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 выбором.**

В исходном массиве находим минимальный элемент, меняем его местами с первым элементом массива. Уже, из всех элементов массива один элемент стоит на своём месте. Теперь будем рассматривать не отсортированную часть массива, то есть все элементы массива, кроме первого. В неотсортированной части массива опять ищем минимальный элемент. Найденный минимальный элемент меняем местами со вторым элементом массива и т. д. Таким образом, суть алгоритма сортировки выбором сводится к многократному поиску минимального (максимального) элементов в неотсортированной части массива.

Алгоритм имеет сложность O().

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 настоящей реализации это O() в лучшем случае и O() в худшем.

1. **Сортировка слиянием.**

Основная идея заключается в том, что сортируемый массив разбивается на две части примерно одинакового размера. Каждая из получившихся частей сортируется отдельно, например, тем же самым алгоритмом. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы. Два упорядоченных массива половинного размера соединяются в один. Слияние двух подмассивов в третий результирующий происходит так: на каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на один.

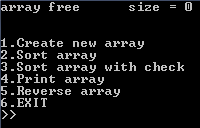
1. **Поразрядная сортировка.**

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

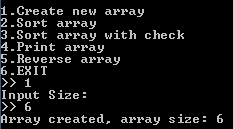
В качестве одного разряда берется 1 байт, так же учитывается, что байты записи числа в памяти хранятся в обратном порядке. В записи отрицательных чисел с плавающей запятой старший бит равен 1, в то время как положительных – 0. Поэтому после побайтовой сортировки необходимо пройти по массиву, переставив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

# Руководство пользователя

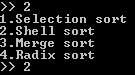
При запуске программы пользователю предлагается меню, состоящее из 6 пунктов.



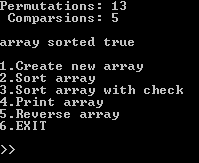
Затем, пользователю нужно создать массив. Для этого ему требуется ввести 1, а затем ввести размер массива.



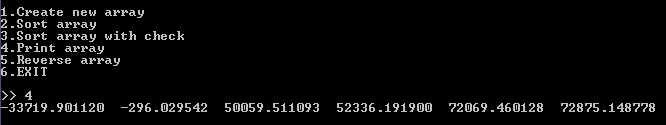
Далее пользователь возвращается в исходное меню, и ему доступны на выбор 3 функции: отсортировать массив, отсортировать его с проверкой, вывести массив. При выборе функции 2 или 3, ему программа предоставит выбор из 4х сортировок:



В результате на экран выводится количество выполненных в процессе сортировки сравнений и перестановок элементов и результат проверки массива на отсортированность, если пользователь выбрал функцию 3.



После этого пользователь может вывести массив с помощью функции 4



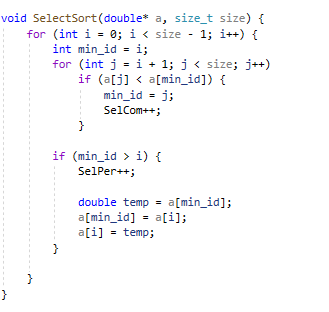
Или развернуть массив, используя функцию 5



Далее пользователь может продолжить работать в программе, используя меню, создать новый массив, попробовать другие виды сортировок, или выйти из нее с помощью функции 6 “EXIT”.

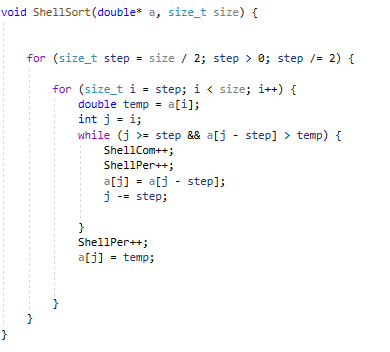
# Описание программной реализации

**Сортировка выбором** принимает в качестве входных данных массив типа double и его целочисленный размер. Для каждого элемента массива кроме последнего, начиная с первого, создается цикл, проходящий снизу вверх от элемента, следующего за текущим элементом до конца массива, в котором происходит поиск минимума в соответствующей части массива. Таким образом, после каждого прохода цикла один из элементов массива оказывается на соответствующем ему месте, в отсортированном массиве. В соответствующих местах сортировки инкрементируются счетчики, отвечающие за подсчет числа перестановок и сравнений.

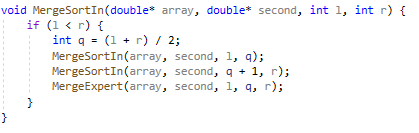


**Сортировка Шелла** принимает в качестве входных данных массив типа double и его целочисленный размер. Создаются три локальные целочисленные переменные - счетчики циклов и шаг сортировки.

Внешний цикл целочисленно уменьшает шаг сортировки вдвое. Для каждого значения шага для каждого элемента от индекса размера шага до конца массива и элементов, отстоящих от него на расстояние, кратное шагу совершается алгоритм сортировки вставками для него и элементов, отстоящих от него на расстояние, равное шагу, для элементов сравнение его значения со значениями элементов, стоящих от него левее на расстояние, кратное шагу. В соответствующих местах сортировки инкрементируются счетчики, отвечающие за подсчет числа перестановок и сравнений. Функции SelectSort и ShellSort определяются и объявляются в файлах SelectShellSort.h и SelectShellSort.cpp



**Сортировка слиянием** принимает в качестве входных данных массив типа double, 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 MergeSortIn 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 MergeExpert 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.

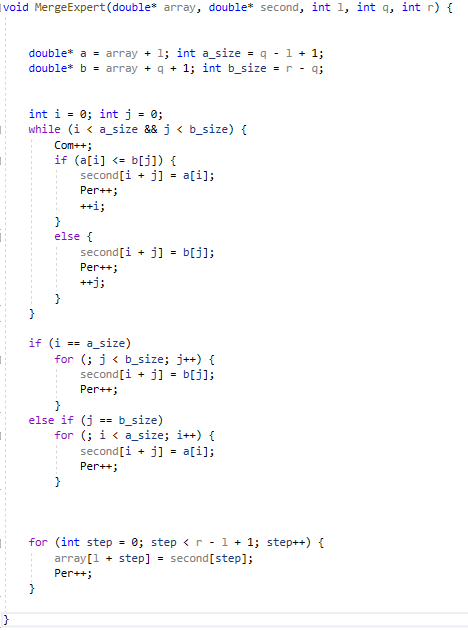
Функция MergeExpert – объединяет отсортированные подмассивы в один результирующий. В качестве входных данных она принимает массив типа double, массив дополнительной памяти размером равным исходному, его целочисленный размер, номера первого и последнего элемента, индекс середины массива. Затем объявляются два указателя на начало массива и его часть, которая начинается с его середины. Таким образом мы получаем указатели на два “подмассива”, и вычисляем через индексы их размеры, которая образована массивом 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.

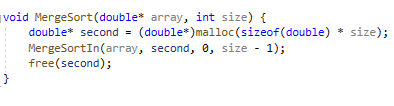
Сравнения элементов происходят только в случае, если оба подмассива «еще содержат» в себе элементы. В случае если в каком-либо подмассиве «закончатся» элементы, все элементы из другого в том же порядке, что расположены в подмассиве, заносятся в результирующий.

Наконец, мы копируем результирующую последовательность элементов из массива дополнительной памяти в исходный массив

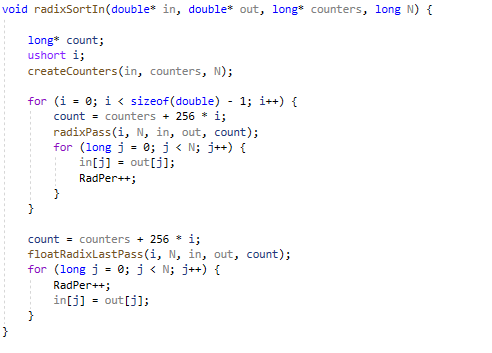
В соответствующих местах сортировки инкрементируются счетчики, отвечающие за подсчет числа перестановок и сравнений. Функции этой сортировки определяются и объявляются в файлах mergeSort.h и mergeSort.cpp.



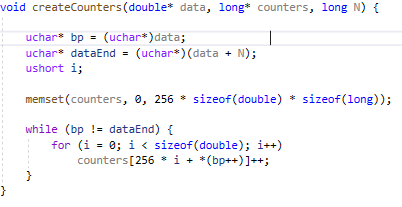
Так же мы используем функцию MergeSort, которая создает “обертку” функции MergeSort. Она принимает исходный массив и его размер, выделяет и освобождает дополнительную память и запускает функцию MergeSortIn.



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

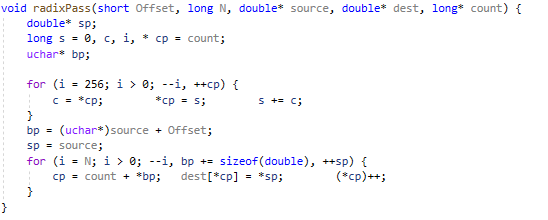


Вспомогательный массив размера (256 \* sizeof(double) \* sizeof(long)) заполняется в функции createCounters(), принимающей в качестве входных данных исходный массив, вспомогательный массив и целочисленный размер исходного массива. Функция записывает во вспомогательный массив элементы из исходного в побитовом виде.

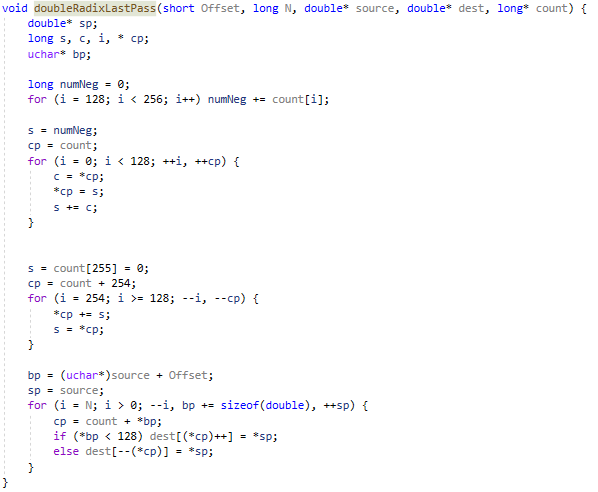


После перезаписи элементов массива вызывается функция radixPass для сортировки подсчетом вспомогательного массива. При этом учитывается, что в памяти байты записи числа хранятся в обратном порядке, а так же то, что старший бит отрицательных чисел равен 1, в то время как положительных – 0. Поэтому в массиве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

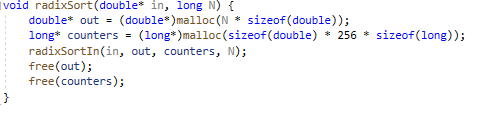
В функции radixPass сортировка проходит по одному байту, результат заносится в дополнительный массив размера исходного, после чего сортировка проходит по следующему байту.



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. Это реализует функция doubleRadixPass().

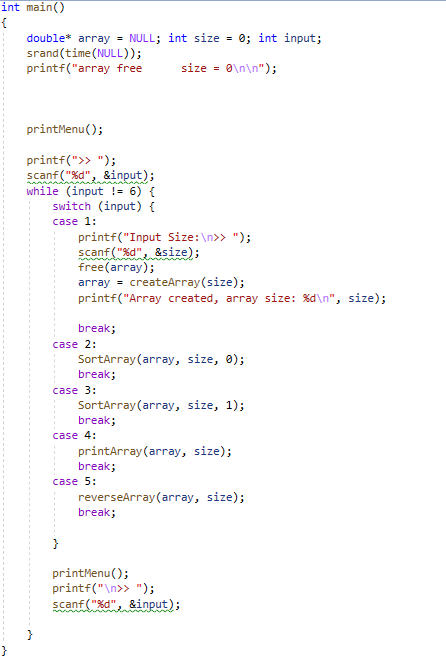


Так же реализована функция-обёртка RadixSort, которая принимает на вход исходный массив, его целочисленный размер, выделяет и освобождает дополнительные массивы



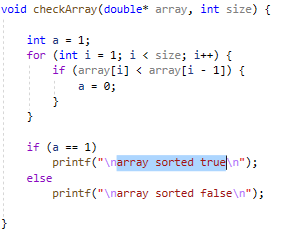
В соответствующих местах сортировки инкрементируются счетчики, отвечающие за подсчет числа перестановок и сравнений. Функции этой сортировки определяются и объявляются в файлах radixSort.h и radixSort.cpp.

В функции main реализуется часть программы, отвечающая за взаимодействие спользователем. Там объявляются переменные указателя на массив и его размера. Далее запускается цикл, который реализует работу пользователя с меню, через оператор switch и работает до того как пользователь ввел цифру 6, которая соответствует пункту EXIT.



# Подтверждение корректности

Для подтверждения корректности в программе используется переменная-флаг a, 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 if else выводит на экран «array sorted true», если флаг равен 0 и сортировка прошла успешно, и «array sorted false», если флаг равен 1 и в ходе сортировки произошли какие-либо ошибки.



# Результаты экспериментов

**Сортировка выбором.** Лучшим случаем для данной сортировки является уже упорядоченный массив. 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 | 45 | 0 |
| Худший случай | 45 | 7 |
| Случайный массив | 45 | 6 |

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.

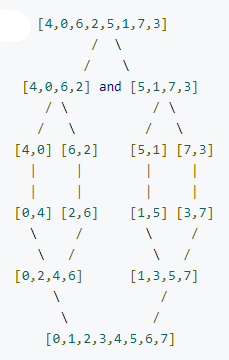
**Сортировка Шелла.** Лучшим случаем для данной сортировки является уже упорядоченный массив. Худший случай – неправильный выбор шага. Лучшая сложность O(), худшая и средняя зависят от выбранной последовательности шагов. Сортировка деградирует до сложности O() при неудачном выборе данной последовательности.

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 | 0 | 22 |
| Неудачный случай | 36 | 14 |
| Случайный массив | 30 | 8 |

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(. Лучшим случаем будет уже отсортированный массив. В качестве примера худшего случая я взял массив [4,0,6,2,5,1,7,3] 

|  |  |  |
| --- | --- | --- |
|  | Количество сравнений | Количество перестановок |
| Лучший случай | 12 | 48 |
| Худший случай | 17 | 48 |
| Случайный массив | 15 | 48 |

**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 | 80 |
| Случайный массив 2 длины 20 | 0 | 160 |
| Случайный массив 3 длины 10000 | 0 | 80000 |

# 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.

# Приложение

