**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. И.РАЗЗАКОВА**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ СИСТЕМ**

**Отчет**

**Дисциплина «Алгоритмы и структуры данных»**

**Лабораторная работа № 9-10**

## Тема: «Анализ алгоритмов методов сортировки»

Выполнил: студент группы

ПИ(б)-3-18

Узаков У.

Проверила: Валеева А.А.

**г.Бишкек 2019**

**Контрольные вопросы**

1. **Что такое сортировка?**

Упорядочивание предметов\чисел и т.д методом определенного алгоритма.

1. **Для чего применяется сортировка?**

Для более быстрого доступа к предметам\числам.

1. **Какие виды сортировки существуют?**

Пузырьковая, шейкерная, выборка, вставка, сортировка шелла, быстрая сортировка, поразрядная и т.д

1. **Назовите критерии эффективности алгоритмов сортировки.**

Устойчивость, время выполнения, естественность поведения, размер.

1. **Инструменты применяются для оценки эффективности алгоритмов?**

Да, используется статистика для сбора информации о алгоритме.

1. **Перечислите критерии выбора алгоритма сортировки.**

При выборе алгоритма, выбирают самый результативный алгоритм по критериям размера и времени выполнения, т.е самый быстрый и самый легкий.

1. **В чем заключается обменная сортировка? Опишите суть метода.**

В массиве из любого количество элементов (n) каждый элемент сравнивается со следующим. Если условие выполняется, то происходит обмен. Сортировка организуется 2 циклами, внешний осуществляет (n-1) проход, а внутренний – (n – 2).

1. **В чем заключается сортировка выбором? Опишите суть метода.**

Есть массив из любого количества элементов (n). За каждый из (n – 1) проходов нужно в неотсортированной части массива найти наименьший (наибольший) элемент и переместить его в отсортированную часть в соответствии с заданным законом упорядочивания.

1. **В чем заключается сортировка вставками? Опишите суть метода.**

Есть массив из любого количества элементов (n). За каждый из (n – 1) проходов нужно взять элемент из неотсортированной части и сохранить его в дополнительную переменную, в отсортированной части найти позицию вставки, в которой новый элемент не нарушит упорядоченности, потом сдвинуть вправо элементы в отсортированной части, чтобы освободить найденную позицию, а затем вставить взятый элемент в найденную позицию.

1. **В чем заключается шейкерная сортировка? Опишите суть метода**

Есть массив из любого количества элементов. Задаются границы left и right для неотсортированного массива. На каждой итерации одного цикла в пределах неотсортированного массива сравниваются соседние элементы, и если наименьший стоит правее, то элементы меняются местами. После этого правая граница сдвигается влево на 1 элемент. На каждой итерации второго цикла происходят аналогичные действия, но наибольшие по значению элементы сдвигаются направо. После этого левая граница сдвигается вправо.

1. **Какие факторы могут влиять на эффективность сортировки**

Размерность массива, объем ОЗУ и тактовая частота процессора.

1. **В чем заключается особенность алгоритма Шелла? Опишите суть алгоритма**. Пусть дан массив из n элементов. В нем сортируются элементы, стоящие на расстоянии 0,5n элементов друг от друга, затем на расстоянии 0,25n и т.д. до тех пор, пока не получим 2 последовательности, элементы которых стоят на расстоянии 1 элемента друг от друга. Затем сортируем эту последовательность выбранным методом. Так получаем отсортированный массив.
2. **В чем заключается особенность алгоритма быстрой сортировки? Опишите суть алгоритма.** Суть алгоритма состоит в том, что происходит разделение массива на две части, но прежде чем разделить массив, мы выбираем опорный элемент и после этого делим массив, то что меньше опорного элемента уходил в левую часть, то что больше или равно вправо, и для каждой части происходит такое же разделение, массива.
3. **В чем заключается алгоритм пирамидальной сортировки? Опишите суть метода**

Из заданного массива строится пирамида, на вершину устанавливается максимальный элемент пирамиды, далее выполняется операция сортировки, вынимается вершина пирамиды, и “просеивается” элемент, вставляемый на место вершины, на последней итерации останется вершина, которая вставится в конец отсортированной последовательности.

1. **В чем заключается сортировка разделением? Опишите суть метода.**

Сначала сравниваются элементы одного разряда, а затем другого далее элементы группируются по результатам сравнения, и так до конца.

1. **В чем заключается алгоритм сортировки слиянием? Опишите суть метода**

Схож с методом быстрой сортировки т.к также происходит разделение массива, но в данном случае после разделение в каждом разделенном блоке происходит отдельная сортировка, далее две отсортированных половины соединяются в одну.

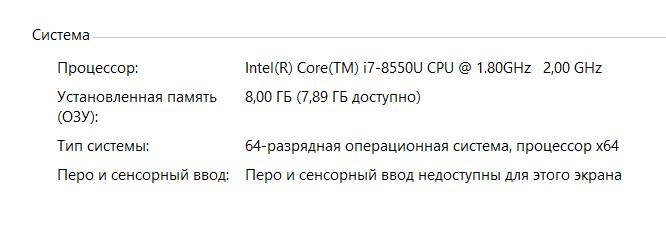
**Введение**

**Алгоритм реализован в языке С++**

**Среда разработки Microsoft Visual Studio 2019**

**Для того, чтобы засечь время, потраченное на выполнение алгоритма используется функция clock() которая находится в библиотеке <time.h>.**

**Данные времени выводится в виде миллисекунды.**

Тестирование алгоритмов сортировки проводилось на следующей конфигурации ПК:

**Диаграмма алгоритмов в лучшем случае**

В простых сортировках по графику в лучшем случае по времени самым наилучшим является методы вставки и шейкера, а самым медленным является метод пузырька. А в улучшенных сортировках наилучшим является шелл и быстрый сортировка.

Количество сравнений зависит от размера массива. В простых методах самый наилучший является вставка и шейкер, а наихудшим является пузырьковый сортировка. В улучшенных методах наилучшим является методы шелла и быстрый сортировки.

В простых методах в лучшем случае количество перестановок равен 0.

В улучшенных методах наилучший результат у метода шелла, так как количество перестановок равен 0.

**Диаграмма алгоритмов в среднем случае**

В простых методах наилучшим и быстрым является сортировка методом вставки, а самый медленный метод это пузырьковый сортировка.

В улучшенных методах самый быстрый метод конечно же быстрый сортировка, в средних случаях и при большом размерности массива наихудшим является метод шелла.

В простых методах при маленьких массивах наилучший вставка и шейкер но при больших массивах наилучшим является пузырьковый сортировка.

Наилучшим в улучшенных сортировках является быстрый сортировка, так как во всех размерах наименьшее количество сравнений.

В среднем случае в простых сортировках наилучшим является сортировка методом выбора. А количество перестановок остальных методов одинаковые.

В улучшенном методах сортировки наилучшим является быстрый сортировка и наихудшим является метод шелла.

**Диаграмма алгоритмов в худшем случае**

В простых сортировках при больших мощностях все сортировки работает медленно. Самым быстрым является метод вставки. А наихудшим является пузырьковый сортировка.

Все улучшенные сортировки работает быстро но из них наилучшим является быстрый сортировка и сортировка шелла.

В простых сортировках наилучшими является методы вставки и шейкера у них почти одинаковые результаты. Наихудшим является сортировка методом выбора, особенно в больших мощностях у метода выбором очень много сравнений.

Наилучшим в улучшенных методах является метод быстрой сортировки, а наихудшим является пирамидальная сортировка.

В худшем случае также как и в среднем случае. В простых сортировках наилучшим является сортировка методом выбора. А количество перестановок остальных методов одинаковые.

В улучшенном методах сортировки наилучшим является быстрый сортировка и наихудшим является пирамидальная сортировка.

**Простые сортировки**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Параметры |  |  | 100 |  | 1000 | 10000 | | 100000 |
| Лучший случай | Переста-новки |  |  | 0 |  | 0 | | 0 | 0 |
| Сравнения |  |  | (100-1)\*100=  9900 |  | (1000-1)\*1000=  999000 | | (10000-1)\*10000= 99990000 | (100000-1)\*100000= 9999900000 |
| Средний случай | Переста-новки |  |  | 100\*(100-1)/4 = 2475 |  | 1000\*(1000-1)/4 = 249750 | | 10000\*(10000-1)/4 = 24997500 | 100000\*(100000-1)/4 = 2499975000 |
| Сравнения |  |  | 100\*(100-1)/2 = 4950 |  | 1000\*(1000-1)/2 = 499500 | | 10000\*(10000-1)/2 = 49995000 | 100000\*(100000-1)/2 = 4999950000 |
| Худший случай | Переста-новки |  |  | 100\*(100-1)/2 = 4950 |  | 1000\*(1000-1)/2 = 499500 | | 10000\*(10000-1)/2 = 49995000 | 100000\*(100000-1)/2 = 4999950000 |
| Сравнения |  |  | 100\*(100-1)/2 =  4950 |  | 1000\*(1000-1)/2 = 499500 | | 10000\*(10000-1)/2 = 49995000 | 100000\*(100000-1)/2 = 4999950000 |

**Пузырьковая сортировка**

Количество сравнений в лучшем случае (n – 1), в среднем и худшем случаях (n\*(n – 1) / 2).

Количество перестановок в лучшем случае 0, в среднем случае (n\*(n – 1) /4), в худшем случае (n\*(n – 1)/2).

**Лучший случай:**

Перестановки во всех случаях и во всех значениях 0, и совпадают с формулой.

Сравнении: f=(n-1)\*n

100 эл. По формуле 9900, а по рассчету 9801. Не совпадают потому, что в программе считывает как f=(n-1)\*(n-1) => (100-1)\*(100-1)=9801.

1000 эл. По формуле 999000, а по рассчету 998001. Не совпадают потому, что в программе считывает как f=(n-1)\*(n-1) => (1000-1)\*(1000-1)=998001.

10000 эл. По формуле 99990000, а по рассчету 99980001. Не совпадают потому, что в программе считывает как f=(n-1)\*(n-1) => (10000-1)\*(10000-1)=99980001.

100000 эл. По формуле 9999900000, а по рассчету 9999800001. Не совпадают потому, что в программе считывает как f=(n-1)\*(n-1) => (100000-1)\*(100000-1)=9999800001.

**Аналогично и в других случаях(средний, худший).**

**Средний случай:**

Перестановки: f=n\*(n-1)/4

100 эл. По формуле 2475, а по рассчету 2262. Не совпадают потому, что программа по другому сравнивают.

1000 эл. По формуле 249750, а по рассчету 253415. Не совпадают потому что программа по другому сравнивают.

10000 эл. По формуле 24997500, а по рассчету 25167848. Не совпадают потому что программа по другому сравнивают .

100000 эл. По формуле 2499975000, а по рассчету 2506331298. Не совпадают потому что программа по другому сравнивают.

**Худший случай:**

Перестановки: f=n\*(n-1)/2

100 эл. По формуле 4950, а по рассчету 4938. Не совпадают потому что программа по другому сравнивают.

1000 эл. По формуле 499500, а по рассчету 499002. Не совпадают потому что программа по другому сравнивают.

10000 эл. По формуле 49995000, а по рассчету 49989896. Не совпадают потому что программа по другому сравнивают .

100000 эл. По формуле 4999950000, а по рассчету 704470997. Не совпадают потому что программа по другому сравнивают.

**Итак, данный метод является самым долгим в случае, когда массив не отсортирован или отсортирован частично, но очень быстрый, когда массив уже отсортирован. В случае, когда массив не отсортирован или отсортирован частично, алгоритм совершает максимально возможное (относительно других алгоритмов сортировки) количество сранений и перестановок. Это делает этот метод в целом непрактичным и нерациональным.**

**Данный алгоритм является устойчивым, его поведение естественное и стабильное.**

**Сортировка выбором**

Количество сравнений во всех случаях ((n\*n – n) / 2).

Количество перестановок в лучшем случае 3\*(n – 1), в среднем случае n\*(log 2 n), в худшем случае (n\*n/4 + 3\*(n -1)).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Параметры |  |  | 100 |  | 1000 | 10000 | 100000 |
| Лучший случай | Переста-новки |  |  | 3\*(100-1)= 297 |  | 3\*(1000-1)= 2997 | 3\*(10000-1)= 29997 | 3\*(100000-1)= 299997 |
| Сравне-ния |  |  | (100^2-100)/2= 4950 |  | (1000^2-1000)/2= 499500 | (10000^2-10000)/2= 49995000 | (100000^2-100000)/2= 4999950000 |
| Средний случай | Переста-новки |  |  | 100\* (log2(100))= 25 |  | 1000\* (log2(1000))= 250 | 10000\* (log2(10000))= 2500 | 100000\* (log2(100000))= 25000 |
| Сравне-ния |  |  | (100^2-100)/2= 4950 |  | (1000^2-1000)/2= 499500 | (10000^2-10000)/2= 49995000 | (100000^2-100000)/2= 4999950000 |
| Худший случай | Переста-новки |  |  | 100\*100/4+3\*(100-1)= 2797 |  | 1000\*1000/4+3\*(1000 -1)= 252997 | 10000\* 10000/4+3\* (10000-1)= 25029997 | 100000\* 100000/4+3\* (100000-1)= 2500299997 |
| Сравне-ния |  |  | (100^2-100)/2= 4950 |  | (1000^2-1000)/2= 499500 | (10000^2-10000)/2= 49995000 | (100000^2-100000)/2= 4999950000 |
|  |  |  |  |  |  |  |  |  |

Сравнении во всех случаях и во всех количествах элементов совпадают с формулой.

Перестановки

В лучшем случае перестановок равен 0.

В среднем случае результат должен быть в середине от n до n/2, но у меня чуть по больше от половины так как я думаю это зависит от разрядностей чисел.

В худшем случае результаты совпадают с формулой, во всех размерах массива количество перестановок равен на половину.

Хорошая особенность сортировки выбора состоит в том, что она никогда не делает больше, чем O (n) перестановок и может быть полезна, когда запись в память является дорогостоящей операцией.

**Стабильность**

Реализация по умолчанию не является стабильной.

**На месте** - не требуется дополнительного пространства

**Итак, сортировка по данному алгоритму в любом случае занимает одинаковое количество времени. Этот алгоритм является самым долгим в случае, когда массив уже отсортирован. Количество сравнений, которые осуществляет данный алгоритм, является максимально возвможным относительно сравнений, осуществляемых другими алгоритмами сортировки (за исключением сортировки вставками в случае, когда массив не отсортирован). Рационально использовать данную сортировку, когда массив не отсортирован.**

**Сортировка вставками**

Количество сравнений в лучшем случае (n – 1), в худшем случае – (0,5\*(n\*n – n)).

Количество перестановок в лучшем случае 2\*(n – 1), в среднем случае (n\*(n - 1) / 4), в худшем случае (n\*(n - 1) / 2).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Параметры |  |  | 100 |  | 1000 | 10000 | 100000 |
| Лучший случай | Переста-новки |  |  | 2\*(100-1)=198 |  | 2\*(1000-1)= 1998 | 2\*(10000-1)= 19998 | 2\*(100000-1)= 199998 |
| Сравне-ния |  |  | 100-1=99 |  | 1000-1= 999 | 10000-1= 9999 | 100000-1= 99999 |
| Средний случай | Переста-новки |  |  | 100\*(100-1)/4 = 2475 |  | 1000\*(1000-1)/4 = 249750 | 10000\*(10000-1)/4 = 24997500 | 100000\*(100000-1)/4 = 2499975000 |
| Сравне-ния |  |  | 2574 |  | 250749 | 25007499 | 2500074999 |
| Худший случай | Переста-новки |  |  | 100\*(100-1)/2 = 4950 |  | 1000\*(1000-1)/2 = 499500 | 10000\*(10000-1)/2 = 49995000 | 100000\*(100000-1)/2 = 4999950000 |
| Сравне-ния |  |  | (100^2-100)/2=4950 |  | (1000^2-1000)/2=499500 | (10000^2-10000)/2= 49995000 | (100000^2-100000)/2= 4999950000 |

**Лучший случай**:

В лучшем случае программа не делает перестановок так как второй внутренний цикл не входить. Если условия во внутреннем цикле ложь, то не делает перестановку.

Количество сравнений в программе совпадают с количеством сравнений по формуле.

**Средний случай:**

Сравнении и перестановки вполне совпадают, есть чуть разница, так как диапазон чисел разные.

**Худший случай:**

В худших случаях сортировка вставками настолько же плоха, как и пузырьковая сортировка и сортировка посредством выбора.

**Сортировка данным методом отличается от других прямых методов быстротой. В лучшем и среднем случаях данный алгоритм совершает наименьшее (относительно других прямых методов сортировки) количество сравнений, а количество перестановок в среднем и худшем случае, напротив, наибольшее.**

**Данный алгоритм является устойчивым и гибким, его поведение естественное, но данный метод нерационально использовать для сортировки больших массивов. Алгоритм эффективен, когда элементов в массиве немного, и он уже частично отсортирован.**

**Шейкерная сортировка**

Количество сравнений в лучшем случае (n – 1), в худшем случае – (0,5\*(n\*n – n)).

Количество перестановок в лучшем случае 0, в среднем случае (n\*(n - 1) / 4), в худшем случае

(n\*(n - 1) / 2).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Параметры |  | |  | | 100 | |  | | 1000 | | 10000 | | 100000 | | |
| Лучший случай | Переста-новки | |  | |  | | 0 | |  | | 0 | | 0 | | 0 |
| Сравне-ния |  | |  | | 100-1= 99 | |  | | 1000-1= 999 | | 10000-1= 9999 | | 100000-1= 99999 | | |
| Средний случай | Переста-новки |  | |  | | 100\* (100-1)/4 = 2475 | |  | | 1000\*(1000-1)/ 4 = 249750 | | 10000\*(10000-1)/ 4 = 24997500 | | 100000\*(100000-1)/4 = 2499975000 | | |
| Сравне-ния |  | |  | | 3774 | |  | | 375249 | | 37502499 | | 3750024999 | | |
| Худший случай | Переста-новки |  | |  | | 100\* (100-1)/2 = 4950 | |  | | 1000\*(1000-1)/ 2 = 499500 | | 10000\*(10000-1)/ 2 = 49995000 | | 100000\*(100000-1)/2 = 4999950000 | | |
| Сравне-ния |  | |  | | (100^2-100)/2= 4950 | |  | | (1000^2-1000)/ 2=499500 | | (10000^2-10000)/ 2= 49995000 | | (100000^2-100000)/ 2= 4999950000 | | |

**Лучшем случае:**

Сравнении совпадают с формулой.

Перестановки = 0 и в программе, и в формуле.

**Среднем случае:**

В программе количество сравнении чуть по больше чем по формуле, так как алгоритм отличается и разница в значениях и их разрядах.

Перестановки тоже чуть отличается.

**Худший случае:**

По формуле результат сравнений и перестановок должен совпадать.

В программе результаты сравнений и перестановок совпадают.

**Среди прямых методов данная сортировка по всем показателям стоит на 2 месте после сортировки вставками в лучшем и худшем случаях, она работает немного медленнее сортировки вставками. Но в худшем случае она работает медленнее, чем сортировки вставками и выбором.**

**ВЫВОД:**

**Итак, в *лучшем случае* все прямые сортировки работают быстро, кроме сортировки выбором. В *среднем случае* рационально использовать сортировку вставками, т.к. она работает быстрее остальных.**

**Сортировки пузырьком, шейкерная и вставками осуществляют одинаковое число сравнений, а сортировки пузырьком, шейкерная и вставками – одинаковое количество перестановок, которые являются наибольшими относительно других методов сортировки.**

**В *худшем случае* сортировка выбором является самой быстрой при наименьшем количестве перестановок. Остальные сортировки совершают гораздо больше перестановок. А вот количество сравнений примерно одинаково.**

**Улучшенные сортировки**

#### **Сортировка Шелла**

Среднее количество операций: **O(N7/6**), в худшем случае - порядка **O(N4/3).**

**Оценка времени выполнения**

|  |  |  |
| --- | --- | --- |
| **Лучший случай** | **Худший случай** | **Средний случай** |
| **O(N3/2)** | **O(N2)** | **O(N(logN)2 )** |

В сортировке эффективность зависит от выбора шагов. В моем случае h=n/2.

В лучшем случае шелл довольно быстро работает.

Сравнении меньше чем у быстрой и пирамидальной сортировки.

100 => 509

1000 => 8015

10000 => 120018

100000 => 1500022

В лучшем случае из улучшенных методов только метод шелла не переставляет элементов.

Перестановки=0

В среднем случае и при больших мощностях алгоритм выполняется медленно.

Сравнении тоже при больших мощностях больше чем остальные.

100 => 1061

1000 => 46754

10000 => 3936018

100000 => 393860654

В перестановках чем больше мощность тем больше перестановок.

100 => 552

1000 => 38739

10000 => 3816000

100000 => 392360632

Сортировка на месте

Алгоритм неустойчивый

Поведение неестественное

**Итак, в данной сортировке мощность зависит от выбранного шага и от размера массива. В лучшем случае данный метод не делает перестановок (относительно других улучшенных методов сортировки). В среднем случае метод шелла самый долгий и делает наибольшее количество сревнений и перестановок. В худшем случае, напротив делает наименьшее количество сравнений и перестановок чем другие методы.**

**Быстрая сортировка**

* Среднее число сравнений - **N\*LogN,**
* среднее число перестановок - **N/6\*LogN.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 100 | 1000 | 10000 | 100000 |
| Сравнении | 100\*2=200 | 1000\*3=3000 | 10000\*4=40000 | 100000\*5=500000 |
| Перестановки | 100/6\*2=33 | 1000/6\*3=500 | 10000/6\*4=6666 | 100000/6\*5=83333 |

По времени быстрая сортировка показывает наилучшее время во всех случаях и занимает первое место по времени работы.

В лучшем случае сравнении меньше чем у пирамидальной сортировки но больше чем у сортировки методом шелла.

100 => 620

1000 => 9983

10000 => 133424

100000 => 1727047

В лучшем случае быстрая сортировка делает перестановку в отличие от метода шелла

100 => 68

1000 => 867

10000 => 8857

100000 => 210668

В среднем случае очень быстро работает чем другие методы.

Сравнения во всех мощностях наилучшее.

100 => 759

1000 => 11289

10000 => 148031

100000 => 1827526

В среднем случае перестановки тоже наименьшее.

100 => 191

1000 => 2858

10000 => 36503

100000 => 505465

В худшем случае быстрый сортировка работает быстро как и метод шелла.

Сравнении меньше чем другие улучшенные методы.

100 => 578

1000 => 9490

10000 => 128426

100000 => 1677044

В худшем случае перестановок мала чем у других улучшенных методах.

100 => 118

1000 => 1366

10000 => 13855

100000 => 260661

**Итак, поведение довольно естественно, если учесть, что при частичной упорядоченности повышаются шансы разделения массива на более равные части. Алгоритм быстрой сортировки часто используется для сортировки больших массивов. Причина заключается в исключительном быстродействии.**

**При классической реализации требует в худшем случае много дополнительной памяти;**

**Неустойчив — если требуется устойчивость, приходится расширять ключ.**

**Пирамидальная сортировка**

Построение пирамиды занимает **O(n log n)** операций, причем более точная оценка дает даже **O(n)** за счет того, что реальное время выполнения **downheap** зависит от высоты уже созданной части пирамиды.Вторая фаза занимает **O(n log n)** времени: **O(n)** раз берется максимум и происходит просеивание бывшего последнего элемента. Плюсом является стабильность метода: среднее число пересылок **(n log n)/2**, и отклонения от этого значения сравнительно малы.

В лучшем случае пирамидальная сортировка больше всех делает сравнений. Сравнении у пирамидальной сортировки больше чем у сортировки методом шелла и быстрой сортировки.

100 => 1462

1000 => 23015

10000 => 214733

100000 => 3955352

В лучшем случае делает больше перестановку в отличие от других методов.

100 => 635

1000 => 9679

10000 => 131226

100000 => 1642188

В среднем случае при больших мощностях работает лучше, чем метод шелла, но не превосходит быструю сортировку.

Сравнения во всех мощностях наилучшее.

100 => 1318

1000 => 21323

10000 => 295501

100000 => 3787761

В среднем случае перестановки тоже наименьшее.

100 => 585

1000 => 9097

10000 => 124240

100000 => 1575092

В худшем случае тоже при больших мощностях работает лучше, чем метод шелла, но не превосходит быструю сортировку.

Сравнении меньше чем другие улучшенные методы.

100 => 1117

1000 => 19201

10000 => 275158

100000 => 3567307

В худшем случае перестановок мала чем у других улучшенных методах.

100 => 512

1000 => 8333

10000 => 116660

100000 => 149577

**Пирамидальная сортировка не использует дополнительной памяти.**

**Метод не является устойчивым: по ходу работы массив так "перетряхивается", что исходный порядок элементов может измениться случайным образом.**

**Поведение неестественно: частичная упорядоченность массива никак не учитывается.**

**Производительность: ~O(n\*log n).**

**Итак, пирамидальную сортировку мы должны использовать при больших мощностях. Хорошо сочетается с алгоритмами кэширования и подкачки памяти.**

**ВЫВОД:**

**Итак, в *лучшем случае* все улучшенные сортировки работают быстро. В *среднем случае* рационально использовать быструю сортировку, т.к. она работает быстрее остальных. В среднем случае и при больших мощностях наихудшим является метод шелл, т.к. оно слишком много сравнений и перестановок.**

**В *худшем случае* быстрая сортировка является самой быстрой при наименьшем количестве перестановок. Пирамидальная сортировки совершают гораздо больше сравнений и перестановок. Метод шелл при средних случаях самый наихудший из перечисленных методов.**