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/ка группы 382003-1

Тятюшин Максим Дмитриевич

**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

1) Реализовать сортировку пузырьком, расчёской, сортировку слиянием, а также поразрядную сортировку для типа данных double.

2) Замерить число перестановок и количество сравнений выполненное при сортировке.

# Метод решения

**Сортировка пузырьком**: Алгоритм пузырьковой сортировки сравнивает и обменивает соседние элементы в массиве, которую нужно отсортировать. Для массива (k1,k2,…,kn) алгоритм сначала (n-1) раз выполняет операции сравнения и обмена между элементами последовательных пар: (k1,k2),…,(kn-1,kn)

После завершения итерации самый большой элемент массива перемещается в его конец. Далее алгоритм повторяется для массива (k’1, k’2, ..., k’n-1).

**Сортировка расчёской**: Алгоритм сортировки расчёской сравнивает элементы стоящие друг от друга на максимальном расстоянии, то есть на расстоянии равном размеру массива минус один. Затем нужно разделить этот шаг на фактор уменьшения и пройтись по массиву вновь и т. д. Когда расстояние между сравниваемыми элементами будет равно единице, то есть будут сравниваться соседние элементы, остается применить одну итерацию сортировки пузырьком.

**Сортировка слиянием**: Алгоритм сортировки слиянием делит исходный массив на два примерно равных массива. Процедура продолжается до тех пор, пока не получатся единичные массивы. Затем сравниваются последовательно единичные массивы, меньшее число идет на первое место, большее идет на второе место в массиве полученном «слиянием». Получаются новые массивы состоящие из двух или одного элемента(в случае нечетного числа элементов исходного массива). Далее сравниваются элементы новых массивов, сначала первый с первым, затем первый со вторым. Получаются новые массивы большего размера. Продолжая алгоритм далее, получим отсортированный массив состоящий из элементов исходного массива.

**Поразрядная сортировка:** Алгоритм поразрядной сортировки сортирует элементы, первоначально группируя отдельные цифры одного и того же разряда. Идея Radix Sort состоит в том, чтобы выполнять сортировку по цифрам, начиная с наименее значащей цифры до самой старшей цифры , в соответствии с их порядком увеличения или уменьшения.

# Руководство пользователя

Пользователю необходимо:

1)Ввести число соответствующее алгоритму сортировки.

Число 1 соответствует сортировке пузырьком.

Число 2 соответствует сортировке расческой.

Число 3 соответствует сортировке слиянием.

Число 4 соответствует поразрядной сортировке.

Число 5 соответствует проверке сортировки массива.

2)Ввести длину массива.

3)Ввести поэлементно массив.

4)Если была выбрана поразрядная сортировка, следует ввести точность чисел, вводимых пользователем. Точность — наибольшее кол-во цифр после запятой числа.

# Описание программной реализации

Функции используемые в программе:

1. bubble\_sort — функция сортировки массива пузырьком.
2. comb\_sort — функция сортировки массива расческой.
3. merge и mergeSort — функции сортировки массива слиянием.
4. get\_max и radix\_sort — функции поразрядной сортировки массива.
5. сompare и qsort – функции быстрой сортировки (проверка сортировки).

# Подтверждение корректности

Для подтверждения корректности в программе используется qsort. Вызвать ее возможно введением числа 5, как написано в руководстве пользователя.

# Результаты экспериментов

Перестановки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Длина массива | Сортировка пузырьком | Сортировка расческой | Сортировка слиянием | Поразрядная сортировка |
| 2 | 1 | 1 | 2 | 9 |
| 4 | 1 | 2 | 8 | 19 |
| 8 | 19 | 5 | 28 | 39 |
| 16 | 58 | 17 | 68 | 77 |
| 32 | 258 | 43 | 162 | 151 |
| 64 | 1112 | 103 | 388 | 303 |
| 128 | 3892 | 207 | 892 | 603 |
| 256 | 16078 | 509 | 2048 | 1218 |
| 512 | 66156 | 1128 | 4608 | 2444 |
| 1024 | 260239 | 2459 | 10244 | 4872 |

Сравнения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Длина массива | Сортировка пузырьком | Сортировка расческой | Сортировка слиянием | Поразрядная сортировка |
| 2 | 1 | 2 | 1 | 0 |
| 4 | 5 | 7 | 4 | 0 |
| 8 | 25 | 42 | 19 | 0 |
| 16 | 123 | 86 | 47 | 0 |
| 32 | 497 | 179 | 119 | 0 |
| 64 | 1853 | 396 | 302 | 0 |
| 128 | 7889 | 1258 | 756 | 0 |
| 256 | 32186 | 2412 | 1719 | 0 |
| 512 | 130997 | 5053 | 3967 | 0 |
| 1024 | 523775 | 12091 | 8988 | 0 |

По результатам представленным в виде таблиц видно, что поразрядной сортировки количество сравнений равно нулю. Такие результаты получаются из-за того, что поразрядная сортировка линейная.

Из таблиц видно, что сортировка пузырьком является самой медленной из представленных сортировок. Сортировка расческой в разы быстрее сортировки пузырьком, ведь в основная идея в том, что промежуток между сравниваемыми элементами может быть больше, чем единица, что позволяет значительно улучшить эффективность.

Временная сложность алгоритма равна количеству элементов в массиве, умноженному на его логарифм. Это одновременно и достоинство, и недостаток метода. Время его работы не меняется даже в худшем случае, когда исходный массив отсортирован в обратном порядке. Однако при обработке полностью отсортированных данных алгоритм не обеспечивает выигрыша во времени.

Из первой таблицы видно, что скорость поразрядной сортировки сильно падает при увеличении количества элементов в массиве.

# Заключение

1. Сортировка пузырьком медленно работает на тестах, в которых маленькие элементы стоят в конце массива. (Сложность по времени: худшее/среднее/лучшее время соответственно O(n^2)/O(n^2)/O(n^2). Затрата памяти: O(1).)
2. Сортировка расческой — улучшенная сортировка пузырьком. Идея состоит в устранении проблем сортировки пузырьком. То есть случаев, когда маленькие элементы находятся в конце массива. (Сложность по времени: худшее/среднее/лучшее время соответственно Ω(n2) /Ω(n2/2p) /O(n) . Затрата памяти: O(1).)
3. Сортировка слиянием. Недостатки метода заключаются в том, что он требует дополнительной памяти по объему равной объему сортируемого файла. Поэтому для больших файлов проблематично организовать сортировку слиянием в оперативной памяти. (Сложность по времени: худшее/среднее/лучшее время соответственно O(n logn)/O(n logn)/O(n logn) . Затрата памяти: O(n).)
4. Поразрядная сортировка. В отличие от сортировки radix, quicksort универсален, а сортировка по методу radix полезна только для целых ключей фиксированной длины. В этом заключается недостаток сортировки. (Сложность по времени: худшее/лучшее время соответственно O(n×k) /O(n) . Затрата памяти: O(n+k).)

# Приложение

int main()

{

int pr[1] = {0};

int sr[1] = {0};

int E;

long int g;

int i=0;

setlocale(0,"");

printf("Введите число соответсвующее алгоритму сортировки\n1 - сортировка пузырьком\n2 - сортировка расчёской\n3 - сортировка слиянием\n4 - поразрядная сортировка\n5 - проверка\n");

scanf("%d", &E);

printf("Введите кол-во чисел массива: ");

scanf("%ld", &g);

if (E==1){

double \*arr;

arr = (double\*)malloc(g\*sizeof(double));

printf("Введите элементы массива: ");

for(i=0;i<g;i++)

scanf("%lf", &arr[i]);

bubble\_sort(arr, g);

printf("Отсортированный массив: ");

for(i=0 ; i<g ; i++)

printf("%lf ",arr[i]);

}

if (E==2){

double \*arr;

arr = (double\*)malloc(g\*sizeof(double));

printf("Введите элементы массива: ");

for(i=0;i<g;i++)

scanf("%lf", &arr[i]);

comb\_sort(arr, g);

printf("Отсортированный массив: ");

for(i=0 ; i<g ; i++)

printf("%lf ",arr[i]);

}

if (E==3)

{

double \*arr;

arr = (double\*)malloc(g\*sizeof(double));

printf("Введите элементы массива: ");

for(i=0;i<g;i++)

scanf("%lf", &arr[i]);

mergeSort(arr, 0, g - 1, sr, pr);

printf("Отсортированный массив: ");

for(i=0 ; i<g ; i++)

printf("%lf ",arr[i]);

printf("\n");

printf("Количество перестановок: %d\n", pr[0]);

printf("Количество сравнений : %d\n", sr[0]);

}

if (E==4)

{

double\* arr = (double\*)malloc(g \* sizeof(double));

double\* arr1 = (double\*)malloc(g \* sizeof(double));

long\* cnts = (long\*)malloc(1024 \* sizeof(double));

printf("\n");

printf("Введите элементы массива: ");

for(i=0;i<g;i++)

scanf("%lf", &arr[i]);

Radix\_Sort(arr, arr1, cnts,g, pr);

printf("Отсортированный массив: ");

for(i=0 ; i<g ; i++)

printf("%lf ",arr[i]);

printf("\nКоличество перестановок : %d\n", pr[0]);

}

if(E==5) {

double arr[g];

printf("Введите элементы массива: ");

for(i=0;i<g;i++)

scanf("%lf", &arr[i]);

qsort(arr, g, sizeof(double), compare);

printf("\nОтсортированный массив: ");

for( i = 0 ; i < g; i++ )

printf("%lf ", arr[i]);

}

return(0);

}