МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Южно-Уральский государственный университет» (национальный исследовательский университет)

Отчёт к итоговой работе

по дисциплине

«Программирование на языках высокого уровня»

Работу выполнил

Студент группы ПИШ-168 _____ Шульц А.В.
подпись, дата

Работу проверил _____ ФИО
подпись, дата

Москва 2024

СОДЕРЖАНИЕ

ПОСТАНОВКА ЗАДАЧИ

Написать программу, которая сортирует массивы данных различными алгоритмами сортировки (быстрая, пузырьковая, вставками и т.д.)

1 ОБЩАЯ ИДЕЯ РЕШЕНИЯ ЗАДАЧИ

Я написала программу с 6 следующими алгоритмами сортировки: пузырьковая, вставками, быстрая, выбором, слиянием, пирамидальная (она же кучей). Изначально пользователю предлагается меню с выбором из 8 вариантов, применить один из алгоритмов сортировки, применить все, а также выход из программы. При выборе любого варианта, кроме выхода, пользователю предлагается ввести размер массива, после удачного ввода есть два варианта, либо заполнить массив вручную, либо случайными числами (у меня диапазон от -10000 до 10000). После опять же успешного ввода, выполняется сортировка массива, вывод его изначального вида и отсортированного, также выводится время, затраченное на сортировку в миллисекундах, с точностью до трех знаков после запятой. Для добавления некоторой логики самой программе как раз таки и был добавлен 7 пункт в меню, за счет которого выполняются все алгоритмы сортировки, после чего выводится время работы каждого в миллисекундах, это может быть полезно для сравнения методов по времени выполнения и соответственно вынесения некоторых выводов на счет скорости их работы на массиве той или иной длины.

2 СТРУКТУРА ПРОГРАММЫ

2.1 Функция main

В главной функции реализовано использование функций сортировки через вывод меню с вариантами в консоль. Так же реализована проверка неудачных сценариев при вводе недопустимых символов или же цифр, которые не соответствуют вариантам выбора в меню.

2.2 Функция getIntInput(const string& prompt, int min, int max)

Данная функция является вспомогательной для проверки корректности ввода пользователем. Применяется в основном при выборе одного из сценариев в меню, на вход подается сообщение, которое будет выведено для понимания пользователем, что от него

требуется ввод, минимальное значение и максимальное, таким образом задается диапазон, в котором должно находиться число.

1. *Создаем переменную типа* int.
2. *Далее запускаем бесконечный цикл, который будет работать до тех пор, пока пользователь не введет правильную цифру.*
3. *Считываем ввод и пытаемся преобразовать его в число типа* int.
4. *Если число не находится в заданном диапазоне, то уточняем диапазон.*
5. *Если ввод пользователя вообще не является цифрой или числом типа* int, *то выводим сообщение о том, что введенные данные неверны.*
6. *Результат работы функции – число типа* int *в нужном диапазоне.*

2.3 Функция getDoubleInput(const string& prompt)

Данная функция является вспомогательной для проверки корректности ввода пользователем. Применяется в основном при заполнении массива вручную на вход подается сообщение, которое будет выведено для понимания пользователем, что от него требуется ввод.

1. *Создаем переменную типа* double.
2. *Далее запускаем бесконечный цикл, который будет работать до тех пор, пока пользователь не введет число типа* double.
3. *Считываем ввод и пытаемся преобразовать его в число типа* double.
4. *Если ввод пользователя вообще не является цифрой или числом, то выводим сообщение о том, что введенные данные неверны.*
5. *Результат работы функции – число типа* double.

2.4 Функция initializeArray(vector<double>& arr)

Данная функция является вспомогательной для заполнения и создания массива. Принимает на вход указатель на динамический массив, ничего не возвращает.

1. *Сначала просим пользователя ввести размер массива, разумеется, с проверкой на корректность ввода.*
2. *Пользователю предоставляется выбор из двух вариантов, заполнить массив самому или же случайными числами.*
3. *После успешного выбора сценария задаем массиву указанную длину и преступаем к заполнению (либо сразу случайными числами, либо же при помощи ввода пользователем с клавиатуры)*

2.5 Функция printArray(const vector<double>& arr)

Вспомогательная функция для вывода массива, применяется для вывода исходного массива и отсортированного. На вход получает указатель на динамический массив, который нужно будет выводить, после чего происходит вывод массива.

2.6 Функция insertionSort(vector<double>& arr)

Принимает на вход указатель на динамический массив, ничего не возвращает и сортирует непосредственно переданный массив. Реализация метода сортировки выглядит так:

1. *Определяем длину массива и сохраняем ее в переменную n.*
2. *Запоминаем текущий элемент*
3. *Далее в цикле, пока элементы левее больше текущего сдвигаем их правее*
4. *Если больше не осталось элементов слева больше или же они кончились, то присваиваем значение нашего начального элемента на его место.*

2.7 Функция partition(vector<double>& arr, int low, int high)

Вспомогательная функция быстрой сортировки. На вход подается указатель на динамический массив, а также начало и конец массива, грубо говоря его длина, но ограниченная индексом первого элемента и последнего, так мы можем обозначить текущую область массива, которая обрабатывается. Функция возвращает позицию опорного элемента. Реализация алгоритма выглядит так:

1. *Создаем опорный элемент, в данном случае он находится всегда в конце массива*
2. *Задаем индекс меньшего элемента*
3. *Далее в цикле, проверяем меньше ли текущий элемент опорного и если да, то меняем их местами. Таким образом опорный элемент окажется посередине, меньше него элементы слева, а больше него справа.*

2.8 Функция quickSort(vector<double>& arr, int low, int high)

Рекурсивная функция, которая делит массив на подмассивы и сортирует их. Сначала при помощи вызова partition() находит индекс опорного элемента, затем рекурсивно вызывается для элементов слева от него и справа от него.

2.9 Функция bubbleSort(vector<double>& arr)

Принимает на вход указатель на динамический массив, ничего не возвращает и сортирует непосредственно переданный массив. Реализация метода сортировки выглядит так:

1. *Определяем длину массива и сохраняем ее в переменную n.*
2. *Сравниваем текущий элемент массива со следующим, если текущий элемент больше следующего, то меняем их местами. Далее с помощью цикла переходим к следующему элементу, который становится текущим и сравниваем его со следующим.*
3. *После выполнения второго цикла мы при помощи первого цикла уменьшаем длину массива на один, поскольку в конце массива у нас уже хранится самый большой элемент из возможных.*
4. *Функция выполняется до тех пор, пока не отсортирует весь массив, после каждого выполнения второго цикла наибольший элемент ставится в конец массива.*

2.10 Функция selectionSort(vector<double>& arr)

Принимает на вход указатель на динамический массив, ничего не возвращает. Реализация метода выглядит так:

1. *Записываем размер массива в переменную n*
2. *Далее начинаем с начала массива и предполагаем, что первый же элемент есть минимальный*
3. *Сравниваем текущий элемент с каждым последующим и определяем настоящий минимальный элемент массива, после чего ставим его в то место, с которого начали проход.*
4. *Далее начинаем уже не с первого, а со второго, потом с третьего и т.д. до тех пор, пока весь массив не будет отсортирован.*

2.11 Функция mergeSort(vector<double>& arr, int left, int right)

Рекурсивная функция, принимает на вход указатель на динамический массив, а также диапазон, в котором будет производиться сортировка. Находит середину массива, далее вызывает саму себя для левой и правой частей, после рекурсивной сортировки двух частей вызывается merge для их объединения.

2.12 Функция merge(vector<double>& arr, int left, int mid, int right)

Принимает на вход указатель на динамический массив, а также диапазоны подмассивов (первый от left до mid, а второй от mid до right). Принцип работы:

1. *Левый подмассив копирует элементы исходного массива с индексами от left до mid, а правый подмассив с индексами от mid+1 до right*
2. *Сравниваются элементы из массивов L и R, меньший элемент записывается в текущую позицию основного массива.*
3. *Если в L или R закончились элементы, то оставшиеся элементы просто копируются в исходный массив.*

2.13 heapify(vector<double>& arr, int n, int i)

Вспомогательная функция, принимает на вход указатель на динамический массив, размер текущей кучи и индекс узла, который нужно упорядочить. Превращает часть массива в кучу, начиная с указанного узла i. Работает так:

1. *Определяем левого и правого потомка при помощи формул*
2. *Сравниваем текущий узел и его левого и правого потомка, если потомок больше текущего узла, то он становится новым корнем поддерева*
3. *Если текущий наибольший элемент не является узлом, то они меняются местами, после чего функция вызывается рекурсивно для затронутого поддерева. В результате узел i и его поддерево становятся кучей.*

2.14 heapSort(vector<double>& arr)

Рекурсивная функция, принимает на вход указатель на динамический массив, работает так:

1. *Строим кучу, итерируем с последнего узла, у которого есть потомки вверх к корню, после чего вызываем heapify, чтобы каждый узел и его потомки соответствовали свойствам кучи.*
2. *Далее на этапе сортировки на каждом шаге перемещаем корень кучи в конец массива (он же набольший элемент), а сам размер уменьшаем на один (поскольку последней элемент уже самый большой) и вызываем heapify для восстановления свойства кучи.*