Министерство науки и высшего образования РФ ФГБОУ ВО “Удмуртский государственный университет” институт математики, информационных технологий и физики кафедра теоретических основ информатики

Отчет по теме

«Анализ сложности алгоритмов сортировки»

Выполнил: студент
группы ОБ-02.03.02.01-41
Полянских Сергей Владимирович

Ижевск, 2023

1 Описание работы алгоритма

Быстрая сортировка является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как «Пузырьковая сортировка» и «Шейкерная сортировка»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы (таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов).

Общая идея алгоритма состоит в следующем:

- Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.
- Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
- Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

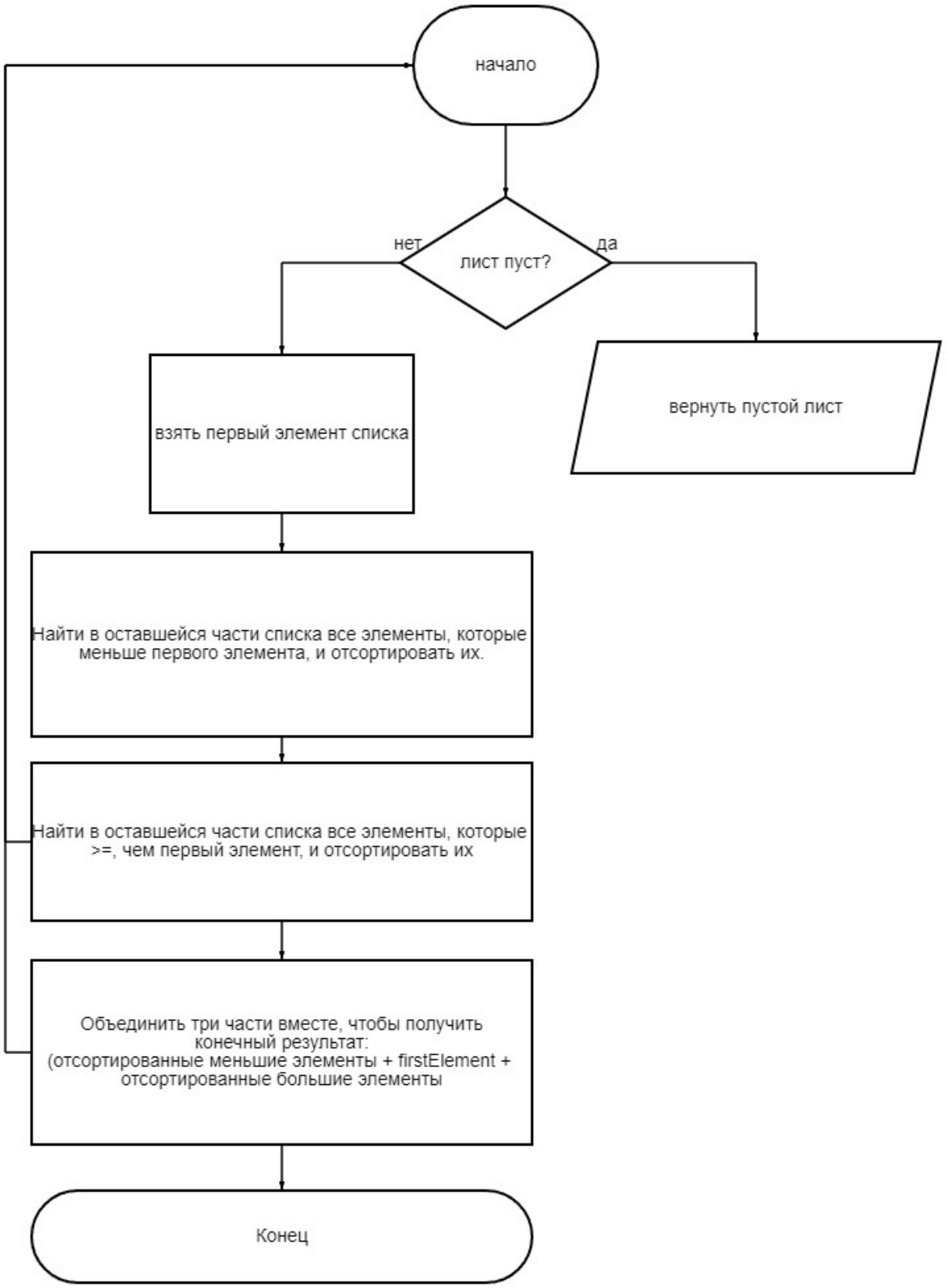

Рис.1. Блок схема алгоритма

2 Временная сложность работы алгоритма

Алгоритм быстрой сортировки имеет наилучший случай временной сложности *O(n log n)*, когда входные данные уже отсортированы или почти отсортированы. В этом случае выбор опорного элемента происходит оптимально, и каждый раз массив делится на две примерно равные части.

Однако, наихудший случай возникает, когда выбирается опорный элемент, который является минимальным или максимальным элементом в массиве. В этом случае массив не делится на две примерно равные части, а одна из них будет пустой. Это приводит к тому, что алгоритм будет работать за *O(n^2)* времени. Для избежания такого случая можно использовать различные стратегии выбора опорного элемента, например, выбирать его случайным образом.

Средняя временная сложность быстрой сортировки также составляет *O(n log n)*, но точное время выполнения зависит от выбранной стратегии выбора опорного элемента и расположения элементов в массиве.

В целом, время выполнения алгоритма быстрой сортировки зависит от выбранной стратегии выбора опорного элемента и расположения элементов в массиве. Однако, в среднем случае, алгоритм быстрой сортировки является одним из самых быстрых и эффективных алгоритмов сортировки.

3 Реализация и тестирование производительности работы алгоритма

Ниже приведена реализация алгоритма быстрой сортировки на языке F#.

Листинг 1 - Быстрая сортировка:

```
let rec quicksort list =
  match list with
  | [] -> []
  | firstElem::otherElements ->
      let smallerElements =
         otherElements
         |> List.filter (fun e -> e < firstElem)
         |> quicksort
      let largerElements =
         otherElements
         |> List.filter (fun e -> e >= firstElem)
         |> quicksort

      List.concat [smallerElements; [firstElem]; largerElements]
```

Измерение скорости работы алгоритма было проведено с помощью библиотеки Benchmark.NET. Она необходима, чтобы получить наиболее точные результаты.

Алгоритм сортировки будет запущен в цикле *3000* раз. Количество необходимых запусков теста для наиболее точных результатов определится “на лету”, в ходе выполнения программы.

Код для измерения производительности работы алгоритмы представлен ниже.

Листинг 2 - Тестирование производительности:

```
type Benchmarks() =

    [<Params(100, 200, 300, 400, 500)>]
    member val size = 0 with get, set

    [<Benchmark(Baseline = true)>]
    member this.Sort() =
        for _ in 1..3000 do
            generateList this.size |> quicksort |> ignore

BenchmarkRunner.Run<Benchmarks>() |> ignore

let generateList length =
    let a, b = -999, 999
    let rnd = Random()
    List.init length (fun _ -> rnd.Next(a, b))
```

Результаты тестирования представлены на графике и таблице.

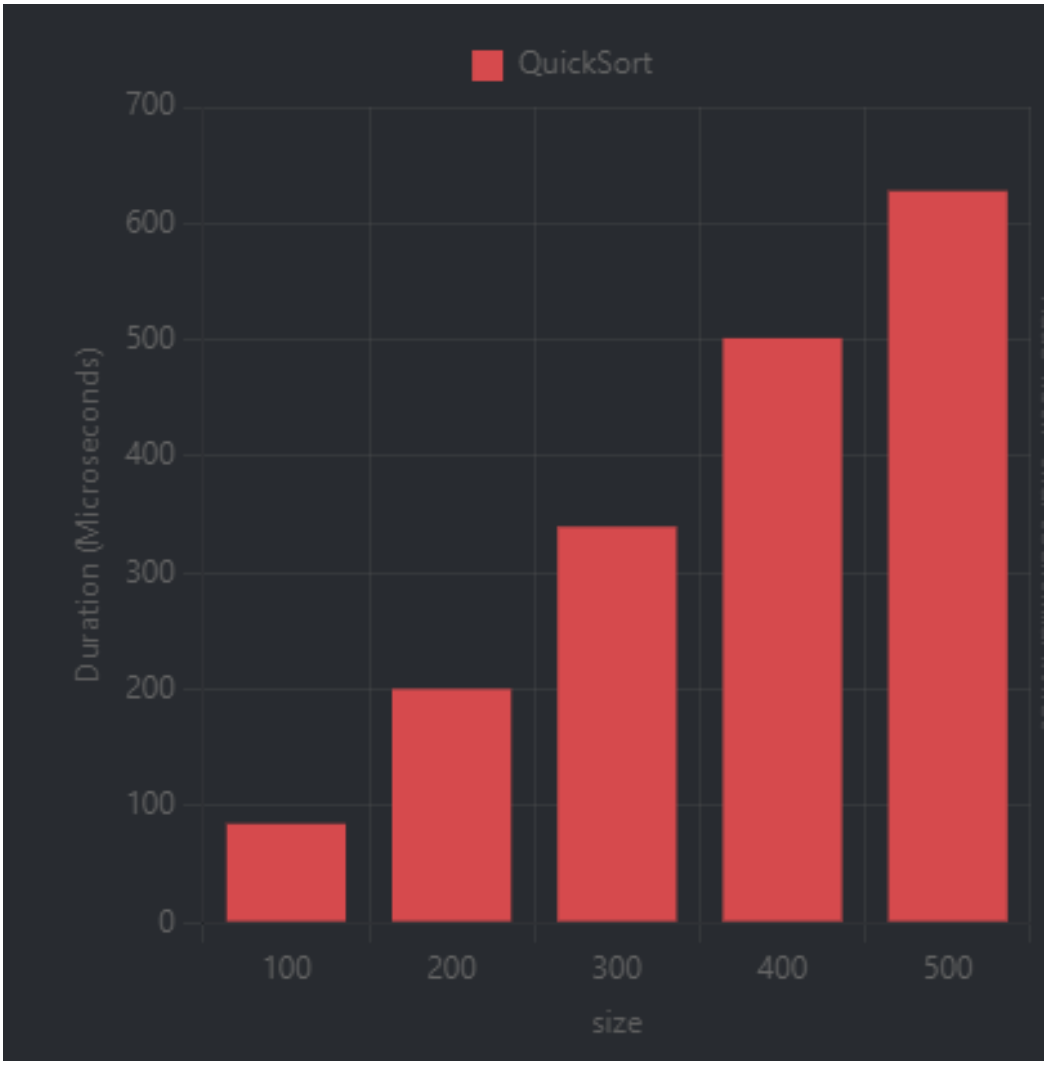

Рис 2. Результаты тестирования

Таблица 1. Результаты тестирования

Размер (size)	Среднее время	Наилучшее время	Наихудшее время
100	84.859 us	84.074 us	86.381 us
200	200.447 us	198.048 us	203.316 us
300	339.719 us	322.809 us	374.567 us
400	501.548 us	475.692 us	529.528 us
500	627.843 us	616.797 us	643.194 us

4 Вывод

В результате исследования и тестирования алгоритма быстрой сортировки было выявлено, что его асимптотическая сложность составляет O(n log n) в среднем и в наихудшем случаях. Однако, при правильном выборе стратегии выбора опорного элемента и оптимизациях, время выполнения алгоритма может быть улучшено.

Также было обнаружено, что в лучшем случае, когда массив уже отсортирован, алгоритм быстрой сортировки работает за O(n) времени, что является очень эффективным.

В целом, алгоритм быстрой сортировки является одним из самых быстрых и эффективных алгоритмов сортировки в среднем случае. Однако, при работе с массивами, которые уже отсортированы или имеют большое количество повторяющихся элементов, стоит использовать другие алгоритмы сортировки, чтобы избежать наихудшего случая быстрой сортировки.