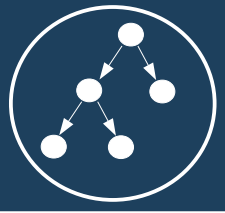


Data Structures and Algorithms

Алгоритмы. Сортировка Шелла.

Сведение о алгоритме

Алгоритм сортировки Шелла.

Сложность по времени в наихудшем случае	$O(n^2)$
Затраты памяти	$O(n)$

Некоторые сведения о алгоритме

Алгоритм сортировки Шелла является усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами.

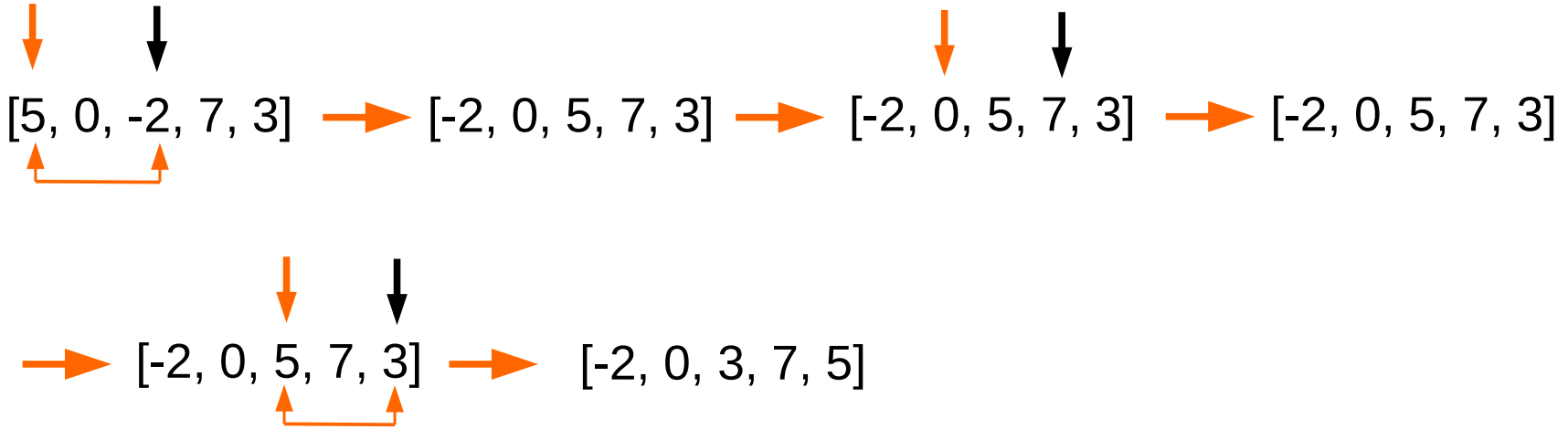
Принцип работы алгоритма

1) Выбирается начальное значение шага сортировки. К выбору шага стоит отнестись серьезно. От выбора шага зависит средняя сложность сортировки.
2) Начиная от первого элемента выполняется сравнение элементов стоящих друг от друга на расстоянии выбранного шага. Для значения элемента (в дальнейшем X) выбирается место в последовательности таких элементов, что
   $a_i \leq X \leq a_{i+h}$
   h - используемый шаг
   a_i, a_{i+h} - значение элемента на i индексе, и на i+h индексе соответственно.
3) После окончания прохода с текущим шагом, шаг уменьшают. Если текущий шаг равен 1 алгоритм заканчивают, если нет его уменьшают согласно выбранному закону его изменения и возвращаются к пункту 2.

Зависимость средней сложности алгоритма от шага

Выбор шага	Автор	Сложность
$d_i=\frac{N}{2},\ d_{i+1}=\frac{d_i}{2}$	Дональд Шелл	$O(N^2)$
$d_i=2^i-1\leqslant N, i\in\mathbb{N}$	Хиббард	$O(N^{3/2})$
$d_i=\begin{cases} 9\cdot 2^i-9\cdot 2^{\frac{i}{2}}+1, i-\textit{четное} \\ 8\cdot 2^i-6\cdot 2^{\frac{i+2}{2}}+1, i-\textit{нечетное} \end{cases}$	Седжевик	$O(N^{4/3})$
$d=2^i\cdot 3^j\leqslant N, i, j\in\mathbb{N}$	Прат	$O(N\cdot \ln(N)^2)$
$d_i=\frac{3^i-1}{2}<\frac{N}{3}$	Кнут	$O(N^{3/2})$

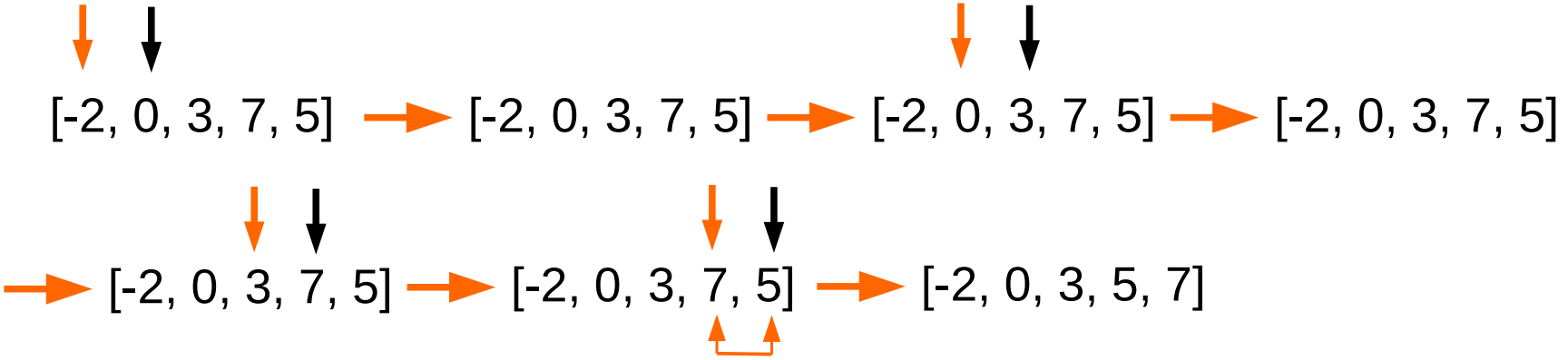
Графическая иллюстрация работы алгоритма



1) Вычисляем шаг, как целую часть от деления размера последовательности на 2

$$step = \left\lfloor \frac{5}{2} \right\rfloor = 2$$

Графическая иллюстрация работы алгоритма



2) Так как предыдущий шаг равен 1. Алгоритм заканчиваем.

Python

Реализация алгоритма на Python

Реализация алгоритма на Python

```
def shell_sort(sequince):
    n = len(sequince)
    step = n//2
    while step > 0:
        for i in range(step, n):
            j = i
            while j >= step and sequince[j] < sequince[j-step]:
                sequince[j], sequince[j-step] = sequince[j-step], sequince[j]
                j = j - step
        step = step // 2
```

Java

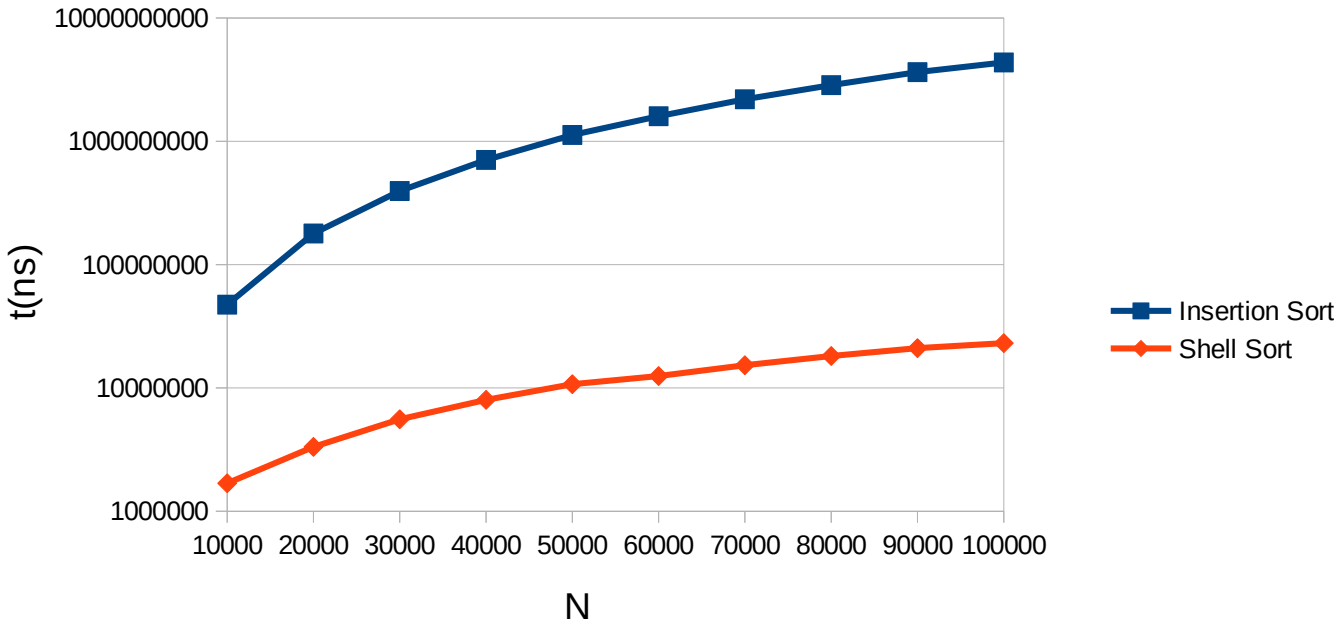
Реализация алгоритма на Java

Реализация алгоритма на Java

```
public static void shellSort(int[] array) {
    int step = array.length / 2;
    for (; step > 0;) {
        for (int i = step; i < array.length; i++) {
            for (int j = i; j >= step && array[j] < array[j - step]; j -= step) {
                int temp = array[j];
                array[j] = array[j - step];
                array[j - step] = temp;
            }
        }
        step = step / 2;
    }
}
```

Вычислительный эксперимент

Для выяснения того насколько более оптимальным является сортировка Шелла по сравнению с обычной сортировкой вставкой, был проведен вычислительный эксперимент. Было замерена зависимость время сортировки массива от его размера для обоих алгоритмов. Результат приведен на графике.



Использование других методов вычисления шага

Как было указанно ранее от выбора шага зависит оптимальность алгоритма сортировки Шелла. Для выяснения зависимости эффективности алгоритма от способа выбора шага код реализации был изменен. Теперь вычисление шага осуществляется с помощью внешнего инструмента.

Реализация алгоритма на Python

```
def shell_sort(sequince, step_implements_class):
    n = len(sequince)
    step_generator = step_implements_class(sequince)
    step = step_generator.next_step()
    while step > 0:
        for i in range(step, n):
            j = i
            while j >= step and sequince[j] < sequince[j-step]:
                sequince[j], sequince[j-step] = sequince[j-step], sequince[j]
                j = j - step
        step = step_generator.next_step()
```

Шаг предложенный Хиббардом

```
class HibbardStep:
    def __init__(self, sequince):
        self.i = 1
        while 2**self.i - 1 < len(sequince):
            self.i += 1

    def next_step(self):
        self.i -= 1
        step = 2**self.i - 1
        return step
```

Python

Шаг предложенный Седжевиком

```
class SedgewickStep:
    def __init__(self, sequince):
        self.i = 0
        number = 9*(2**self.i - 2**(self.i//2))+1
        while number < len(sequince):
            self.i += 1
            if self.i % 2 == 0:
                number = 9*(2**self.i - 2**(self.i//2))+1
            else:
                number = 8*2**self.i - 6 * 2**(self.i+1)//2 + 1

    def next_step(self):
        self.i -= 1
        if(self.i < 0):
            return 0
        if self.i % 2 == 0:
            step = 9*(2**self.i - 2**(self.i//2))+1
        else:
            step = 8*2**self.i - 6 * 2**(self.i+1)//2 + 1
        return step
```

Шаг предложенный Кнутом

```
class KnuthStep:
    def __init__(self, sequince):
        self.i = 1
        while (3**self.i - 1)//2 < len(sequince)//3:
            self.i += 1

    def next_step(self):
        step = (3**self.i - 1)//2
        self.i -= 1
        return step
```

Реализация алгоритма на Java

```
public static void shellSort(int[] array, StepGenerator stepGen) {
    int step = stepGen.nextStep();
    for (; step > 0;) {
        for (int i = step; i < array.length; i++) {
            for (int j = i; j >= step && array[j] < array[j - step]; j -= step) {
                int temp = array[j];
                array[j] = array[j - step];
                array[j - step] = temp;
            }
        }
        step = stepGen.nextStep();
    }
}
```

Вспомогательный интерфейс для работы

```
interface StepGenerator {
    public int nextStep();
}
```

Шаг предложенный Шеллом

```
class ShellStep implements StepGenerator {
    private int step;

    public ShellStep(int[] array) {
        step = array.length / 2;
    }

    @Override
    public int nextStep() {
        step = step / 2;
        return step;
    }

}
```

Шаг предложенный Хиббардом

```
class HibbardStep implements StepGenerator {
    private int i;

    public HibbardStep(int[] array) {
        for (; (int) (Math.pow(2, i) - 1) < array.length;) {
            i += 1;
        }
    }

    @Override
    public int nextStep() {
        i = i - 1;
        return (int) (Math.pow(2, i) - 1);
    }

}
```

Шаг предложенный Седжевиком

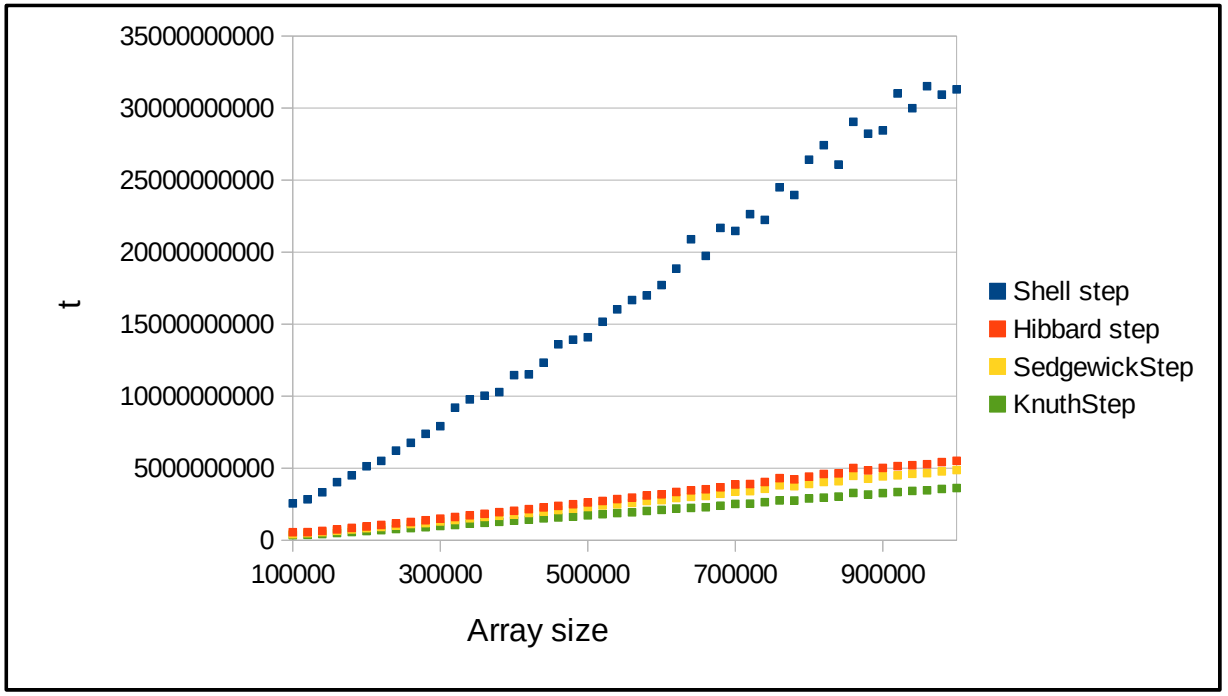
```
class SedgewickStep implements StepGenerator {
      private int i;

      public SedgewickStep(int[] array) {
            long number = (long) (9 * (Math.pow(2, i) - Math.pow(2, i / 2)) + 1);
            for (; number < array.length;) {
                  i += 1;
                  if (i % 2 == 0) {
                        number = (long) (9 * (Math.pow(2, i) - Math.pow(2, i / 2)) + 1);
                  } else {
                        number = (long) (8 * Math.pow(2, i) - 6 * Math.pow(2, (i + 1) / 2) + 1);
                  }
            }
      }
      @Override
      public int nextStep() {
            i = i - 1;
            if (i <= -1) {
                  return 0;
            }
            if (i % 2 == 0) {
                  return (int) (9 * (Math.pow(2, i) - Math.pow(2, i / 2)) + 1);
            } else {
                  return (int) (8 * Math.pow(2, i) - 6 * Math.pow(2, (i + 1) / 2) + 1);
            }
      }
}
```

Шаг предложенный Кнутом

```
class KnuthStep implements StepGenerator {
    private int i;

    public KnuthStep(int[] array) {
        for (; (Math.pow(3, i) - 1) / 2 < array.length / 3;) {
            i = i + 1;
        }
    }

    @Override
    public int nextStep() {
        int step = (int) ((Math.pow(3, i) - 1) / 2);
        i = i - 1;
        return step;
    }

}
```

Вычислительный эксперимент

Для определения зависимости скорости сортировки от алгоритма выбранного шага был проведен вычислительный эксперимент. Один и тот же массив сортировался с помощью алгоритма Шелла, при этом менялся только алгоритм выбора шага. На графике приведена зависимость времени сортировки от алгоритма выбора шага.



Список литературы

1) Роберт Седжевик, Кевин Уэйн. Алгоритмы на Java, 4-е издание.ISBN 978-5-8459-1781-2 Стр. [241-246]
2) Д. Кнут. Искусство программирования. Том 3. Сортировка и поиск, 2-е изд. ISBN 5-8459-0082-4 Стр. [85-98]