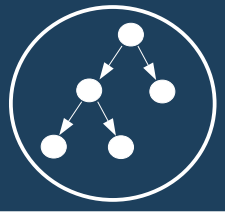


Data Structures and Algorithms

Алгоритмы.
Быстрая сортировка. Оптимизация

Модификация алгоритма быстрой сортировки

Для увеличения оптимальности алгоритма можно применить следующую модификацию. Рекурсивное разбиения последовательности до размера 1 заменить переходом в режим сортировки вставками в случае если размер подпоследовательности меньше или равен определенному значению. Оптимальным называется значения 16, 32,64. При это сама реализация остается довольно простой.

Python

Реализация алгоритма на Python

Функция для сортировки вставкой

```
def insertion_sort(sequince, start_index, end_index):
    for i in range(start_index, end_index+1):
        paste_element = sequince[i]
        while i > start_index and sequince[i-1] > paste_element:
            sequince[i] = sequince[i-1]
            i = i-1
        sequince[i] = paste_element
```

Python

Реализация алгоритма быстрой сортировки

```
def quick_sort(sequince, lo_index=None, hi_index=None):
    if lo_index is None:
        lo_index = 0
    if hi_index is None:
        hi_index = len(sequince)-1
    if hi_index-lo_index <= 32:
        insertion_sort(sequince, lo_index, hi_index)
        return None
    h = partition(sequince, lo_index, hi_index)
    quick_sort(sequince, lo_index, h-1)
    quick_sort(sequince, h+1, hi_index)
```

Java

Реализация алгоритма на Java

Java

Метод для для обмена местами элементов массива

```
public static void swap(int[] array, int i, int j) {
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}
```

Метод для для сортировки вставкой

```
public static void insertionSort(int[] array, int begin, int end) {
    for (int i = begin; i <= end; i++) {
        int pasteElement = array[i];
        int j;
        for (j = i; j > begin; j--) {
            if (array[j - 1] <= pasteElement) {
                break;
            }
            array[j] = array[j - 1];
        }
        array[j] = pasteElement;
    }
}
```

Реализация алгоритма на Java

Метод для запуска рекурсивного метода сортировки

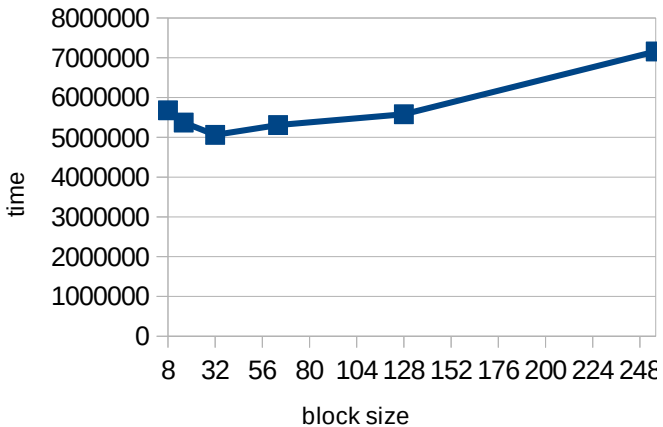
```
public static void quickSort(int[] array) {
    quickSort(array, 0, array.length - 1);
}
```

Рекурсивный метод реализующий быструю сортировку

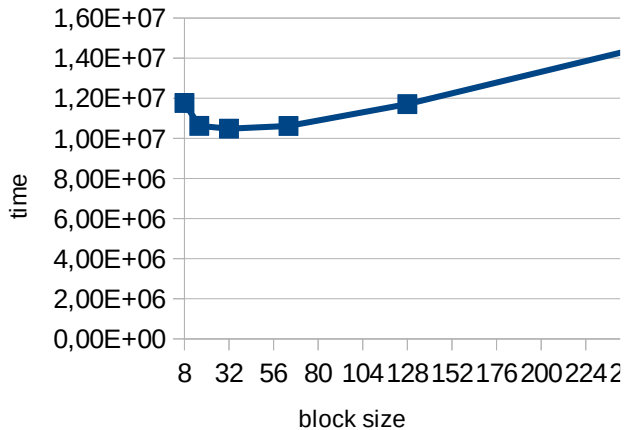
```
public static void quickSort(int[] array, int lo, int hi) {
        if (hi - lo <= 32) {
            insertionSort(array, lo, hi);
            return;
        }
        int h = breakPartition(array, lo, hi);
        quickSort(array, lo, h - 1);
        quickSort(array, h + 1, hi);
    }
```

Вычислительный эксперимент

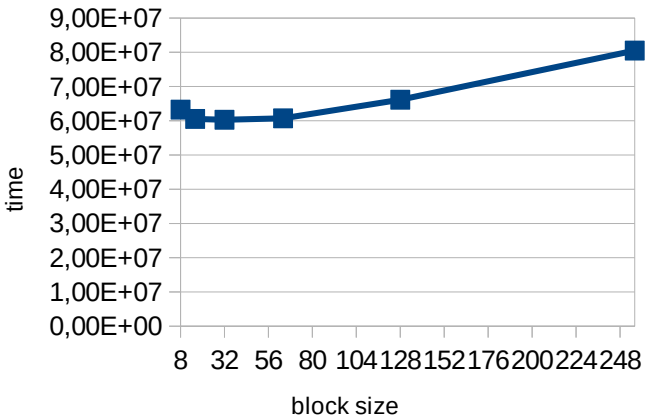
Интересным вопросом является размер подпоследовательности при котором стоит переходить на сортировку вставкой. Для определения оптимального размера выло измерено время, затраченное на сортировку массива, в зависимости от размера подпоследовательности для которого выполнялся переход к сортировке вставкой. Алгоритмы были реализованы на Java. На графике ниже вы видите результаты этих замеров.



Размер массива 50 000



Размер массива 100 000

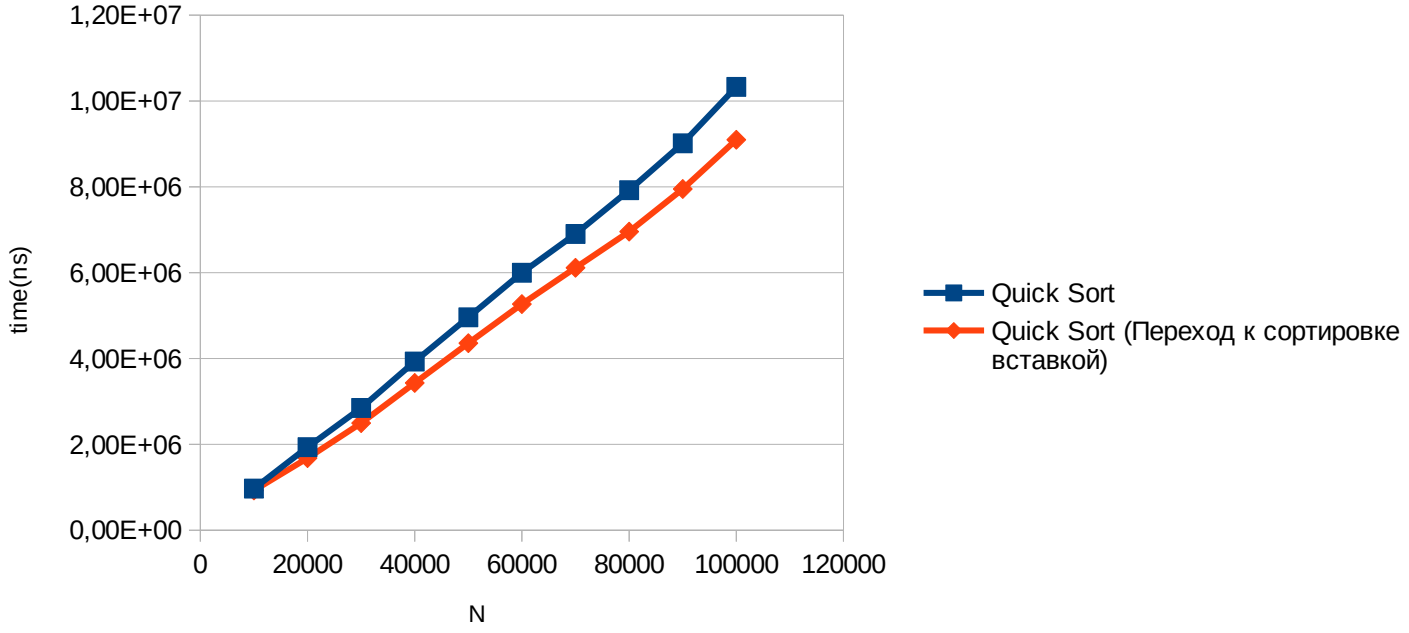


Размер массива 500 000

Из этого вычислительного эксперимента можно сделать вывод что наиболее оптимальным значением размера подпоследовательности, при которой нужно выполнять переход к сортировке вставкой, является 32.

Вычислительный эксперимент

Для проверки эффективности модифицированного алгоритма быстрой сортировки был проведен вычислительный эксперимент. Провели сравнение скоростей сортировки массивов разного размера для алгоритма быстрой сортировки и его модифицированного варианта. Оба алгоритма были реализованы на Java. На графике ниже вы можете видеть зависимость времени сортировки от размера массива.



Проблема выбора опорного элемента

Производительность алгоритма быстрой сортировки значительно зависит от выбора опорного элемента. Так если например если последовательность уже отсортирована то выбор первого элемента приведет к квадратичной сложности. Так как любая последовательность содержит отсортированные подпоследовательности, то вопрос о выборе опорного элемента довольно важен.

Одним из самых простых способов выбора опорного элемента является выбор медианы из трех элементов подпоследовательности (первого, последнего, и элемента из середины подпоследовательности).

Медиана в математической статистике — число, характеризующее выборку (например, набор чисел). Если все элементы выборки различны, то медиана — это такое число, что половина из элементов выборки больше него, а другая половина меньше.

Для выборки из 3-х элементов это среднее значение.

Python

Реализация алгоритма на Python

Функция для вычисления медианы

```
def median(sequince, lo, mid, hi):
    if sequince[lo] <= sequince[mid]:
        if sequince[mid] <= sequince[hi]:
            return mid
    else:
        if sequince[lo] <= sequince[hi]:
            return lo
    return hi
```

Функция сортировки

```
def quick_sort(sequince, lo_index=None, hi_index=None):
    if lo_index is None:
        lo_index = 0
    if hi_index is None:
        hi_index = len(sequince)-1
    if hi_index-lo_index <= 32:
        insertion_sort(sequince, lo_index, hi_index)
        return None
    mid = median(sequince, lo_index, lo_index+(hi_index-lo_index)//2, hi_index)
    sequince[lo_index], sequince[mid] = sequince[mid], sequince[lo_index]
    h = partition(sequince, lo_index, hi_index)
    quick_sort(sequince, lo_index, h-1)
    quick_sort(sequince, h+1, hi_index)
```

Java

Реализация алгоритма на Java

Метод для вычисления медианы

```
public static int median(int[] array, int lo, int mid, int hi) {
    if (array[lo] <= array[mid]) {
        if (array[mid] <= array[hi]) {
            return mid;
        }
    } else {
        if (array[lo] <= array[hi]) {
            return lo;
        }
    }
    return hi;
}
```

Изменение метода сортировки

```
public static void quickSort(int[] array, int lo, int hi) {
    if (hi - lo <= 32) {
        insertionSort(array, lo, hi);
        return;
    }
    int med = median(array, lo, lo + (hi - lo) / 2, hi);
    swap(array, lo, med);
    int h = breakPartition(array, lo, hi);
    quickSort(array, lo, h - 1);
    quickSort(array, h + 1, hi);
}
```

Девятки Тьюки

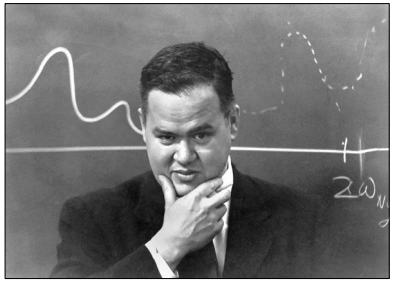
Джон Уайлдер Тьюки предложил алгоритм по улучшению выбора медианы в подпоследовательности (девятки Тьюки). Он предложил разбить подпоследовательность на три части, найти медиану в каждой из трех частей (начало, середина, конец) и после этого найти медианное значение для трех найденных медиан.

Например дана последовательность

[3, 1, 4, 4, 5, 9, 9, 8, 2]

yA = median(3, 1, 4) = 3
yB = median(4, 5, 9) = 5
yC = median(9, 8, 2) = 8

median = (3,5,8) = 5



John Wilder Tukey (1915–2000)

Python

Реализация алгоритма на Python

Python

Функция для вычисления медианы

```
def median(sequince, lo, mid, hi):
    if sequince[lo] <= sequince[mid]:
        if sequince[mid] <= sequince[hi]:
            return mid
    else:
        if sequince[lo] <= sequince[hi]:
            return lo
    return hi
```

Функция для вычисления медианы по 9 точкам

```
def tukey_median(sequince, lo, hi):
    part = len(sequince)//3
    median_a = median(sequince, lo, lo+part//2, lo+part)
    median_b = median(sequince, lo+part+1, lo+(3*part)//2+1, lo+2*part)
    median_c = median(sequince, lo+2*part+1, lo+(5*part)//2+1, hi)
    return median(sequince, median_a, median_b, median_c)
```

Функция сортировки

```
def quick_sort(sequince, lo_index=None, hi_index=None):
    if lo_index is None:
        lo_index = 0
    if hi_index is None:
        hi_index = len(sequince)-1
    if hi_index-lo_index <= 32:
        insertion_sort(sequince, lo_index, hi_index)
        return None
    mid = tukey_median(sequince, lo_index, hi_index)
    sequince[lo_index], sequince[mid] = sequince[mid], sequince[lo_index]
    h = partition(sequince, lo_index, hi_index)
    quick_sort(sequince, lo_index, h-1)
    quick_sort(sequince, h+1, hi_index)
```

Java

Реализация алгоритма на Java

Java

Метод для вычисления медианы

```
public static int median(int[] array, int lo, int mid, int hi) {
     if (array[lo] <= array[mid]) {
          if (array[mid] <= array[hi]) {
               return mid;
          }
     } else {
          if (array[lo] <= array[hi]) {
               return lo;
          }
     }
     return hi;
}
```

Метод для вычисления медианы по девяти точкам

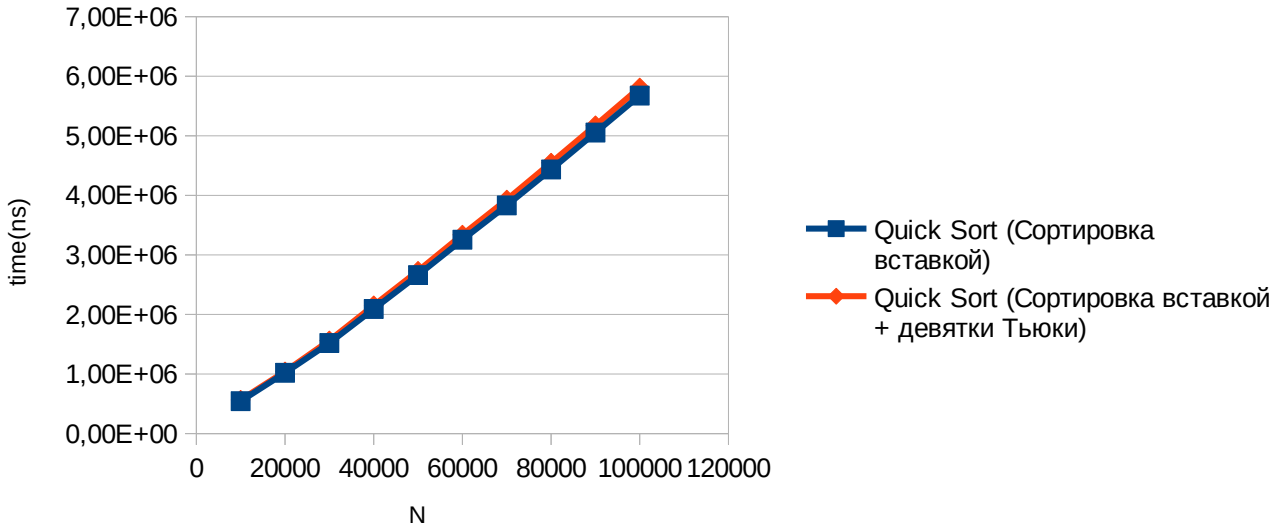
```
public static int tukeyMedian(int[] array, int lo, int hi) {
     int part = (hi - lo) / 3;
     int medianA = median(array, lo, lo + part / 2, lo + part);
     int medianB = median(array, lo + part + 1, lo + 3 * part / 2 + 1, lo + 2 * part);
     int medianC = median(array, lo + 2 * part + 1, lo + 5 * part / 2 + 1, hi);
     return median(array, medianA, medianB, medianC);
}
```

Метод сортировки

```
public static void quickSort(int[] array, int lo, int hi) {
    if (hi - lo <= 32) {
        insertionSort(array, lo, hi);
        return;
    }
    int med = tukeyMedian(array, lo, hi);
    swap(array, lo, med);
    int h = breakPartition(array, lo, hi);
    quickSort(array, lo, h - 1);
    quickSort(array, h + 1, hi);
}
```

Вычислительный эксперимент

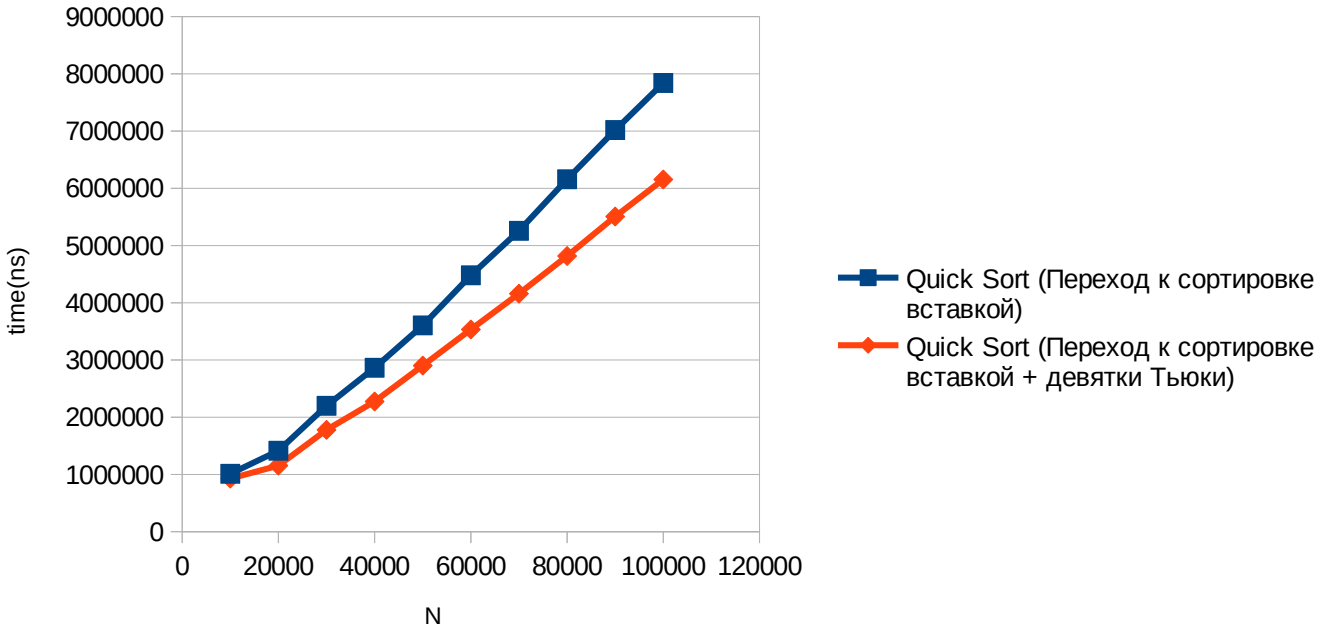
Для проверки увеличения производительности проведем вычислительный эксперимент. Сравним производительность алгоритмов быстрой сортировки с выбором первого элемента подпоследовательности и выбором элемента с использованием девяток Тьюки. Оба алгоритма реализованы на Java. Массивы заполнены случайными числами.



Как можно видеть для массивов заполненных случайными данными выигрыша в производительности нет. Так и должно быть, в этом случае выбор первого и медианного элемента равнозначен. Но последовательности совершенно случайных данных встречаются редко, обычно данные расположены в частичной упорядоченности.

Вычислительный эксперимент

Проведем вычислительный эксперимент аналогичный описанному выше, но теперь будем использовать массивы с частично упорядоченными данными (некоторые части этого массива отсортированы). На рисунке показана зависимость среднего времени сортировки от размера массива.



Список литературы

1) Д. Кнут. Искусство программирования. Том 3. «Сортировка и поиск», 2-е изд. ISBN 5-8459-0082-4
2) Роберт Седжвик, Кевин Уэйн «Алгоритмы на java 4-е издание» Пер. с англ. - М. : ООО "И.Д. Вильямс", 2013. ISBN 978-5-8459-1781-2.