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
3821Б1ПМ2
Трофимов В.В.

Проверил:
преподаватель каф. МОСТ,
Волокитин В.Д.

Нижний Новгород
2021

Содержание

Постановка задачи

Целями данной работы были:

1) реализация указанных сортировок для типа данных числа с плавающей запятой двойной точности;

2) замеры числа перестановок и количество сравнений, выполненное при сортировке;

3) эксперименты, показывающие теоретическую сложность алгоритма;

Метод решения

1. Сортировка выбором: на каждой итерации будем находить минимум в массиве после текущего элемента и менять его с ним, если нужно. Таким образом после i-ой итерации первые i элементов будут стоять на своих местах. Асимптотика: $O(n^2)$ в лучшем, худшем и среднем случае.

2. Сортировка Шелла: зафиксируем некоторое расстояние. Тогда элементы массива разобъются на классы - ва один класс попадают элементы, расстояние между которыми кратно зафиксированному расстоянию. Каждый класс нужно отсортировать вставками: будем менять текущий элемент с предыдущим, пока они стоят в неправильном порядке. В данной работе использована последовательность Шелла - первый шаг между элементами равно половине длины массива, каждый следующий в два раза меньше предыдущего. Асимптотика в худшем случае - $O(n^2)$.

3. Сортировка слиянием: разделим массив пополам, рекурсивно отсортируем части, после чего выполним процедуру слияния: поддерживаем два указателя на текущие элементы первого и второго подмассива. Из двух элементов выбираем минимальный и записываем в буферный массив и сдвигаем указатели. Алгоритм требует $O(n)$ дополнительной памяти. Слияние работает за $O(n)$, уровней слияния $O(\log n)$, поэтому асимптотика $O(n*\log n)$.

4. Поразрядная сортировка: разобьем каждое число на разряды. Для каждой цифры используем сортировку подсчётом. Пройдём по массиву и подсчитаем количество вхождений каждого элемента. Теперь можно пройти по массиву значений и выписать каждое число столько раз, сколько нужно. Асимптотика: $O(n)$.

Руководство пользователя

При запуске программы предлагается выбрать размер максимального по размеру массива и шаг увеличения размера массива. После этого на выбор пользователя предоставляется меню с семью вариантами. Первый запускает сортировку выбором с указанным увеличением размера массива, второй делает тоже самое, но используя сортировку Шелла. Третий выполняет эту же операцию путем сортировки слиянием, а четвертый - поразрядной. Пятый пункт меню позволяет присвоить каждому элементу массива псевдослучайное число. С помощью шестого пункта можно изменить уже заданный размер массива и шаг прохода. Седьмой, он же нулевой пункт, позволяет выйти из программы. Каждая сортировка выводит длину отсортированного массива, количество сравнений и перестановок, сумму действий, деленных на теоретическую сложность.

Описание программной реализации

Программа состоит из одного файла с данными подпрограммами:

double* Randomize(double a[], int size) - получает на вход указатель на массив и его длину. На выходе возвращает указатель на рандомизированный массив с элементами типа double в промежутке [-1000;1000). В качестве сида используется текущее время системы от 00:00, 1 января 1970 UTC.

void Selection(double a[],int size) - получает на вход указатель на массив и его длину. Выполняет сортировку выбором.

void Shell(double a[], int size) - получает на вход указатель на массив и его длину. Выполняет сортировку Шелла.

void merge(double a[], int left, int right) - получает на вход указатель на массив, левый индекс сортируемой части, правый индекс сортируемой части. Выполняет сортировку слиянием.

void createCounters(double* in, int* counters, int N) - вспомогательная функция поразрядной сортировки. Получает на вход указатель на сортируемый массив, указатель на вспомогательный массив, длину сортируемого массива. Считает одинаковые цифры в каждом разряде.

void SignedRadixSort(short int offset, int N, double* in, double* out, int* count) - вспомогательная функция поразрядной сортировки. Получает на вход позицию, с которой нужно вставлять число в выходной массив, длину сортируемого массива, указатели на сортируемый массив и два вспомогательных массива. Сортирует отрицательные числа.

void radixPass(short int offset, int N, double* in, double* out, int* count) - вспомогательная функция поразрядной сортировки. Получает на вход позицию, с которой нужно вставлять число в выходной массив, длину сортируемого

массива, указатели на сортируемый массив и два вспомогательных массива. Выполняет поразрядную сортировку.

void RadixSort(double* in, double* out, int* counters, int N) - получает на вход указатели на сортируемый массив и два вспомогательных, длину сортируемого массива. Сортирует массив с помощью вышеописанных функций.

void callradixsort(double* in, int N) - вспомогательная функция поразрядной сортировки. Получает на вход указатель на сортируемый массив и его длину. Создает вспомогательные массивы double* out , int* counters и вызывает RadixSort().

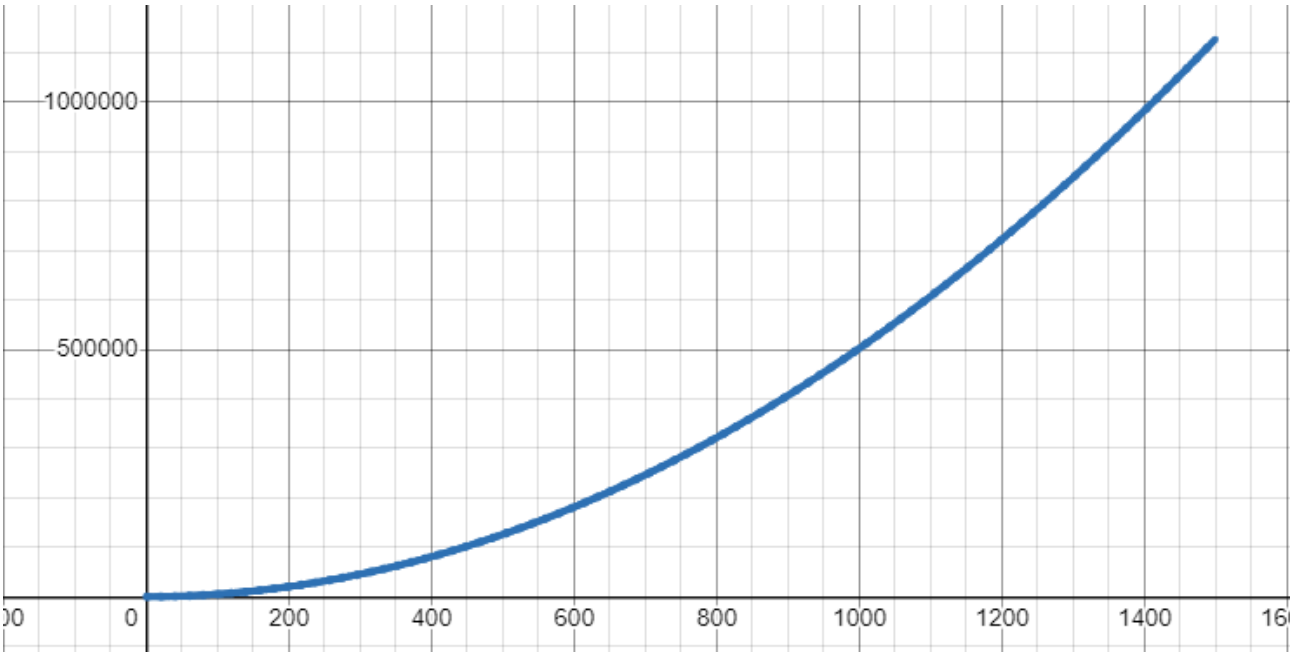
Подтверждение корректности

Для подтверждения корректности в программе написана функция isSort(), которая проверяет отсортированность массива. Функция сравнивает каждый элемент массива с предыдущим элементом, и в случае, если последующий элемент оказывается меньше прошлого, на консоль выводится сообщение "sort order isn't correct!" и программа завершается со значением 1.

Результаты экспериментов

Для генерации элементов массивов был использован сид time(NULL). Длина массива менялась от двух до тысячи пятисот с увеличением массива на 2. После сортировки каждого массива вызывалась проверка отсортированности. После этого на консоль выводилась длина массива, количество сравнений и перестановок, количество действий, деленное на теоретическую сложность. По полученным данным строился график в графическом калькуляторе Desmos.

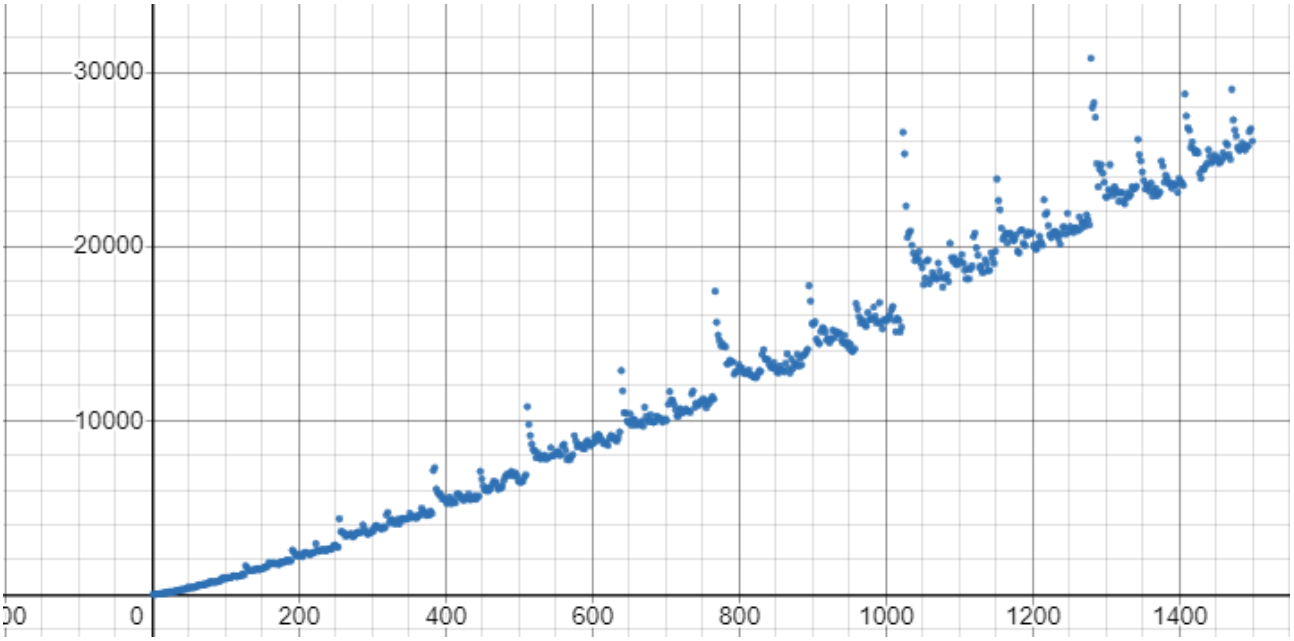
1. Сортировка выбором



Сложность данной сортировки O(n²), поэтому график функции является параболой.При делении на n²получается точки, попадающие в окрестность 0.5.

```
1330: (885114_1321);0.501122
1332: (887777_1319);0.501118
1334: (890444_1330);0.501122
1336: (893115_1329);0.501118
1338: (895790_1330);0.501116
1340: (898469_1334);0.501116
1342: (901152_1334);0.501113
1344: (903839_1339);0.501113
1346: (906530_1336);0.501108
1348: (909225_1337);0.501106
1350: (911924_1341);0.501106
1352: (914627_1341);0.501103
1354: (917334_1347);0.501103
1356: (920045_1347);0.501101
1358: (922760_1346);0.501098
1360: (925479_1355);0.501100
1362: (928202_1356);0.501098
1364: (930929_1358);0.501096
1366: (933660_1357);0.501093
1368: (936395_1359);0.501091
1370: (939134_1360);0.501089
1372: (941877_1362);0.501087
1374: (944624_1365);0.501086
1376: (947375_1369);0.501086
1378: (950130_1373);0.501085
1380: (952889_1369);0.501081
1382: (955652_1374);0.501081
```

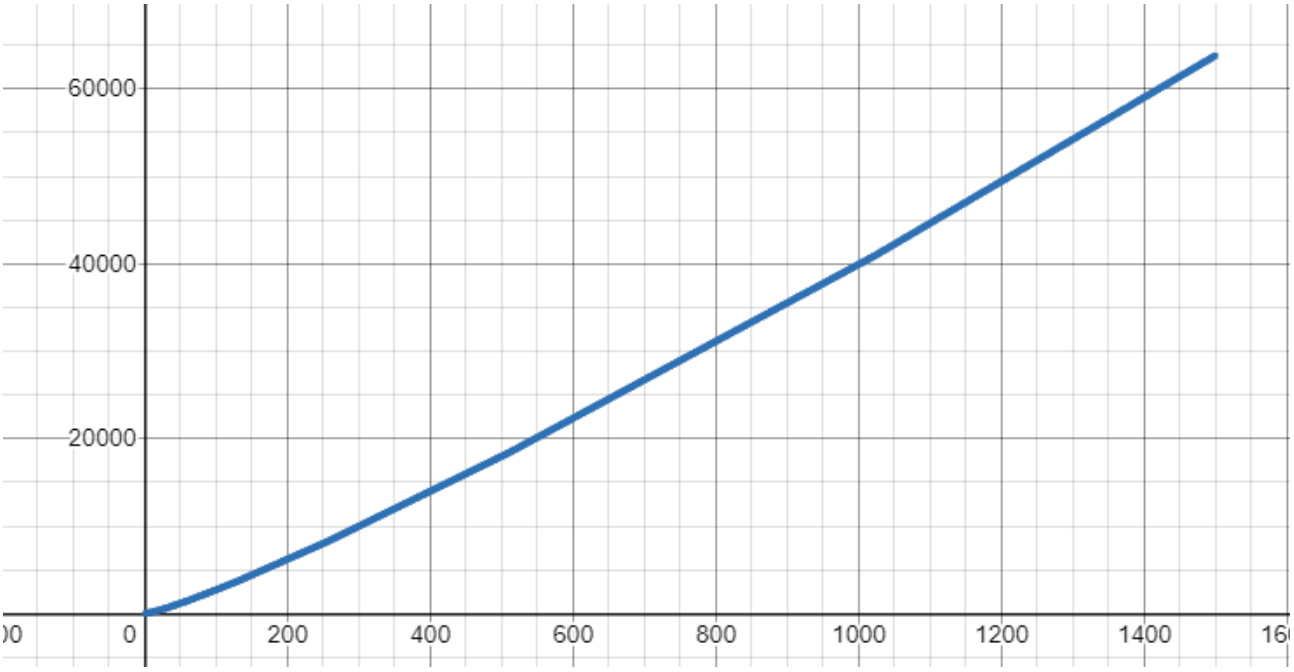
2. Сортировка Шелла



Сложность сортировки O(n²). Пики увеличения сложности вызваны использованной последовательностью шагов расстояний сравнения элементов. При делении количества действий на теоретическую сложность получаем 0.015 после определенного количества элементов.

```
1076: (9688_8344);0.015575
1078: (9707_8409);0.015589
1080: (9724_8236);0.015398
1082: (9743_7954);0.015116
1084: (9761_8138);0.015232
1086: (9780_8582);0.015569
1088: (9794_10459);0.017109
1090: (9813_9841);0.016542
1092: (9831_9456);0.016174
1094: (9850_9254);0.015962
1096: (9867_8557);0.015338
1098: (9886_8988);0.015655
1100: (9904_8506);0.015215
1102: (9923_8383);0.015074
1104: (9939_9338);0.015816
1106: (9958_9035);0.015527
1108: (9976_9037);0.015487
1110: (9995_8565);0.015064
1112: (10012_8429);0.014913
1114: (10031_8454);0.014895
1116: (10049_8781);0.015119
1118: (10068_8753);0.015058
1120: (10083_9384);0.015519
1122: (10102_9277);0.015394
1124: (10120_8878);0.015037
1126: (10139_9142);0.015207
1128: (10156_8375);0.014564
1130: (10175_8666);0.014755
1132: (10193_8969);0.014954
1134: (10212_8787);0.014774
1136: (10228_9358);0.015177
1138: (10247_9633);0.015351
1140: (10265_9674);0.015342
1142: (10284_9822);0.015417
```

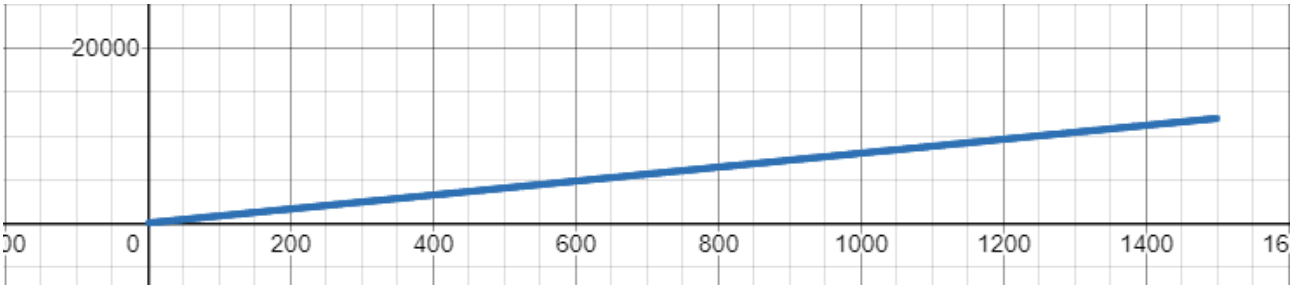
3. Сортировка слиянием



Сложность данной сортировки O(n*log n). График функции - логарифмическая, повернутая на $\pi/4$. Если поделить на теоретическую сложность, то получим 1.7

```
1388: (43824_14608);0.173243
1390: (43896_14632);0.173278
1392: (43968_14656);0.173313
1394: (44040_14680);0.173347
1396: (44112_14704);0.173382
1398: (44184_14728);0.173416
1400: (44256_14752);0.173451
1402: (44328_14776);0.173485
1404: (44400_14800);0.173520
1406: (44472_14824);0.173554
1408: (44544_14848);0.173588
1410: (44616_14872);0.173622
1412: (44688_14896);0.173656
1414: (44760_14920);0.173689
1416: (44832_14944);0.173723
1418: (44904_14968);0.173757
1420: (44976_14992);0.173790
1422: (45048_15016);0.173823
1424: (45120_15040);0.173857
1426: (45192_15064);0.173890
1428: (45264_15088);0.173923
1430: (45336_15112);0.173956
1432: (45408_15136);0.173989
1434: (45480_15160);0.174022
1436: (45552_15184);0.174055
1438: (45624_15208);0.174087
1440: (45696_15232);0.174120
1442: (45768_15256);0.173439
1444: (45840_15280);0.173471
1446: (45912_15304);0.173503
1448: (45984_15328);0.173535
1450: (46056_15352);0.173567
1452: (46128_15376);0.173599
1454: (46200_15400);0.173631
1456: (46272_15424);0.173662
1458: (46344_15448);0.173694
1460: (46416_15472);0.173726
```

4. Поразрядная сортировка



Сортировка имеет линейную сложность O(k*(n+m)+n),где k-длина элемента, m - количество возможных значений. При делении на размер массива, то получим число 8

```
1376: (0_11008);8.000000
1378: (0_11024);8.000000
1380: (0_11040);8.000000
1382: (0_11056);8.000000
1384: (0_11072);8.000000
1386: (0_11088);8.000000
1388: (0_11104);8.000000
1390: (0_11120);8.000000
1392: (0_11136);8.000000
1394: (0_11152);8.000000
1396: (0_11168);8.000000
1398: (0_11184);8.000000
1400: (0_11200);8.000000
1402: (0_11216);8.000000
1404: (0_11232);8.000000
1406: (0_11248);8.000000
1408: (0_11264);8.000000
1410: (0_11280);8.000000
1412: (0_11296);8.000000
1414: (0_11312);8.000000
1416: (0_11328);8.000000
1418: (0_11344);8.000000
1420: (0_11360);8.000000
1422: (0_11376);8.000000
1424: (0_11392);8.000000
1426: (0_11408);8.000000
1428: (0_11424);8.000000
1430: (0_11440);8.000000
1432: (0_11456);8.000000
1434: (0_11472);8.000000
1436: (0_11488);8.000000
1438: (0_11504);8.000000
1440: (0_11520);8.000000
1442: (0_11536);8.000000
1444: (0_11552);8.000000
1446: (0_11568);8.000000
1448: (0_11584);8.000000
1450: (0_11600);8.000000
1452: (0_11616);8.000000
1454: (0_11632);8.000000
1456: (0_11648);8.000000
1458: (0_11664);8.000000
1460: (0_11680);8.000000
1462: (0_11696);8.000000
1464: (0_11712);8.000000
1466: (0_11728);8.000000
```

Заключение

В ходе лабораторной работы были реализованы сортировки выбором, Шелла, слиянием, поразрядная для типа данных числа с плавающей запятой двойной точности. Выполнены замеры числа перестановок и количество сравнений, выполненное при каждой из сортировок.Поставлены эксперименты, показывающие теоретическую сложность алгоритмов.

Приложение

https://github.com/SirTruber/mp1-3821B1PM2