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»**

**Выполнил**:

студент группы 3822Б1ПМ1

Смирнов И. К.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Содержание**

[Постановка задачи](#_Toc26962562) 3

[Метод решения 4](#_Toc26962563)

[Руководство пользователя](#_Toc26962564) 7

[Описание программной реализации](#_Toc26962565) 8

[Подтверждение корректности](#_Toc26962566) 10

[Результаты экспериментов](#_Toc26962567) 11

[Заключение](#_Toc26962568) 13

[Приложение 1](#_Toc26962569)4

# Постановка задачи

В данной лабораторной работе основной задачей является реализация известных алгоритмов сортировки массивов данных. Также задачей представляется проанализировать реализованные подпрограммы: проверить код на корректность работы и провести эксперимент по определению времени, затрачиваемого на работу алгоритмов. Ещё необходимо осуществить взаимодействие пользователя с программой.

# Метод решения

**Сортировка вставками.**

Условно массив разделяется на две части — отсортированную и неотсортированную. Далее из неотсортированной части берется один элемент и помещается в отсортированную часть «на свое место», то есть на то место, где он должен находиться, не нарушая отсортированности первой части. Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним. В зависимости от результата сравнения элемент либо остается на текущем месте (вставка завершена), либо они меняются местами и процесс повторяется. Таким образом, выбранный элемент вставляется куда нужно, в результате чего отсортированная часть массива увеличивается, а неотсортированная уменьшается.

Общее время работы алгоритма оценивается сверху, как O(n2) (где n – размер входных данных), так как количество сравнений на каждой итерации увеличивается на 1, то есть всего сравнений 1+2+3+...+(n-1)+n = (n(n-1))/2.

**Сортировка выбором.**

В данном алгоритме, как и в сортировке вставками, массив делится на две части — отсортированную и неотсортированную. Далее из неотсортированной части выбирается наименьший элемент и вставляется как последний элемент отсортированной части, то есть меняется местами с первым элементом неотсортированного фрагмента массива, тем самым постепенно увеличивается отсортированная часть массива.

Общее время работы алгоритма оценивается сверху, как O(n2) (где n – размер входных данных), так как количество сравнений на каждой итерации для выбора минимального элемента уменьшается на 1, то есть всего сравнений n+(n-1)+...+2+1 = (n(n-1))/2.

**Сортировка Хоара** (быстрая сортировка).

Метод этой сортировки основывается на том, чтобы на первой итерации расположить все элементы относительно одного (опорного) так, чтобы слева от опорного элемента оказались числа меньшие его, а справа — равные или большие его. На следующей итерации приведенная последовательность действий производится над получившимися частями массива, то есть над фрагментом, где числа меньше опорного, и отдельно, где числа больше или равны опорному элементу. Вне зависимости от того, какой элемент выбран в качестве опорного, массив будет отсортирован, но все же наиболее удачным считается ситуация, когда по обеим сторонам от опорного элемента оказывается примерно равное количество элементов.

Оценка времени. Если опорный элемент выбран так, что слева и справа от него оказалось примерно одинаковое число элементов, то количество операций примерно равно: n + n/2 +n/2 + n/4 + n/4 +n/4 +n/4 +… = n +2\*(n/2) +4\*(n/4)+… В сумме порядка log2(n) слагаемых, так как массив каждый раз делится на две части (то есть n делится на 2) . Среднее время работы O(n\*log(n)).

**Сортировка слиянием.**

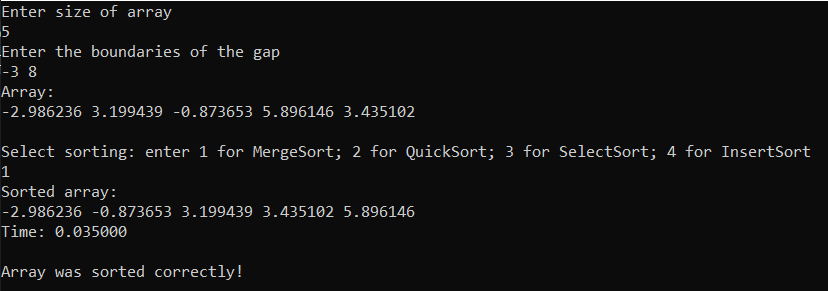
Данный алгоритм основывается на том, чтобы исходный массив чисел делить пополам до тех пор, пока не останутся массивы, состоящие из одного элемента. Далее к каждой паре одноэлементных массивов применяется процедура слияния: для этого на каждом шаге сравниваются первые два элемента выбранных массивов, наименьший элемент записывается во вспомогательный массив, и в том подмассиве, откуда записали наименьший элемент, переходим к следующему элементу. Когда один из массивов закончился, оставшиеся элементы второго массива добавляются во вспомогательный массив. Процедура слияния применяется ко всем парам одноэлементных массивов, пока те не закончатся, далее слияние применяется к парам двухэлементных массивов, затем к парам четырехэлементных массивов и так далее, пока не сольются половины исходного массива, образовав тем самым отсортированный вспомогательный массив. В завершении, элементы из вспомогательного массива копируются в исходный.

Оценка времени. Сначала во вспомогательный массив сливаются все пары одноэлементных массивов (всего операций 1+1+...+1 = n), затем пары двухэлементных массивов (всего операций 2+2+...+2 [всего n/2 слагаемых] = n) и так далее. Общая сумма операций n+n+...+n, всего слагаемых log2(n), так как массив каждый раз делится пополам. Среднее время работы O(n\*log(n)).

# Руководство пользователя

При использовании программы пользователь должен ввести число — количество элементов в сортируемом массиве. Далее необходимо ввести пару чисел — границы промежутка, из которого будут выбраны числа для автоматической генерации массива (числа в массиве случайные). Также пользователь выбирает какой именно алгоритм сортировки нужно применить к данному массиву: сортировка слиянием, быстрая сортировка, сортировка выбором или вставкой.

В результате работы на консоль выводится отсортированный массив, время, за которое был отсортирован массив с помощью выбранной сортировки, а также пользователь видит сообщение о корректности сортировки сгенерированного массива — в программе реализовано сравнение полученного массива с результатом сортировки исходного массива с помощью встроенного алгоритма “quicksort” .

Рис. 1. Пример полученного пользователем результата работы программы.

# Описание программной реализации

Программа состоит из одного файла (*main.cpp*), в котором реализованы все алгоритмы сортировок и вспомогательные функции.

Функция сортировки выбором — **void selectSort(float\* arr, int n)** — принимает на вход массив и значение количества элементов в нем. Основой функции являются два вложенных цикла: внешний цикл проходит по всем элементам массива и впоследствии добавляет в конец отсортированной части наименьший элемент, а внутренний цикл ищет в неотсортированном фрагменте массива наименьший элемент.

Функция сортировки вставками — **insertSort(float\* arr, int n)** — принимает на вход массив и значение количества элементов в нем. Основой функции, так же как и в сортировке выбором, являются два вложенных цикла: внешний фиксирует первый элемент в неотсортированной части массива, а внутренний цикл вставляет этот элемент на нужное место в отсортированной части. Вставка происходит путем сравнения элементов: если выбранное число оказалось меньше другого в отсортированной части, то эти элементы меняются местами; иначе если выбранный элемент больше всех чисел отсортированного фрагмента массива, то он остается на своем месте.

Функция быстрой сортировки — **void quickSort(float\* arr, int l, int r)** — принимает массив и его правую и левую границы, а именно: номера первого и последнего элементов массива. Подпрограмма quickSort реализована рекурсивно. С помощью вспомогательной функции **part()** функция быстрой сортировки меняет местами элементы относительно опорного элемента так, чтобы слева от опорного элемента оказались числа меньшие его, а справа — равные или большие его. Сама вспомогательная функция **int part(float\* arr, int l, int r)** принимает так же на вход массив и его правые и левые границы, а возвращает значение *p*— индекс элемента, слева от которого элементы меньшие опорного. Дальше функция быстрой сортировки запускает саму себя от значения левой границы *l* и правой границы *p*, а также от левой границы *(p+1)* правой — *r*, пока *l* не станет равной *r*.

Функция сортировки слиянием — **void mergeSort(float\* in, float\* out, int start, int end)** — принимает на вход исходный сортируемый массив, вспомогательный пустой массив, а также границы сортировки исходного массива, а именно: индекс элемента, с которого нужно начать сортировку (включительно), и индекс на котором нужно ее закончить (невключительно). Подпрограмма mergeSort реализована рекурсивно. Внутри самой функции считается индекс *m* центрального элемента в массиве, а затем функция вызывает саму себя от параметров *start* и *m*, а также от *m* и *end.* Так функция делит исходный массив на каждом шаге на две части, пока эти подмассивы не будут иметь длину в один элемент. После вспомогательная функция **merge()** сливает подмассивы во вспомогательный массив *out* за счет чего и происходит сортировка. Функция **void merge(float\* in, float\* out, int start1, int start2, int end)** принимает исходный и вспомогательный массивы, переданные в *mergeSort*, а также границы подмассивов: *start1* — элемент с которого начнется слияние первого подмассива (включительно), *start2* — элемент, на котором закончится сортировка первого подмассива (невключительно) и начнется сортировка второго подмассива (включительно), *end* — элемент, на котором закончится сортировка второго подмассива (невключительно).

# Подтверждение корректности

Для подтверждения корректности в программе используется встроенная функция void qsort(void \*base, size\_t nitems, size\_t size, int (\*compar) (const void \*, const void\*)), которая принимает сортируемый массив, количество элементов в нем, размер каждого элемента и вспомогательную функцию сравнения.

В программе создается копия исходного массива, который сортируется с помощью функции *qsort,* затем массив-результат поэлементно сравнивается с массивом, отсортированным одним из реализованных алгоритмов сортировки. Сообщение о корректности результата выводится пользователю в консоли.

# Результаты экспериментов

Эксперимент состоит в том, чтобы сравнить время работы сортировок с теоретическими расчетами. Для этого в программе производится измерение времени с помощью встроенной функции *clock().* Результаты измерений занесены в таблицу (Рис. 2): в первом столбце написано количество элементов *n* в массиве (все числа в промежутке [-1000; 1000]), в строках — время работы программы в секундах.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Выбором | Вставкой | Слиянием | Быстрая |
| 250 | 0,001 | 0,002 | 0,000001 | 0,000001 |
| 1000 | 0,026 | 0,025 | 0,000001 | 0,001 |
| 5000 | 0,595 | 0,686 | 0,004 | 0,001 |
| 9000 | 2,159 | 2,26 | 0,009 | 0,002 |
| 15000 | 5,682 | 6,299 | 0,016 | 0,003 |
| 25000 | 13,47 | 14,97 | 0,027 | 0,006 |

Рис. 2. Результаты измерения времени.

Чтобы сравнить время работы с теоретическим разделим полученное время на теоретическое, то есть время работы делится на *n2* для пузырьковых сортировок (выбором и вставкой) и на *log2(n)\*n* для быстрой сортировки и сортировки слиянием. Результаты вычислений представлены на диаграмме (Рис. 3) (все результаты получены в 10-8 у.е.).

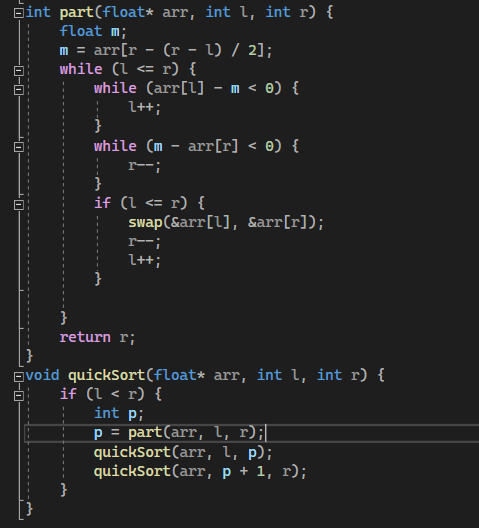
Из диаграммы видно, что каждый из графиков стремится к какой-то собственной константе, следовательно, теоретическая оценка времени работы алгоритмов была верна.

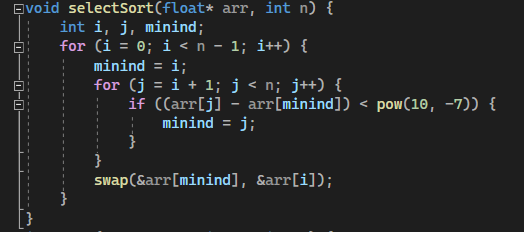
Рис. 3. Сравнение теоретических расчетов времени с действительными.

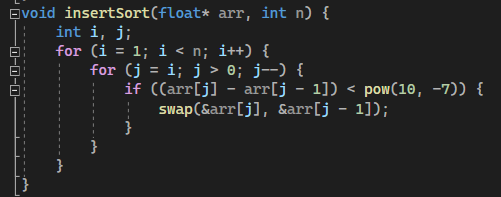
# Заключение

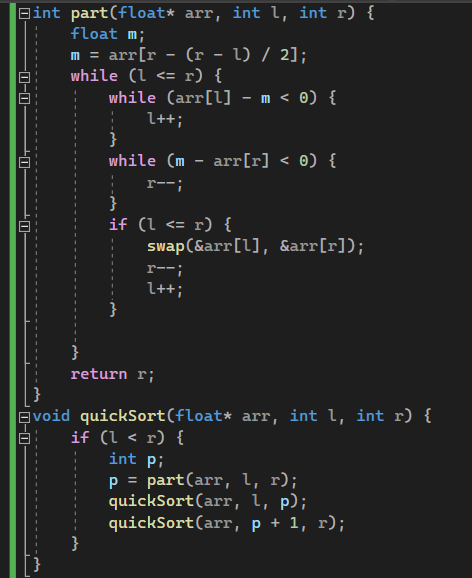
В итоге, поставленная задача была выполнена: успешно реализованы алгоритмы сортировки, описаны взаимодействие пользователя с программой, структура исполняемой программы, проверка на корректность результатов работы алгоритмов, проведен эксперимент по измерению времени работы подпрограмм, результаты которого совпадают с теоретическими вычислениями.

# Приложение

Рис. 5. Быстрая сортировка.

Рис. 4. Сортировка выбором.

Рис. 6. Сортировка вставками

Рис. 5. Быстрая сортировка.

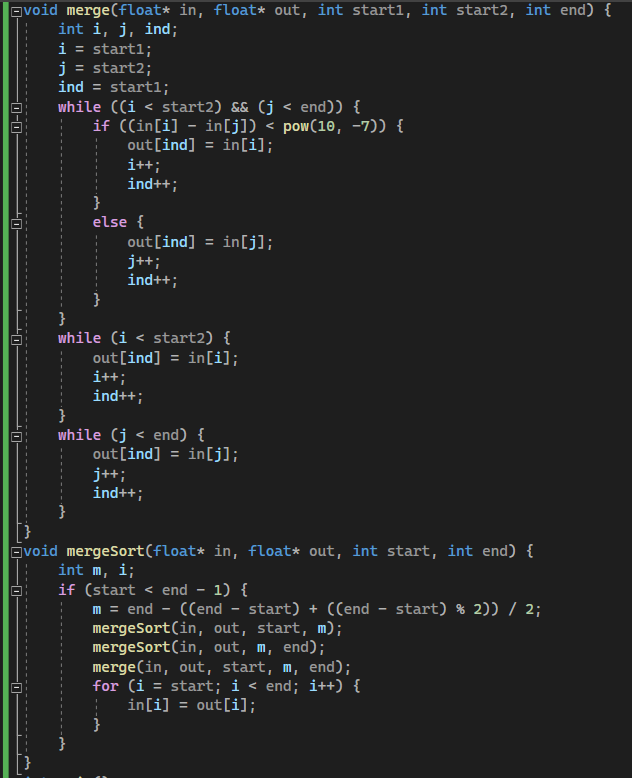


Рис. 7. Сортировка слиянием.