Сортировки за линейное время

6 октября 2019 г.

Понятие сортировки

- **Сортировка** — это алгоритм для упорядочивания элементов в списке.

Понятие сортировки

- **Сортировка** — это алгоритм для упорядочивания элементов в списке.
- **Время работы**. Минимальное время работы алгоритма на каком-либо наборе.

Понятие сортировки

- **Сортировка** — это алгоритм для упорядочивания элементов в списке.
- **Время работы**. Минимальное время работы алгоритма на каком-либо наборе.
- **Память**. Параметр сортировки, показывающий, сколько дополнительной памяти требуется алгоритму.

Понятие сортировки

- **Сортировка** — это алгоритм для упорядочивания элементов в списке.
- **Время работы**. Минимальное время работы алгоритма на каком-либо наборе.
- **Память**. Параметр сортировки, показывающий, сколько дополнительной памяти требуется алгоритму.
- **Устойчивость**. Устойчивой сортировкой называется сортировка, не меняющая порядка объектов с одинаковыми значениями.

Понятие сложности алгоритма

Понятие сложности алгоритма

- Под сложностью алгоритма понимается зависимость объёма работы, от размера входных данных.

Понятие сложности алгоритма

- Под сложностью алгоритма понимается зависимость объёма работы, от размера входных данных.
- фраза «сложность алгоритма есть O(f(n))» означает, что с увеличением параметра n, характеризующего количество входной информации алгоритма, время работы алгоритма будет возрастать не быстрее, чем некоторая константа, умноженная на f(n).

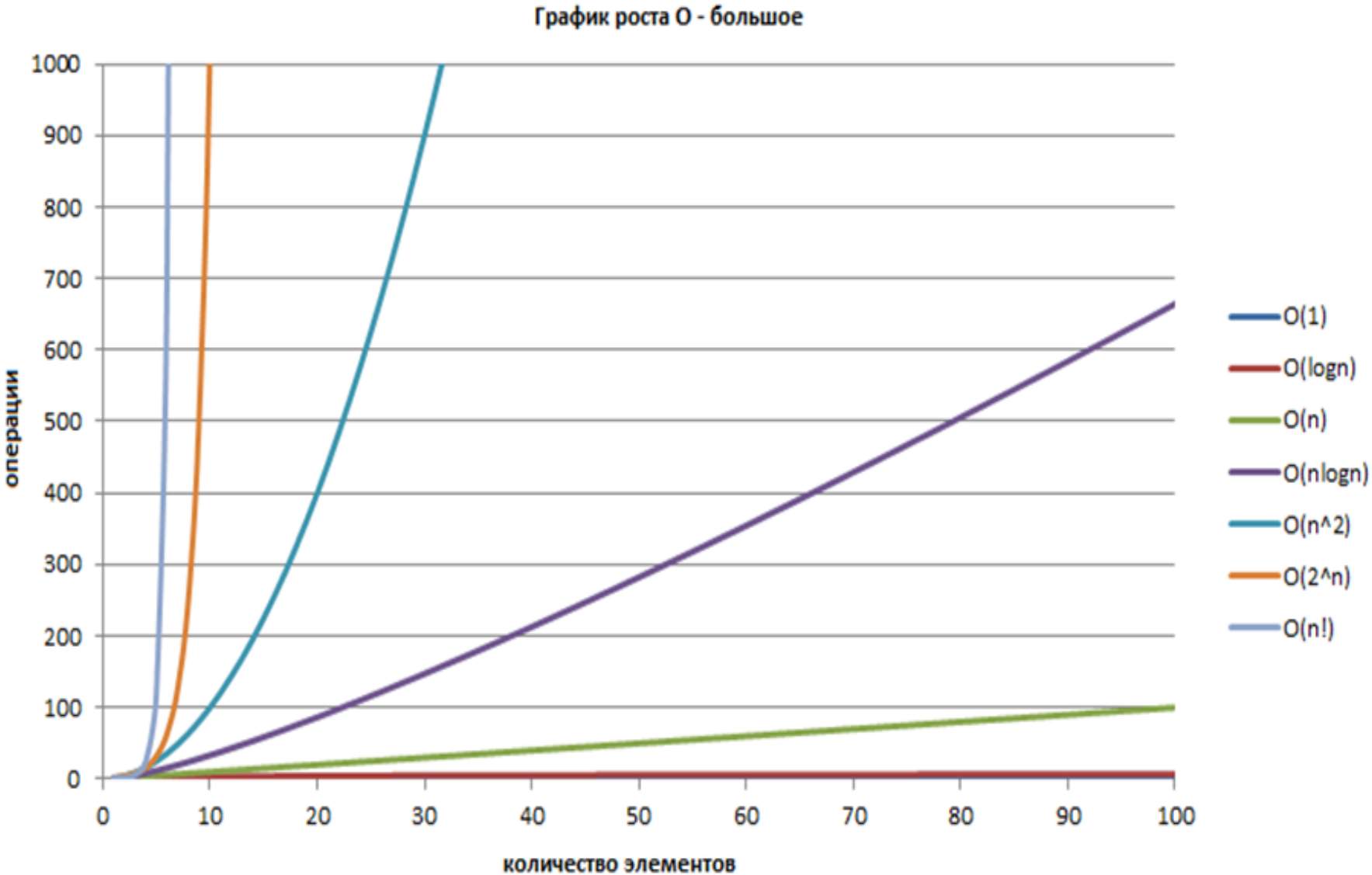
Понятие сложности алгоритма

- Под сложностью алгоритма понимается зависимость объёма работы, от размера входных данных.
- фраза «сложность алгоритма есть O(f(n))» означает, что с увеличением параметра n, характеризующего количество входной информации алгоритма, время работы алгоритма будет возрастать не быстрее, чем некоторая константа, умноженная на f(n).
- O() исключает коэффициенты и члены меньшего порядка.

Понятие сложности алгоритма

- Под сложностью алгоритма понимается зависимость объёма работы, от размера входных данных.
- фраза «сложность алгоритма есть O(f(n))» означает, что с увеличением параметра n, характеризующего количество входной информации алгоритма, время работы алгоритма будет возрастать не быстрее, чем некоторая константа, умноженная на f(n).
- O() исключает коэффициенты и члены меньшего порядка.
- Говорят, что алгоритм работает за линейное время, или O(n), если его сложность равна O(n). Неформально, это означает, что для достаточно большого размера входных данных время работы увеличивается линейно от размера входа.

Понятие сложности алгоритма



Мотивация

Мотивация

- Ускорение обработки данных.

Мотивация

- Ускорение обработки данных.
- Упрощение обработки данных.

Мотивация

- Ускорение обработки данных.
- Упрощение обработки данных.
- Зачем столько разных алгоритмов сортировки?

Мотивация

- Ускорение обработки данных.
- Упрощение обработки данных.
- Зачем столько разных алгоритмов сортировки?

Counting Sort

Counting Sort

- Идея алгоритма заключается в следующем: сначала создается вспомогательный массив, равный длине исходного массива, поданного на сортировку. Последовательно для всех элементов исходного массива выполняется проход, цель которого определить, сколько раз каждый элемент встречается в исходном массиве. Эти данные заносятся во вспомогательный массив, для i элемента вспомогательного массива B номер i записывается в исходный массив B[i] раз.

Counting Sort

- Идея алгоритма заключается в следующем: сначала создается вспомогательный массив, равный длине исходного массива, поданного на сортировку. Последовательно для всех элементов исходного массива выполняется проход, цель которого определить, сколько раз каждый элемент встречается в исходном массиве. Эти данные заносятся во вспомогательный массив, для i элемента вспомогательного массива B номер i записывается в исходный массив B[i] раз.
- Главный недостаток алгоритма - работает лишь для целых положительных чисел.

Утверждение

Алгоритм работает за O(n+k)

- Counting sort

Листинг 1: Реализация на Python

```
for i in range (len(a)):
  A[a[i]]=A[a[i]]+1
for i in range(len(A)):
  while(A[i]>0):
    a[j]=i
    A[i]-=1
    j+=1
```

Radix Sort

Radix Sort

- Алгоритм состоит в последовательной сортировке объектов какой-либо устойчивой сортировкой по каждому разряду, в порядке от младшего разряда к старшему, после чего последовательности будут расположены в требуемом порядке.

Radix Sort

- Алгоритм состоит в последовательной сортировке объектов какой-либо устойчивой сортировкой по каждому разряду, в порядке от младшего разряда к старшему, после чего последовательности будут расположены в требуемом порядке.
- Наиболее часто в качестве устойчивой сортировки применяют сортировку подсчетом.

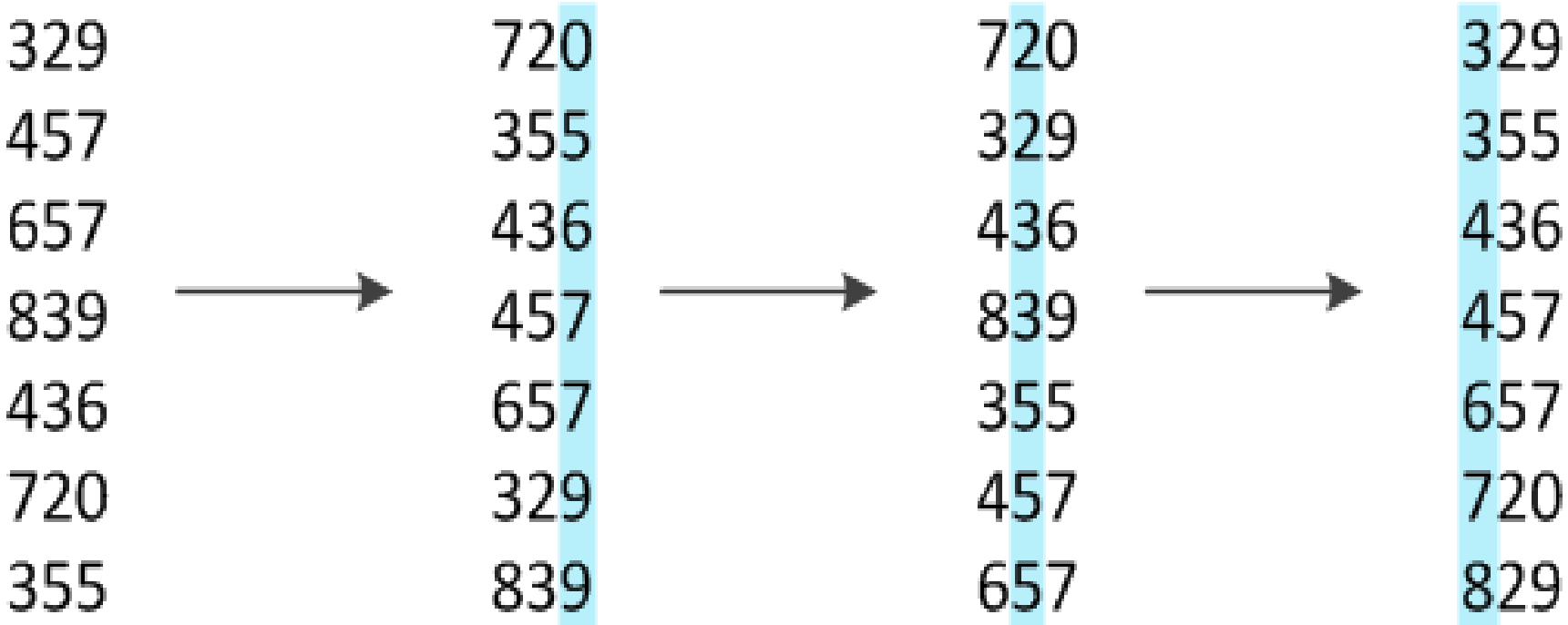
Утверждение

Алгоритм работает за $O(n+k)$

Counting sort

Radix Sort

- Пример Поразрядной сортировки



Листинг 2: Реализация на Python

```
length = len(str(max(A)))
lente = len(A)
for i in range(length):
  B = [[] for k in range(lente)]
    for j in range(len(A)):
        n = A[i] // 10**i % 10
        B[n].append(A[i])
    A = []
   for k in range(lente):
        A = A + B[k]
```

Insertion Sort

Insertion Sort

- На каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированной части массива, до тех пор пока весь набор входных данных не будет отсортирован. Метод выбора очередного элемента из исходного массива произволен, однако обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве.

Insertion Sort

- На каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированной части массива, до тех пор пока весь набор входных данных не будет отсортирован. Метод выбора очередного элемента из исходного массива произволен, однако обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве. Insertion sort

Листинг 3: Реализация на Python

```
for i in range(len(data)):
    j = i - 1
    key = data[i]
    while (data[j] > key and j >= 0):
      data[j + 1] = data[j]
      j -= 1
    data[j + 1] = key
```

Bucket Sort

Bucket Sort

- Для карманной сортировки нужно разбить элементы массива входных данных на k блоков (карманов, корзин). Далее каждый из таких блоков сортируется либо другой сортировкой, либо рекурсивно тем же методом разбиения. После сортировок внутри каждых блоков данные записываются в массив в порядке разбиения на блоки. При этом нужно учитывать, что данная сортировка работает только в том случае, если разбиение на блоки производится таким образом, чтобы элементы каждого следующего блока были больше предыдущего.

Bucket Sort

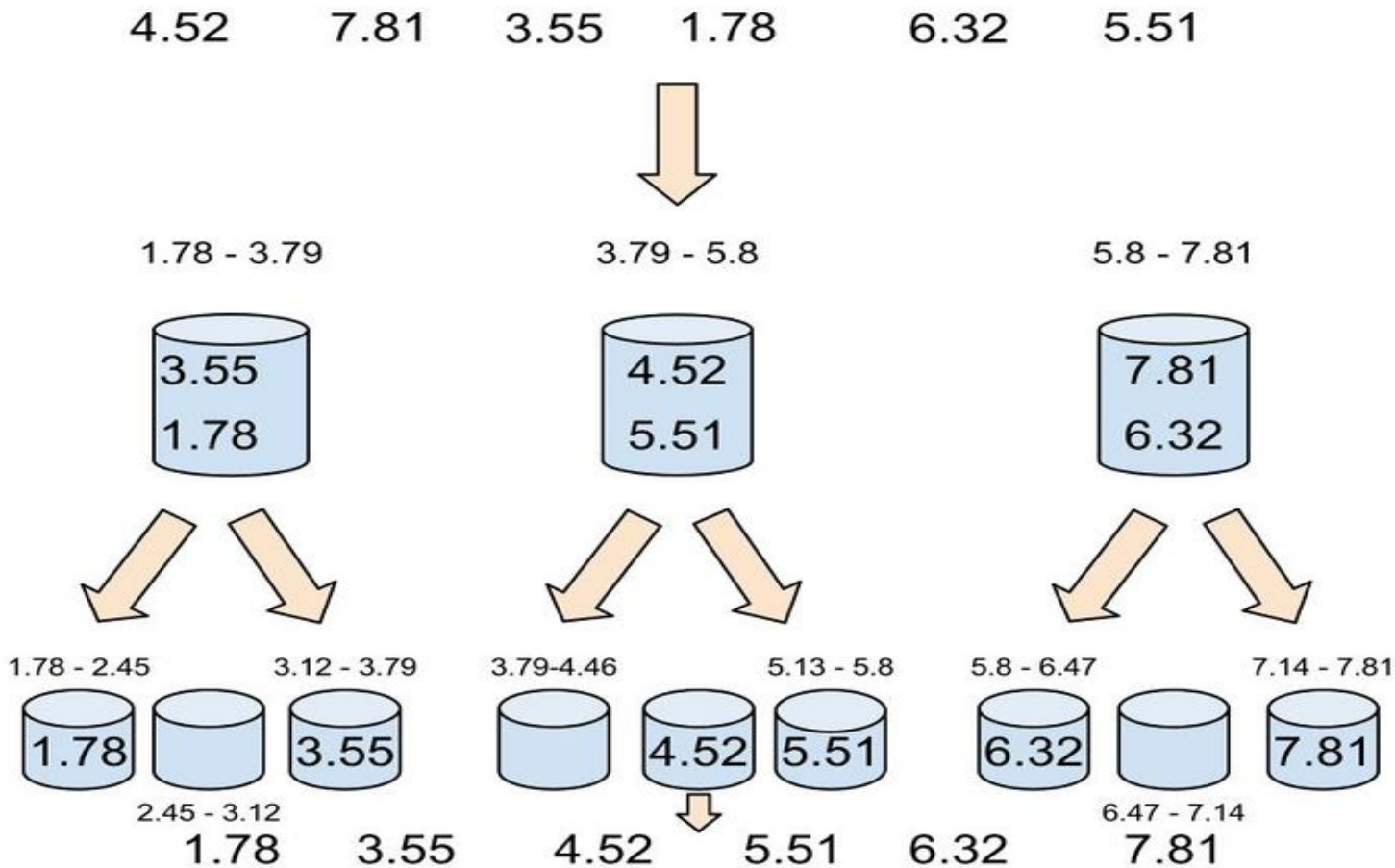
- Для карманной сортировки нужно разбить элементы массива входных данных на k блоков (карманов, корзин). Далее каждый из таких блоков сортируется либо другой сортировкой, либо рекурсивно тем же методом разбиения. После сортировок внутри каждых блоков данные записываются в массив в порядке разбиения на блоки. При этом нужно учитывать, что данная сортировка работает только в том случае, если разбиение на блоки производится таким образом, чтобы элементы каждого следующего блока были больше предыдущего.
- Карманная сортировка сильно деградирует при большом количестве мало отличных элементов (большинство элементов попадёт в одну корзину). Поэтому такой тип сортировки использовать, когда велика вероятность того, что числа редко повторяются (например, последовательность случайных чисел)

Bucket Sort

- Для карманной сортировки нужно разбить элементы массива входных данных на k блоков (карманов, корзин). Далее каждый из таких блоков сортируется либо другой сортировкой, либо рекурсивно тем же методом разбиения. После сортировок внутри каждых блоков данные записываются в массив в порядке разбиения на блоки. При этом нужно учитывать, что данная сортировка работает только в том случае, если разбиение на блоки производится таким образом, чтобы элементы каждого следующего блока были больше предыдущего.
- Карманная сортировка сильно деградирует при большом количестве мало отличных элементов (большинство элементов попадёт в одну корзину). Поэтому такой тип сортировки использовать, когда велика вероятность того, что числа редко повторяются (например, последовательность случайных чисел)

Bucket Sort

- Пример сортировки вычёрпыванием
- Bucket Sort



Листинг 4: Реализация на Python

```
def bucketSort(x):
arr = []
slot_num = 10
for i in range(slot_num):
    arr.append([])

for j in x:
    index_b = int(slot_num * j)
    arr[index_b].append(j)

for i in range(slot_num):
    arr[i] = insertionSort(arr[i])
k = 0
for i in range(slot_num):
    for j in range(len(arr[i])):
        x[k] = arr[i][j]
        k += 1
return x
```

Bucket Sort

- Теорема. Алгоритм SortB работает за время O(N) в среднем, где N – количество сортируемых элементов.
  Доказательство. Пусть p=1/N. Вероятность попадания в один контейнер k элементов равна pk=CNk pk (1-p)N-k (биноминальное распределение). Время работы алгоритма сортировки в одном контейнере равно S O(k2), где k – количество элементов, попавших в i-ый контейнер.
  Согласно свойствам биномиального распределения, среднее (математическое ожидание) количество элементов в контейнере равно M(k)= Sk pkk=Np=1.
  Средне-квадратичное отклонение от среднего значения (дисперсия) количества элементов в контейнере равно D(k)= S k pk(k- M(k))2=S k pk(k-1)2=Np(1-p)=1-1/N.
  D(k)=M(k2) – (M(k))2 из чего сразу следует M(k2)=D(k) +(M(k))2=2-1/N. Итого, среднее время сортировки одного контейнера равно O(1), а среднее время сортировок N контейнеров равно O(N).

Pancake Sort

- Алгоритм сортировки с помощью одной операции — переворота элементов последовательности до какого-то индекса (префикса последовательности). Разумеется, разрешены сравнения, при оценке времени работы этого алгоритма оценивается количество переворотов, а не сравнений. Название алгоритма пошло от изначальной задачи отсортировать стопку блинов по возрастанию размера.
- Pancake sort

Утверждение

Алгоритм корректен и работает за линейное время.

Pancake Sort

- Покажем, что любую последовательность можно отсортировать с помощью блинной сортировки. Для этого будет предложен алгоритм, позволяющий отсортировать любой массив, сделав не более 2n операций, где n — размер массива. Найдём max элемент последовательности с номером i и развернём префикс массива до i-го элемента. Теперь max элемент находится в начале массива. Развернём весь массив, теперь max элемент находится в конце массива. Сделаем то же самое рекуррентно для префикса длины n—1. Переместим второй по возрастанию элемент в конец подотрезка, после чего последние два элемента будут отсортированы, и продолжим для префикса длины n—2. Таким образом, на каждой итерации мы сделаем две операции, и всего итераций будет не больше n. Тогда суммарное количество операций не превосходит 2n и любая последовательность может быть отсортирована таким образом.

Листинг 5: Реализация на Python

```
def pancake(data):

  if len(data) > 1:
    for i in range(len(data), 1, -1):
    mxin=data.index(max(data[:i]))
    if(mxin==(i-1)):
        continue
    data[:(mxin+1)]=list(reversed(data[:(mxin+1)]))
    data[:i]=list(reversed(data[:i]))

return data
```

Литература

- wikipedia.org
- Алгоритмы и алгоритмические языки-Староверов, В.М.,МГУ им. М.В.Ломоносова,2018.
- neerc.ifmo.ru
- Алгоритмы: построение и анализ-Книга, Клиффорд Штайн, Рональд Линн Ривест, Томас Кормен, и Чарльз Эрик Лейзерсон.
- habr.com