САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «Алгоритмы и структуры данных»

Тема: Быстрая сортировка, сортировки за линейное время

Вариант 3

Выполнил:
Брылев М.И.
К3126

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту

Задача №1. Улучшение Quicksort

1. Используя псевдокод процедуры Randomized - QuickSort, а так же Partition
из презентации к Лекции 3 (страницы 8 и 12), напишите программу быстрой
сортировки на Python и проверьте ее, создав несколько рандомных массивов,
подходящих под параметры:
• Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \leq n \leq 10^4$) — число элементов в массиве.
Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9.
• Формат выходного файла (output.txt). Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.
• Ограничение по времени. 2 сек.
• Ограничение по памяти. 256 мб.
• Для проверки можно выбрать наихудший случай, когда сортируется массив рамера 103, 104 , 105 чисел порядка 10^9 , отсортированных в обратном порядке; наилучший, когда массив уже отсортирван, и средний - случайный. Сравните на данных сетах Randomized-QuickSort и простой QuickSort. (А также есть Median-QuickSort, см. задание 10.2;
и Tail-Recursive-QuickSort, см. Кормен. 2013, стр. 217)
2. Основное задание. Цель задачи - переделать данную реализацию рандо-
мизированного алгоритма быстрой сортировки, чтобы она работала быстро
даже с последовательностями, содержащими много одинаковых элементов.
Чтобы заставить алгоритм быстрой сортировки эффективно обрабатывать
последовательности с несколькими уникальными элементами, нужно заме-
нить двухстороннее разделение на трехстороннее (смотри в Лекции 3 слайд
17). То есть ваша новая процедура разделения должна разбить массив на три
части:

• A[k] < x для всех l + 1 ≤ k ≤ m1 − 1
• A[k] = x для всех m1 ≤ k ≤ m2
• A[k] > x для всех m2 + 1 ≤ k ≤ r
• Формат входного и выходного файла аналогичен п.1.
• Аналогично п.1 этого задания сравните Randomized-QuickSort +c Partition и ее с Partition3 на сетах случайных данных, в которых содержатся всего несколько уникальных элементов при n = 103 , 104 , 105 .
Что быстрее, Randomized-QuickSort +c Partition3 или Merge-Sort?
• Пример:
input.txt
output.txt
5
22239
23922

Листинг кода. (именно листинг, а не скрины)

```
function quickSort (arr, left = 0, right = arr.length - 1) {
 if (arr.length > 1) {
   const position = partition(arr, left, right)
   if (left < position - 1) quickSort(arr, left, position - 1)
   if (position < right) quickSort(arr, position, right)
 }
 return arr
}

function partition (arr, left, right) {
  const pivot = arr[Math.floor(Math.random() * (right - left + 1) +
left)]

 while (left <= right) {
   while (arr[left] < pivot) {
     left += 1
   }
   while (arr[right] > pivot) {
     right -= 1
   }
   if (left <= right) {
     swap(arr, left, right)
     left += 1
     right -= 1
```

```
    }
  }
  return left
}

function swap (arr, left, right) {
  const temp = arr[left]
  arr[left] = arr[right]
  arr[right] = temp
}

import { readFile, writeFile } from 'fs'

readFile('./input.txt', 'utf8', (err, data) => {
  const input = data.match(/[^\r\n]+/g)
  const output = quickSort(input[1].split(' ').map(Number))
  writeFile('./output.txt', output.join(' '), () => {})
})
```

Реализация quicksort

```
[maksim@maksim-pc Задача 1]$ cat input.txt
5
2 3 9 2 2[maksim@maksim-pc Задача 1]$ node main1.mjs
[maksim@maksim-pc Задача 1]$ cat output.txt
2 2 2 3 9[maksim@maksim-pc Задача 1]$ ~
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	00.218s	60,0MB
Пример из задачи	00.298s	61,4 MB
Верхняя граница диапазона значений входных данных из текста задачи	00.312s	66,0MB

Вывод по задаче: Quicksort может плохо себя показывать на массивах, в которых много повторяющихся элементах. Важно помнить об этом и по возможности улучшать алгоритм.

Задача №4. Точки и отрезки

Допустим, вы организовываете онлайн-лотерею. Для участия нужно сделать
ставку на одно целое число. При этом у вас есть несколько интервалов после-
довательных целых чисел. В этом случае выигрыш участника пропорционален
количеству интервалов, содержащих номер участника, минус количество интер-
валов, которые его не содержат. (В нашем случае для начала - подсчет только
количества интервалов, содержащих номер участника). Вам нужен эффективный
алгоритм для расчета выигрышей для всех участников. Наивный способ сделать
это - просто просканировать для всех участников список всех интевалов. Однако
ваша лотерея очень популярна: у вас тысячи участников и тысячи интервалов. По
этой причине вы не можете позволить себе медленный наивный алгоритм.
4
• Цель. Вам дается набор точек и набор отрезков. Цель состоит в том, чтобы
вычислить для каждой точки количество отрезков, содержащих эту точку.
• Формат входного файла (input.txt). Первая строка содержит два неот-
рицательных целых числа s и p. s - количество отрезков, p - количество
точек. Следующие s строк содержат 2 целых числа ai , bi, которые определя-
ют i-ый отрезок [ai , bi]. Последняя строка определяет p целых чисел - точек
x1 , x2, ..., xp. Ограничения: $1 \le s, p \le 50000$; $-10^8 \le a_i \le b_i \le 10^8$ для
всех $0 \le i < s$; $-10^8 \le x_i \le 10^8$ для всех $0 \le j < p$.
• Формат выходного файла (output.txt). Выведите p неотрицательных целых
чисел k0 , k1 ..., kp−1 , где ki - это число отрезков, которые содержат xi. То
есть,
$k_i = |j : a_j \le x_i \le b_j|$.

• Пример 1.
input.txt
 output.txt
23
 100
05
7 10
1 6 11
Здесь, у нас есть 2 отрезка и 2 точки. Первая точка принадлежит интервалу
[0, 5], остальные точки не принадлежат ни одному из данных интервалов.
• Пример 2.
input.txt
13
-10 10
-100 100 0
output.txt
001
• Пример 3.
input.txt
32
05
-3 2
7 10
16
output.txt
20

```
function quickSort (arr, left = 0, right = arr.length - 1, predicate) {
 if (arr.length > 1) {
   const position = partition(arr, left, right, predicate)
     if (left < position - 1) quickSort(arr, left, position - 1,
predicate)
   if (position < right) quickSort(arr, position, right, predicate)
 }
 return arr
}

function partition (arr, left, right, predicate) {
  const pivot = arr[Math.floor(Math.random() * (right - left + 1) +
left)]
```

```
 // console.log(arr, left, right, pivot)
 while (left <= right) {
   // while (arr[left] < pivot) {
   while (!predicate(arr[left], pivot)) {
     left += 1
   }
   //while (arr[right] > pivot) {
   while (predicate(arr[right], pivot)) {
     right -= 1
   }
   if (left <= right) {
     swap(arr, left, right)
     left += 1
     right -= 1
   }
 }
 return left
}

function swap (arr, left, right) {
 const temp = arr[left]
 arr[left] = arr[right]
 arr[right] = temp
}

function findPoints(segments, points) {
 //return points.map((p) => segments.filter((s) => s[0] <= p && s[1] >=
p).length)
 const sorted = quickSort(segments, undefined, undefined, (a, b) => {
   if (a === undefined) return true
   if (b == undefined) return false
   return a[0] > b[0] && a[1] > b[1]
 })
 return points.map((p) => {
   const first = sorted[0]
   const last = sorted[sorted.length - 1]
   if (first[0] > p) return 0
   if (last[1] < p) return 0
    const i = sorted.findIndex((value) => value[0] <= p && value[1] >=
p)
   if (i === -1) return 0
   const workslice = sorted.slice(i, sorted.length)
```

```
    return workslice.filter((value) => value[0] <= p && value[1] >=
p).length
 })
}

import { readFile, writeFile } from 'fs'

readFile('./input.txt', 'utf8', (err, data) => {
 const input = data.match(/[^\r\n]+/g)
 const segments = input.slice(1, input.length - 1).map((x) => {
   const s = x.split(' ')
   return [Number(s[0]), Number(s[1])]
 })
 const points = input[input.length - 1].split(' ').map(Number)
 const result = findPoints(segments, points)
 writeFile('./output.txt', result.join(' '), () => {})
})
```

Для более эффективного нахождения точек мы сортируем все отрезки по первой координате. Для каждой точки мы проверяем, находится ли она вообще хотя бы в рабочем районе (координата больше чем первая координата самого дальнего слева отрезка и координата меньше чем последняя координата самого дальнего справа отрезка).

```
[maksim@maksim-pc Задача 4]$ cat input.txt
1 3
-10 10
-100 100 0[maksim@maksim-pc Задача 4]$ node main.mjs
[maksim@maksim-pc Задача 4]$ cat output.txt
0 0 1[maksim@maksim-pc Задача 4]$
```

```
[maksim@maksim-pc Задача 4]$ cat input.txt
2 3
0 5
7 10
1 6 11[maksim@maksim-pc Задача 4]$ node main.mjs
[maksim@maksim-pc Задача 4]$ cat output.txt
1 0 0[maksim@maksim-pc Задача 4]$
```

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)

Проверка задачи на (openedu, acmp и тд при наличии в задаче). (скрин)

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	00.192s	44,8MB
Пример из задачи	00.184s	46,8MB
Пример из задачи	00.201s	46,8MB
Верхняя граница диапазона значений входных данных из текста задачи	00.218s	57,0MB

Вывод по задаче: часто можно обойтись без квадратичного перебора “всех элементов на все элементы”, если правильно подойти к созданию алгоритма по работе с данными.

Задача №7. Цифровая сортировка

Дано n строк, выведите их порядок после k фаз цифровой сортировки.
• Формат входного файла (input.txt). В первой строке входного файла содержатся числа n - число строк, m - их длина и k - число фаз цифровой сортировки ($1 \le n \le 106$, $1 \le k \le m \le 106$, $n \cdot m \le 5 \cdot 107$). Далее находится описание строк, но в нетривиальном формате. Так, i-ая строка ($1 \le i \le n$) записана в i-ых символах второй, ..., (m + 1)-ой строк входного файла. Иными словами, строки написаны по вертикали. Это сделано
специально, чтобы сортировка занимала меньше времени.
7
Строки состоят из строчных латинских букв: от символа "a"до символа "z"включительно. В таблице символов ASCII все эти буквы располагаются
подряд и в алфавитном порядке, код буквы "a"равен 97, код буквы "z"равен
122.
• Формат выходного файла (output.txt). Выведите номера строк в том порядке, в котором они будут после k фаз цифровой сортировки.
• Ограничение по времени. 3 сек.
• Ограничение по памяти. 256 мб.
•Примеры:
input.txt
331
bab
bba
baa
332
bab
bba
baa
333
bab
bba
baa
output.txt
231

321
231
• Примечание. Во всех примерах входных данных даны следующие строки:
– «bbb», имеющая индекс 1;
– «aba», имеющая индекс 2;
– «baa», имеющая индекс 3.
Разберем первый пример. Первая фаза цифровой сортировки отсортирует
строки по последнему символу, таким образом, первой строкой окажется «aba» (индекс 2), затем «baa» (индекс 3), затем «bbb» (индекс 1). Таким образом, ответ равен «2 3 1».

Листинг кода. (именно листинг, а не скрины)

```
function mergeSort (arr, predicate) {
 if (arr.length === 1) {
   return arr
 }
 const middle = Math.floor(arr.length / 2)
 const left = arr.slice(0, middle)
 const right = arr.slice(middle)
 return merge(
   mergeSort(left, predicate),
   mergeSort(right, predicate),
   predicate
 )
}


function merge (left, right, predicate) {
 const result = []
 let indexLeft = 0
 let indexRight = 0

 while (indexLeft < left.length && indexRight < right.length) {
   if (predicate(left[indexLeft], right[indexRight])) {
     result.push(left[indexLeft])
     indexLeft++
   } else {
     result.push(right[indexRight])
     indexRight++
```

```
    }
  }

                                                                  return
result.concat(left.slice(indexLeft)).concat(right.slice(indexRight))
}

function transpose(x) {
  const arr = []
  for (const [k] of Object.entries(x[0])) {
    arr[k] = ''
    for (let i = 0; i < x.length; i += 1) {
      arr[k] += x[i][k]
    }
  }
  return arr
}

function compare(a, b, ai, bi) {
  if (a === undefined) return true
  if (b === undefined) return false
  if (a !== b) {
    for (const [k, v] of Object.entries(a)) {
      const bv = b[k]
      if (bv === undefined) return 1
      if (v === bv) continue
      return v < bv
    }
    return true
  }
  return ai < bi
}

function sort(arr, k) {
  const x = transpose(arr)
  const b = mergeSort(x.map((e, i) => [e, i + 1]), (a, b) => {
    if (a === undefined) return true
    if (a === b) return false
    const index = a[0].length - k
    const copy_a = a[0].slice(index)
    const copy_b = b[0].slice(index)
    return compare(copy_a, copy_b, a[1], b[1])
  })
```

```
 return b.map((w) => x.indexOf(w[0]) + 1)
}

import { readFile, writeFile } from 'fs'

readFile('./input.txt', 'utf8', (err, data) => {
 const input = data.match(/[^\r\n]+/g)
 const [n, m, k] = input[0].split(' ').map(Number)
 const x = input.slice(1, input.length)
 const output = sort(x, k)
 writeFile('./output.txt', output.join(' '), () => {})
})
```

Для более простой работы с строками следует ее перевернуть из вертикальной в горизонтальную форму. Потом просто применяем сортировку.

```
[maksim@maksim-pc Задача 9]$ cat input.txt
3 3 1
bab
bba
baa[maksim@maksim-pc Задача 9]$ node main.mjs
[maksim@maksim-pc Задача 9]$ cat output.txt
2 3 1[maksim@maksim-pc Задача 9]$
```

```
[maksim@maksim-pc Задача 9]$ cat input.txt
3 3 2
bab
bba
baa[maksim@maksim-pc Задача 9]$ node main.mjs
[maksim@maksim-pc Задача 9]$ cat output.txt
3 2 1[maksim@maksim-pc Задача 9]$
```

```
[maksim@maksim-pc Задача 9]$ cat input.txt
3 3 3
bab
bba
baa[maksim@maksim-pc Задача 9]$ node main.mjs
[maksim@maksim-pc Задача 9]$ cat output.txt
2 3 1[maksim@maksim-pc Задача 9]$
```

Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)
Проверка задачи на (openedu, acmp и тд при наличии в задаче). (скрин)

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	00.297s	61,1MB
Пример из задачи	00.300s	60,8MB
Пример из задачи	00.290s	61,2MB
Верхняя граница диапазона значений входных данных из текста задачи	00.302s	63,2MB

Вывод по задаче: вертикальная запись строк может помочь в сортировке строк.

Вывод

Сортировки за линейное время являются самыми эффективными сортировками. Важно выстраивать модель данных для того чтобы была возможность проводить сортировки за линейное время.