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**

**«Виды сортировок»**

**Выполнил:**

студентка группы 382003-1

Макарова Н. И.

**Проверил:**

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2019

**Содержание**

Постановка задачи................................................................................................3

Метод решения.....................................................................................................4

Руководство пользователя....................................................................................5

Описание программной реализации.....................................................................6

Подтверждение корректности..............................................................................7

Результаты экспериментов....................................................................................8

Заключение..........................................................................................................11

Приложение........................................................................................................12**Постановка задачи**

1) Необходимо было написать такую программу, чтобы были реализованы сортировки, представленные ниже, для типа данных double:

* Сортировка вставками
* Быстрая сортировка
* Сортировка слиянием
* Поразрядная сортировка

2) Вычислить количество сравнений и перестановок во время реализации сортировки.

**Метод решения**

Описание алгоритмов сортировок:

1) **Сортировка вставками** состоит из вложенного цикла. Оба цикла, и внешний, и внутренний, начинаются со второго элемента. Во внутреннем цикле работа происходит следующим образом: второй элемент сравнивается с первым, если второй меньше, чем первый, то они меняются местами при условии, что упорядочивание происходит по возрастанию. Затем внутренний и внешний циклы принимают значение третьего элемента, а во внутреннем также происходит сравнение элемента с предыдущим. Алгоритм повторяется вплоть до того, пока элементы в массиве не кончатся.

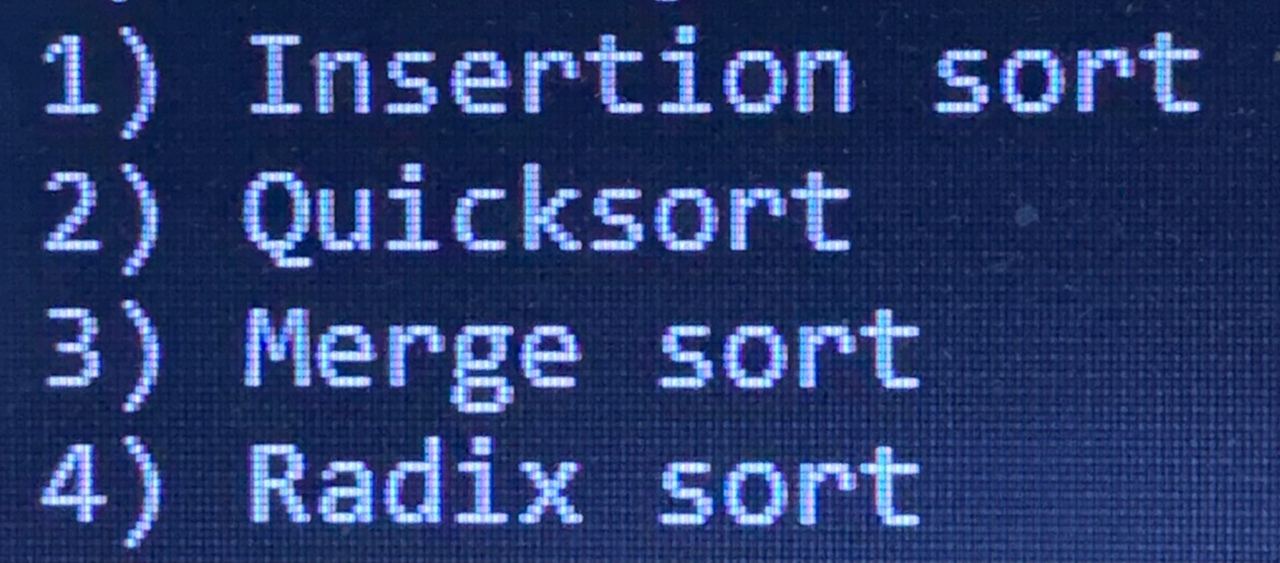
2) **Быстрая сортировка Хоара.** Её идея заключается в том, что из исходного массива выбирается некоторый элемент, который принимается в качестве разделителя или опорного элемента. затем все ключи меньше разделителя располагаются до него, а все большие - после него. Перестановка элементов выполняется путём обмена местами ключей, которые необходимо переместить в другую часть массива. При этом обмениваются ключи, расположенные на большом расстоянии друг от друга и этим достигается наивысший эффект упорядочивания.

3) **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.

4) **Поразрядная сортировка**. Сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.

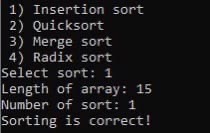
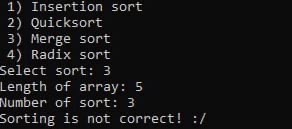
**Руководство пользователя**

При запуске программы пользователю предстоит выбрать любое целое положительное число и ввести его.

Далее необходимо выбрать любую из четырех представленных сортировок и ввести цифру от 1 до 4.

При выборе "1" массив будет отсортирован с помощью сортировки вставками (Insertion sort), при выборе "2" будет применена быстрая сортировка (Quicksort), "3" - сортировка слиянием (Merge sort), "4" - поразрядная сортировка (Radix sort).

При корректном вводе вышеуказанных данных программа производит сортировку, а после выводит уже отсортированные данные и результаты: размер массива, номер выбранной сортировки, число перестановок и сравнений элементов массива на отсортированность. "Sorting is correct!" выводится при верной работе алгоритма, "Sorting is not correct!:/" - при неверной.



**Описание программной реализации**

Структура проекта выглядит следующим образом:

1) В самом начале проекта мы подключаем все необходимые библиотеки.

2) Далее вводим переменные, которые будут отвечать за количество сравнений и перестановок.

3) Затем следует функция смены мест переменных. После нее идет код первой сортировки – сортировка вставками.

4) Перестановка элементов для быстрой сортировки.

5) Быстрая сортировка.

6) Слияние двух упорядоченных массивов.

7) Сортировка слиянием.

8) Подсчет значений байтов (одна из составляющий поразрядной сортировки).

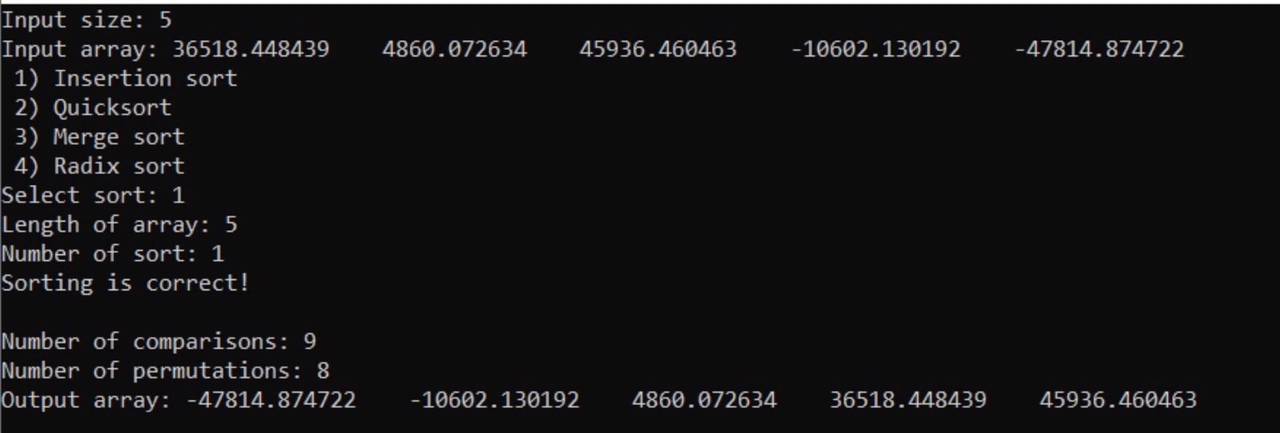
9) Сортировка по основанию с отсчетом сортировки (также составляющая поразрядной сортировки).

10) Поразрядная сортировка.

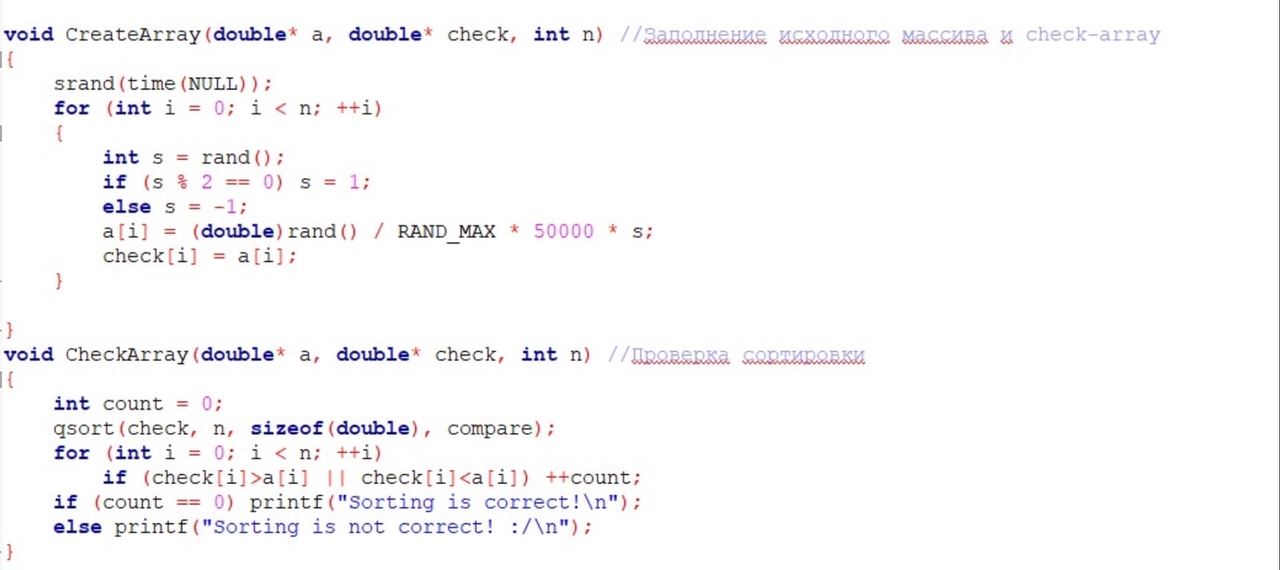
11) Функция по генерированию массива состоящего из случайных чисел.

12) Проверка сортировки.

13) Главная функция (содержит часть кода отвечающий за меню и его вывод в консоль).

**Подтверждение корректности**

Для подтверждения корректности в программе присутствует функция void СheckArray. В функции проверяется, больше или равен ли каждый следующий элемент массива по сравнению с предыдущим.

 **Результаты экспериментов**

Далее в виде таблицы представим результаты эксперимента. Первое число - количество сравнений, второе число - количество перестановок.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Вид сортировки** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **Вставками** | 45 0 | 45 45 | 30 23 | 33 25 | 31 24 | 31 22 | 30 23 | 25 17 | 26 18 | 38 32 |
| **Хоара** | 19 0 | 12 5 | 24 9 | 22 10 | 25 10 | 22 12 | 48 16 | 21 14 | 22 10 | 26 11 |
| **Слиянием** | 19 34 | 15 34 | 22 34 | 24 34 | 25 34 | 22 34 | 24 34 | 25 34 | 22 34 | 25.34 |
| **Поразрядная** | 0 50 | 0 50 | 0 50 | 0 50 | 0 50 | 0 50 | 0 50 | 0 50 | 0 50 | 0 50 |

**1) Сортировка вставками**

**Худшее время**: O(n^2) сравнений, обменов

**Лучшее время**: O(n) сравнений, 0 обменов

Самым благоприятным случаем является отсортированный массив. Наихудшим случаем является массив, отсортированный в порядке, обратном нужному. Средним случаем – случайно заполненный массив.

Для эксперимента возьмем три массива:

|  |  |
| --- | --- |
| **1** | **2** |
| 0 | 9 |
| 1 | 8 |
| 2 | 7 |
| 3 | 6 |
| 4 | 5 |
| 5 | 4 |
| 6 | 3 |
| 7 | 2 |
| 8 | 1 |
| 9 | 0 |

Первый массив – наилучший случай, так как он упорядочен. Второй массив – наихудший случай, отсортирован в порядке, обратный нужному. Третий массив – заполнен случайно сгенерированными числами

По результатам эксперимента, вносим результаты в таблицу (первое число – кол-во сравнений, второе число – кол-во перестановок

|  |  |  |  |
| --- | --- | --- | --- |
| Массив | 1 | 2 | 3 |
| Результаты | 45 0 | 45 45 | 45 23 |

Количество перестановок в данном случае зависит от упорядоченности массива.

Для обработки упорядоченных массивов лучше всего подходит именно сортировка вставками.

**2) Быстрая сортировка**

**Худшее время:** O(n^2)

**Лучшее время:** O(n \* log(n)

Лучшим случаем также является отсортированный массив. Средний случай – случайно заполненный массив. Худший случай трудно реализовать.

Возьмем те же массивы, что и в при первой сортировке:

|  |  |  |  |
| --- | --- | --- | --- |
| Массив | 1 | 2 | 3 |
| Результаты | 19 0 | 12 5 | 24 9 |

Сравнение с теоретической сложностью:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | 50 | 100 | 1000 | 10000 |
| Лучший случай(сравнения) | 229 | 564 | 8956 | 123596 |
| перестановки | 0 | 0 | 0 | 0 |
| Средний случай случай(сравнения) | 315 | 609 | 11650 | 137697 |
| перестановки | 64 | 152 | 2283 | 31304 |

**3) Сортировка слиянием**

**Худшее время: O(n \* log(n))**

**Среднее время: O(n \* log(n))**

**Лучшее время: O(n \* log(n))**

Пусть массив содержит n элементов. Тогда за O(n) его можно разделить на две части и после сортировки слить их вместе. Каждая из этих двух частей имеет размер n/2, и за O(n) шагов каждую из них можно поделить на две части размером n/4 и затем после сортировки слить их вместе.

Аналогично, четыре части размером n/4 за суммарное O(n) шагов делятся на части размером n/8 и сливаются вместе. Этот процесс «в глубину» продолжается столько раз, сколько раз можно число n делить на 2, до тех пор, пока размер части не станет равен 1, то есть (n log2 n) .  
  
Одним из недостатков сортировки слиянием является тот факт, что он требует много вспомогательной памяти (столько же, каков размер исходного массива) для реализации.

|  |  |  |  |
| --- | --- | --- | --- |
| Массив | 1 | 2 | 3 |
| Результаты | 19 34 | 15 34 | 19 34 |

**4) Поразрядная сортировка**

**Худшее время:** O(w \* n) w- кол-во бит, требуемых для хранения каждого ключа

Размер типа увеличился в 2 раза, поэтому для небольших n поразрядная сортировка, конечно, отстает, но перегоняет быструю на массивах из нескольких тысяч элементов. Небольшое преимущество radixSort длится приблизительно до 16000 элементов, когда данные начинают вылезать за пределы кэша. Соответственно, начинает играть роль медленный непоследовательный доступ (с шагами по 8 байт) к основной памяти.. В результате поразрядная сортировка снова выбивается вперед лишь после 500000 элементов.

**Заключение**

В заключение хочется отметить, что программа была успешно реализована.

Было использовано четыре вида необходимых сортировок: вставками, быстрая, слиянием и поразрядная.

Было замерено количество сравнений и перестановок во время реализации сортировки.

**Приложение**

int main()

{

int length; /\*Array length\*/

int a = 0;

int exitcode = 0;

printf("Input size: ");

scanf\_s("%d", &length);

double\* arr = (double \*)malloc(length \* sizeof(double)); /\*Original array\*/

double\* add = (double\*)malloc(length \* sizeof(double)); /\*Additional array (radix sort)\*/

double\* check = (double\*)malloc(length \* sizeof(double)); /\*Check-array\*/

CreateArray(arr, check, length);

printf("Input array: ");

for (int i=0; i < length; i++) printf("%f ", arr[i]); printf("\n");

printf(" 1) Insertion sort\n");

printf(" 2) Quicksort\n");

printf(" 3) Merge sort\n");

printf(" 4) Radix sort\n");

printf("Select sort: ");

scanf\_s("%d", &a);

//Выбор сортировки

do

{

switch (a)

{

case 1:

insert\_sort(arr, length);

exitcode = 1;

break;

case 2:

quickSort(arr, 0, length - 1);

exitcode = 1;

break;

case 3:

mergeSort(arr, length);

exitcode = 1;

break;

case 4:

radixSort(arr, add, length);

exitcode = 1;

break;

default:

printf("Wrong value, try again\n\n");

printf("Enter the number of sort: ");

scanf\_s("%d", &a);

}

} while (exitcode != 1);

printf("Length of array: %d\n", length);

printf("Number of sort: %d\n", a);

CheckArray(arr, check, length);

printf("\n");

printf("Number of comparisons: %d\n", comp);

printf("Number of permutations: %d\n", perm);

printf("Output array: ");

for (int i=0; i < length; i++) printf("%f ", arr[i]); printf("\n");

free(arr);

free(add);

free(check);

}