**Министерство образования Республики Беларусь**

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

|  |
| --- |
|  |
|  |

ОТЧЕТ

по лабораторной работе

на тему:

СОРТИРОВКИ

|  |  |  |
| --- | --- | --- |
| Выполнил  Студент гр. 051006 |  | И.Д. Сидоренко |
| Проверил |  | Асс. C.В. Болтак |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск, 2021

1. Задание на лабораторную работу

Произвести анализ сортировок массивов в соответствии с вариантом, выданным преподавателем. Для чего отсортировать по возрастанию массивы целочисленных элементов различной размерности: 10 элементов, 100 элементов, 2000 элементов. Анализ произвести по числу сравнений и перестановок двух элементов. Исследования производить над массивами трех типов:

1. Массив, содержащий не отсортированные элементы;
2. Массив, содержащий отсортированные элементы;
3. Массив, содержащий элементы, отсортированные в обратном порядке.

Виды сортировок: Шейкерная сортировка, Сортировка Шелла, Quicksort

Требования к программе:

- программа должна работать в режиме формы, а не консоли;

- интуитивно понятный пользовательский интерфейс;

- построение гистограмм сравнений и перестановок, используя свойство Canvas;

Результат работы программы: таблица, гистограммы сравнений и перестановок.

2.Описание сортировок с примерами

2.1 Шейкерная сортировка

Шейкерная сортировка является усовершенствованным методом пузырьковой сортировки. Анализируя метод пузырьковой сортировки, можно отметить два обстоятельства:

* Если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, ее можно исключить из рассмотрения.
* При движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо. Эти две идеи приводят к модификациям в методе пузырьковой сортировки.
* От последней перестановки до конца (начала) массива находятся отсортированные элементы. Учитывая данный факт, просмотр осуществляется не до конца (начала) массива, а до конкретной позиции. Границы сортируемой части массива сдвигаются на 1 позицию на каждой итерации.
* Массив просматривается поочередно справа налево и слева на право.
* Просмотр массива осуществляется до тех пор, пока все элементы не встанут в порядке возрастания (убывания).
* Количество просмотров элементов массива определяется моментом упорядочивания его элементов.

Пример использования Шейкерной сортировки:



2.2 Сортировка Шелла

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

1. Выбираем интервал, равный половине длины массива (10/2=5)
2. Сравниваем i=1 и i=1+5 элемент и меняем местами, если шестой элемент меньше
3. Проделываем то же самое со 2 и 7, 3 и 8, 4 и 9, 5 и 10 элементами
4. В итоге массив выглядит так:

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

5)Выбираем интервал, равный половине половины длины массива с округлением в большую сторону (5/2=2.5=3)

6) Сравниваем i=1 и i=1+3 элемент и меняем местами, если четвертый меньше

7) То же самое с 0 и 6, 1 и 3, 8 и 5

8) По принципу челночной сортировки продолжаем сравнивать. 5 сравним с 2. 2 меньше—оставляем как есть и возвращаемся к нашей сортировке

9) Сравниваем 6 и 4, меняем местами, и, как и в предыдущем случае, продолжим сравнивать, сравниваем 4 с 0. Оставляем как есть. Возвращаемся к сортировке.

10) Сравниваем 9 и 3, 8 и 7

11) Массив:

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

9) Теперь интервал, равный 3 делим еще раз пополам с округлением в большую сторону. Интервал равен 2.

10) Сравниваем и меняем местами 2 и 1

11) По принципу челночной сортировки, сравниваем 1 сравниваем с 0. Меняем.

12) 2 и 4, 2 и 1, 5 и 3, 3 и 2, 4 и 7, 4 и 3, 5 и 6, 5 и 4, 7 и 9, 7 и 5, 6 и 8, 6 и 7, 9 и 8, 8 и 7.

12) Массив:

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

Массив отсортирован.

2.3 QuickSort

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Исходный массив: | | | | | | | |
| 44 | 55 | 12 | 42 | 94 | 6 | 18 | 67 |

N=8, L=1, R=8, Mid= (1+8) div 2 = 4, Х=42

После первого прохода по массиву массив разделился на две части относительно числа 42:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 18 | 6 | 12 | **42** | 94 | 55 | 44 | 67 |

Последние значения индексов: i=5, j=3

Сортируем левую часть (от элемента 42) с границами 1 и 3. (правая часть имеет границы 5 и 8)

n=3, Mid= ((L=1)+(R=3)) div 2 = 2, Х=6

Меняем местами 18 и 6:

|  |  |  |
| --- | --- | --- |
| **6** | 18 | 12 |

Последние значения индексов: i=2, j=1. Так как L=j=1, то левый рекурсивный вызов не срабатывает. Так как i=2 и R=3 (i<R), то срабатывает правый рекурсивный вызов:

Mid= ((L=2)+(R=3)) div 2 = 2, Х=18.

Меняем местами 18 и 12:

|  |  |
| --- | --- |
| 12 | 18 |

Последние значения индексов: i=3, j=2. Так как L=j=2, то левый рекурсивный вызов не срабатывает.

Так как i=3=R, то правый рекурсивный вызов не срабатывает.

Сортируем правую часть (от элемента 42) с границами 5 и 8.

n=4, Mid= (5+8) div 2 = 6, Х=55

|  |  |  |  |
| --- | --- | --- | --- |
| 44 | **55** | 94 | 67 |

Переставляем элемент 55 сам с собой.

Последние значения индексов: i=7, j=5. Так как L=j=5, то левый рекурсивный вызов не срабатывает. Так как i=7 и R=8, то срабатывает правый рекурсивный вызов. Исходный массив:

|  |  |
| --- | --- |
| **94** | 67 |

Mid= ((L=7)+(R=8)) div 2 = 7, Х=94. После прохода по массиву меняем местами 94 и 67:

|  |  |
| --- | --- |
| 67 | 94 |

Последние значения индексов: i=8, j=7. Так как L=7=j, i=8=R, то рекурсивных вызовов нет.

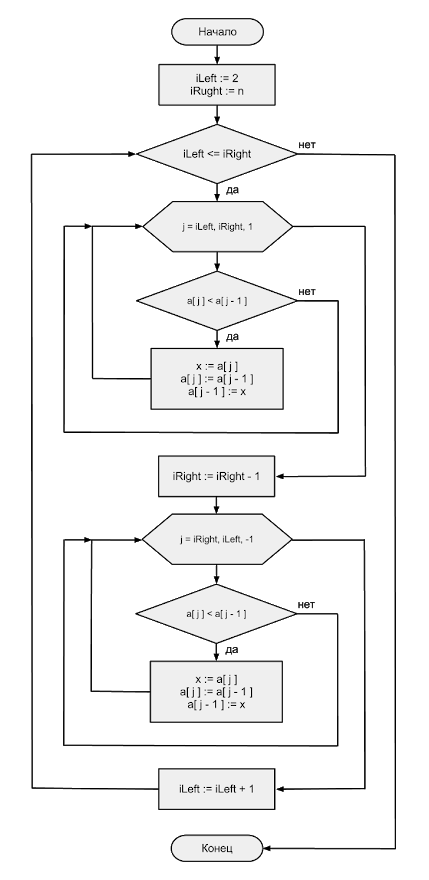
Массив:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 12 | 18 | **42** | 44 | 55 | 67 | 94 |

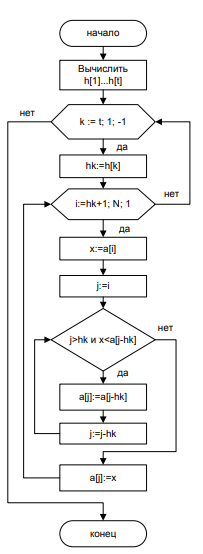
Массив отсортирован.

3. Схемы алгоритмов сортировок

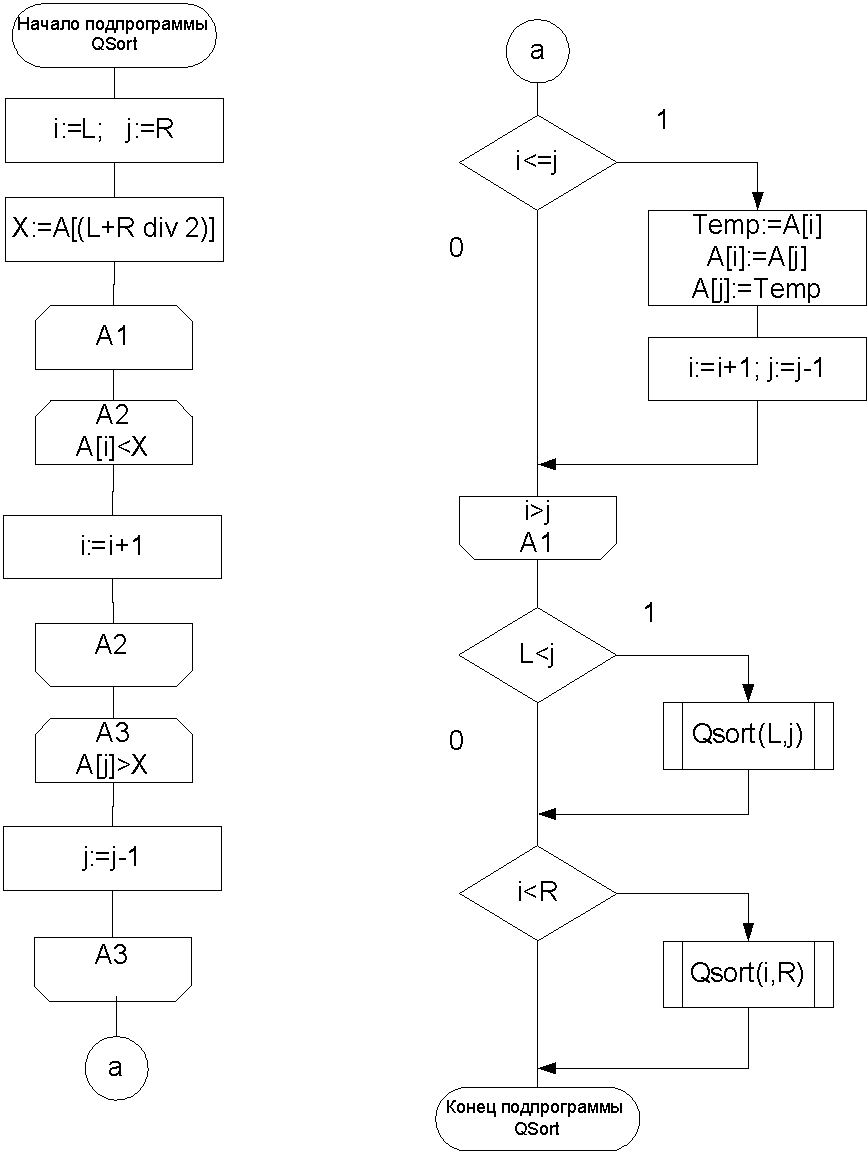
3.1 Шейкерная сортировка



* 1. Сортировка Шелла



* 1. QuickSort



4. Анализ сортировок

4.1 Шейкерная сортировка

Анализ шейкерной сортировки довольно сложен. Минимальное число сравнений равно **Cmin = ( n–1)**.

Итак, для числа перестановок справедливы следующие выражения:

**Mmin = 0;**

**Mavg = 3 \* (n2 – n) / 2;**

**Mmax = 3 \* (n2 – n) / 4,**

где **n** – число элементов в сортируемой последовательности; **Мmin**, **Мavg**, **Мmax** – соответственно минимальное, среднее и максимальное значение перестановок.

4.2 Сортировка Шелла

Приводимая программа не ориентирована на некую определенную последовательность расстояний. Все **t** расстояний обозначаются соответственно

**h1, h2, ..., ht** ,

для них выполняются условия

**ht=1;**

**hi+1<hi.**

Каждая **h**-сортировка программируется как сортировка с помощью прямого включения.

В алгоритме не известно, какие расстояния дают наилучшие результаты.

Дональд Кнут рекомендует такую последовательность:

1, 3, 7, 15, 31, ...,

то есть

**hk-1=2hk+1,**

**ht=1** и **t = [log2n] - 1.**

4.3 Быстрая сортировка

Анализ эффективности быстрой сортировки достаточно труден. Будет лучше показать ее вычислительную сложность, подсчитав число сравнений при некоторых идеальных допущениях. Пусть размерность исходного массива n является степенью двойки, n=2k(k=log2n), а центральный элемент располагается точно посередине каждого массива и разбивает его на два подмассива примерно одинаковой длины. При первом просмотре производится (n-1) сравнений. В результате создаются два подмассива размером (n/2). На следующей фазе обработка каждого подмассива требует примерно (n/2) сравнений. Общее число сравнений на этой фазе равно 2\*(n/2)=n. На следующей фазе обрабатываются четыре подмассива, что требует 4\*(n/4) сравнений, и т.д. Процесс разбиения прекращается после k проходов, когда получившиеся подмассивы содержат по одному элементу. Общее число сравнений приблизительно равно

n +2(n/2) + 4(n/4) + ...+ n(n/n ) = n + n +...+n = n\* k = n\* log2 n.

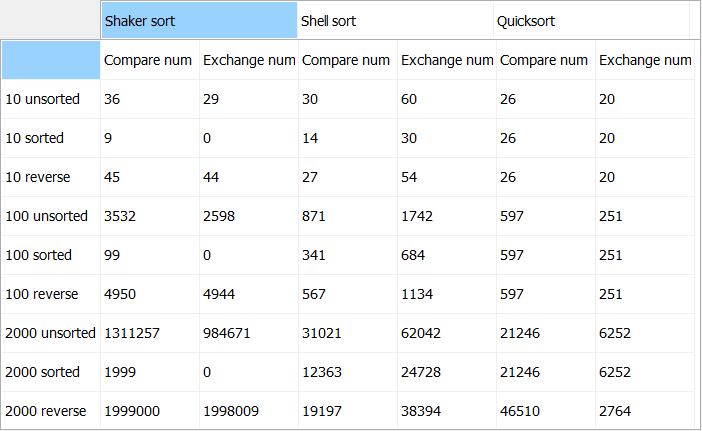
Таким образом, для массива общего вида вычислительная сложность быстрой сортировки равна (n\*log2n). Идеальный случай, рассмотренный выше, фактически возникает тогда, когда массив уже отсортирован по возрастанию. Тогда центральный элемент попадает точно в середину каждого подмассива. Наихудшим сценарием для быстрой сортировки будет тот, при котором центральный элемент все время попадает в одноэлементный подмассив, а все прочие элементы остаются во втором подмассиве. Это происходит тогда, когда центральным всегда является наименьший элемент (или наибольший). В этом случае на первом проходе производится n сравнений, а больший подмассив содержит (n-1) элементов. На следующем проходе этот подмассив требует (n-1) сравнений и дает подмассив из (n-2) элементов и т.д. Сложность сортировки для худшего случая пропорциональна n 2, т.е. не лучше, чем для сортировок вставками и выбором. Однако этот случай маловероятен на практике. В целом, средняя производительность быстрой сортировки выше, чем у всех рассмотренных ранее сортировок. Алгоритм быстрой сортировки (QuickSort) выбирается за основу в большинстве универсальных сортирующих утилит.

Количество перестановок:

cреднее**: Mavg=n\*ln(n)**

минимальное: **Mmin=1/3\*n\*ln(n)**

6. Результаты работы сортировок



7. Вывод о эффективности сортировок

Шейкерная сортировка уступает сортировкам Шелла и Быстрой. Сортировка Шелла при использовании эффективной последовательности может быть сопоставима по эффективности с быстрой сортировкой в если массив отсортирован(или обратно отсортирован) или же в случае работы с небольшими массивами. Быстрая сортировка является самой эффективной из всех трёх, но она может быть не эффективна для малых или для частично отсортированных массивов.