

Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: Быстрая сортировка

Выполнил студент группы 3331506/80401:

Смолин И. В.

Преподаватель:

Ананьевский М. С.

« » 2021 г.

Санкт-Петербург

2021

1.1

Алгоритм быстрой сортировки применяется для упорядочивания элементов в массиве данных.

1.2

Описание работы алгоритма:

Данная сортировка построена на принципе «разделяй и властвуй», смысл которого заключается в рекурсивном разбиении задач на несколько подзадач, до тех пор, пока подзадача не станет элементарной. В использованном мной алгоритме базовым случаем является массив из одного элемента, сортировка которого не требуется, и рекурсивный спуск закончится. Теперь вопрос как дойти до этого элементарного случая:

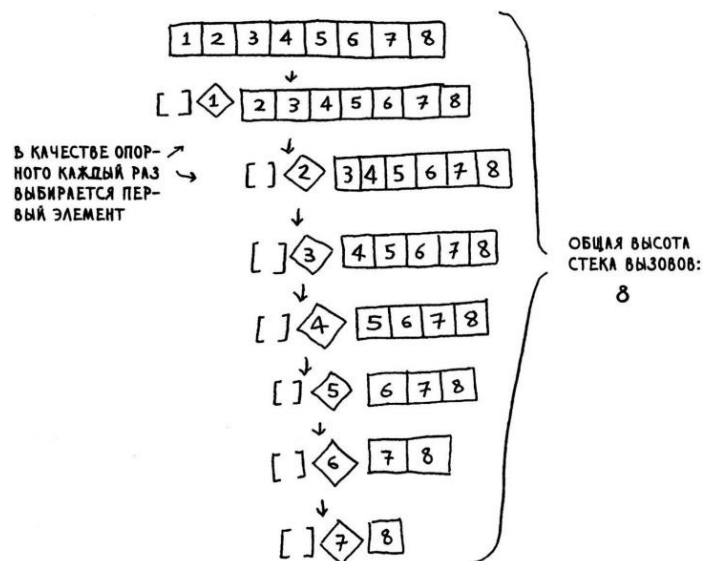
- Сначала выбирается опорный элемент и дальше элементы в массиве, который мы сортируем, идут в два подмассива, в зависимости от того, они больше или меньше, чем опорный элемент
- Затем для подмассивов применяется та же операция с выбором опорного элемента и разбиением на два подмассива и дальше все это рекурсивно повторяется до тех пор, пока мы не дойдем до базового случая.

1.3

Описание скорости работы алгоритма, в зависимости от размера входных данных:

Главным вопросом и ключом к эффективности алгоритма является выбор опорного элемента.

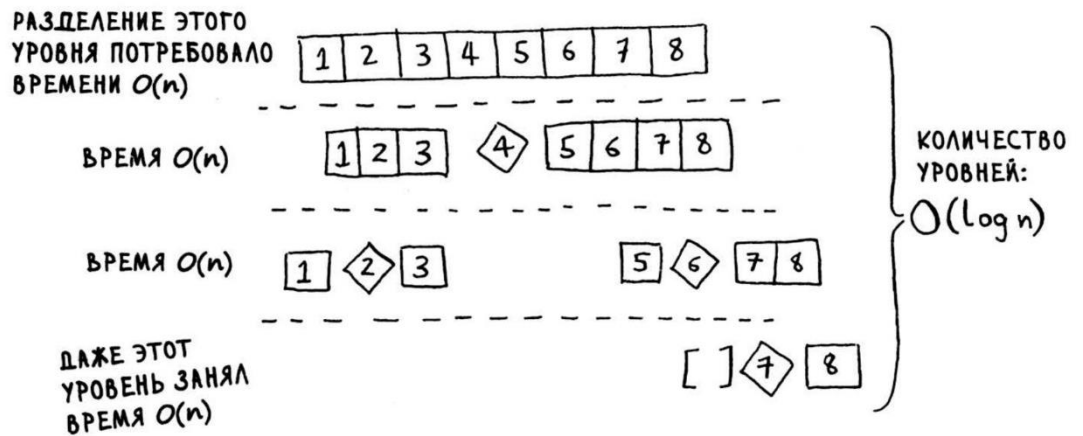
Рассмотрим сначала случай, когда мы в качестве опорного элемента используем крайний элемент в массиве:



В данном случае видно, что массив не разбивается на две половины, и один из подмассивов всегда пуст, так что стек вызовов получается длинным.

Перебор всех элементов на одном уровне выполняется за $O(n)$, при этом так как мы берем всегда крайний элемент как опорный, то высота стека вызовов будет иметь длину $O(n)$, таким образом общее время сортировки будет $O(n^2)$. В таком случае это не будет быстрее стандартных алгоритмов сортировки. А поэтому будем считать этот вариант худшим.

А что, если брать средний элемент в качестве опорного для данного массива?



В данном случае стек вызовов получился намного короче.

Первый пример описывает худший сценарий, а второй – лучший.

В худшем случае размер стека вызовов описывается как $O(n)$. В лучшем случае он составит $O(\log n)$.

На каждом уровне вложенности мы обращаемся с n элементами, поэтому конечная скорость выполнения алгоритма в худшем случае будет $O(n^2)$, а в лучшем $O(n \log n)$.

Если выбирать опорным элементом случайным элемент в массиве, то лучший случай будет и средним, и сортировка в среднем будет завершаться за $O(\log n)$.

Ниже представлен график среднего времени выполнения сортировки массива, в зависимости от его размера:

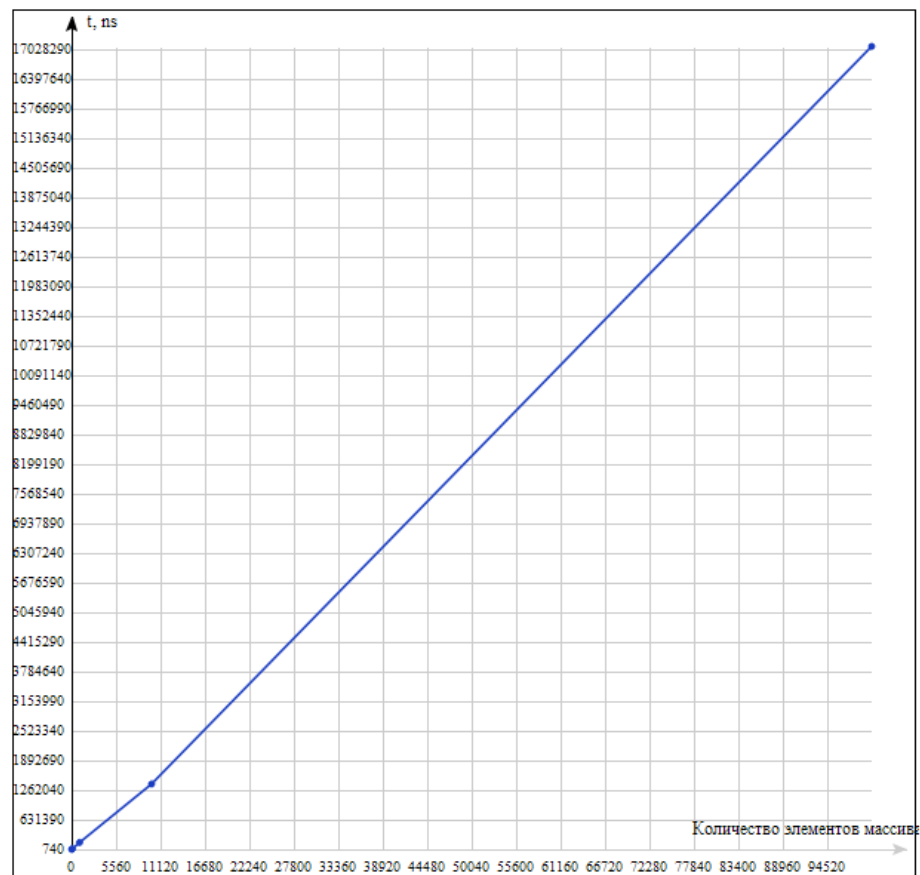


Рисунок 1

Список литературы

1. Бхаргава А., Грокам Алгоритмы / Питер, 2017.

Приложение

```
#include <chrono>
#include <stdint.h>
#include <stdlib.h>
#include <iostream>
#include <algorithm>

using namespace std;

void sort( int32_t * arr, uint32_t size )
{
    if (size > 1)
    {
        int32_t left = 0;
        int32_t right = size - 1;
        int32_t pivot = arr[rand() %size];

        while (left <= right)
        {
            while (arr[left] < pivot)
                left++;
            while (arr[right] > pivot)
                right--;
            if (left <= right)
            {
                swap(arr[left], arr[right]);
                left++;
                right--;
            }
        }

        if (right > 0)
            sort(arr, right + 1);

        if (right < size - 1)
            sort(&arr[right + 1], size - 1 - right);
    }
}

int main()
{
    int32_t ar[1000000]= {0};
    for (int & i : ar)
    {
        i = rand();
    }
    auto begin = std::chrono::steady_clock::now();
    sort(ar, sizeof(ar)/sizeof(int32_t));
    auto end = chrono::steady_clock::now();
    auto elapsed_ms = chrono::duration_cast<std::chrono::nanoseconds>(end -
begin);
    std::cout << "The time: " << elapsed_ms.count() << " ns\n";
}
```