# УНИВЕРСИТЕТ ИТМО

## Факультет инфокоммуникационных технологий

## Интеллектуальные системы в гуманитарной среде

## Алгоритмы и структуры данных

## Лабораторная работа №3 «Сортировка слиянием. Метод декомпозиции»

Студентка: Демша Евгения Сергеевна, К3143
Преподаватель: Харьковская Татьяна Александровна

Санкт-Петербург
10/09/2021

## 1 задача. Сортировка слиянием

1. Используя псевдокод процедур Merge и Merge-sort из презентации к Лекции 2, напишите программу сортировки слиянием на Python и проверьте сортировку, создав несколько рандомных массивов, подходящих под параметры:
• Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \le n \le 2 \cdot 10^4$) — 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. 2сек.
• Ограничение по памяти. 256 мб.
2. Для проверки можно выбрать наихудший случай, когда сортируется массив размера 1000, $10^4$, $10^5$ чисел порядка $10^9$, отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний. Сравните, например, с сортировкой вставкой на этих же данных.
3. Перепишите процедуру Merge так, чтобы в ней не использовались сигнальные значения. Сигналом к остановке должен служить тот факт, что все элементы массива L или R скопированы обратно в массив A, после чего в этот массив копируются элементы, оставшиеся в непустом массиве.

Исходный код для 1 пункта:

```
def merge(A, p, q, r):
    n1 = q - p + 1
    n2 = r - q
    L = list(range(n1 + 1))
    R = list(range(n2 + 1))
    for i in range(n1):
        L[i] = A[p + i - 1]
    for j in range(n2):
        R[j] = A[q + j]
    L[-1] = float("inf")
    R[-1] = float("inf")
    i, j = 0, 0
    for k in range(p-1, r):
        if L[i] <= R[j]:
            A[k] = L[i]
            i += 1
        else:
            A[k] = R[j]
            j += 1


def merge_sort(A, p, r):
    if p < r:
        q = (p + r) // 2
        merge_sort(A, p, q)
        merge_sort(A, q+1, r)
        merge(A, p, q, r)
        return A
```

Исходный код для 3 пункта:

```
def merge_(A, p, q, r):
    print(A,p,q,r)
    n1 = q - p + 1
    n2 = r - q
    L = list(range(n1))
    R = list(range(n2))
    for i in range(n1):
        L[i] = A[p + i - 1]
    for j in range(n2):
        R[j] = A[q + j]
    print(L,R)
    i, j = 0, 0
    for k in range(p-1, r):
        if i < len(L) and j < len(R):
            if L[i] <= R[j]:
                A[k] = L[i]
                i += 1
            else:
                A[k] = R[j]
                j += 1
        elif i >= len(L):
            A[k] = R[j]
            j += 1
        elif j >= len(R):
            A[k] = L[i]
            i += 1


def merge_sort_(A, p, r):
    if p < r:
        q = (p + r) // 2
        merge_sort_(A, p, q)
        merge_sort_(A, q+1, r)
        merge_(A, p, q, r)
        return A
```

## 2 задача. Сортировка слиянием+

Дан массив целых чисел. Ваша задача — отсортировать его в порядке неубывания с помощью сортировки слиянием.
Чтобы убедиться, что Вы действительно используете сортировку слиянием, мы просим вас после каждого осуществленного слияния (то есть, когда соответствующий подмассив уже отсортирован!), выводить индексы граничных элементов и их значения.
• Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \le n \le 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих $10^9$.
• Формат выходного файла (output.txt). Выходной файл состоит из нескольких строк.
– В последней строке выходного файла требуется вывести отсортированный в порядке неубывания массив, данный на входе. Между любыми двумя числами должен стоять ровно один пробел.
– Все предшествующие строки описывают осуществленные слияния, по одному на каждой строке. Каждая такая строка должна содержать по четыре числа: If , Il, Vf , Vl, где If — индекс начала области слияния, Il — индекс конца области слияния, Vf — значение первого элемента области слияния, Vl — значение последнего элемента области слияния.
– Все индексы начинаются с единицы (то есть, $1 \le If \le Il \le n$ ).

Индексы области слияния должны описывать положение области слияния в исходном массиве! Допускается не выводить информацию о слиянии для подмассива длиной 1, так как он отсортирован по определению.
• Ограничение по времени. 2сек.
• Ограничение по памяти. 256 мб.

Исходный код:

```
fi = open("Input", "r")
fo = open("Output", "w")
lines = fi.readlines()
n = int(lines[0])
lst = list(lines[1].split())
array = []
for e in lst:
    array.append(int(e))


def merge(A, p, q, r):
    fo.write(str(p))
    fo.write(" ")
    fo.write(str(r))
    fo.write(" ")
    n1 = q - p + 1
    n2 = r - q
    L = list(range(n1 + 1))
    R = list(range(n2 + 1))
    for i in range(n1):
        L[i] = A[p + i - 1]
    for j in range(n2):
        R[j] = A[q + j]
    L[-1] = float("inf")
    R[-1] = float("inf")
    i, j = 0, 0
    for k in range(p-1, r):
        if L[i] <= R[j]:
            A[k] = L[i]
            i += 1
        else:
            A[k] = R[j]
            j += 1
    fo.write(str(A[p-1]))
    fo.write(" ")
    fo.write(str(A[r-1]))
    fo.write("\n")


def merge_sort(A, p, r):
    if p < r:
        if ((p + r) // 2) % 2 == 0 or (p + r)//2 == p or (p + r)//2 == r:
            q = (p + r) // 2
        else:
            q = (p + r) // 2 + 1
        merge_sort(A, p, q)
        merge_sort(A, q+1, r)
        merge(A, p, q, r)
        return A
```

```
merge_sort(array, 1, n)
for e in array:
   fo.write(str(e))
   fo.write(" ")

fi.close()
fo.close()
```

Тесты:

| Input | Output | Time |
|---|---|---|
| 10<br>1 8 2 1 4 7 3 2 3 6 | 1 2 1 8<br>3 4 1 2<br>1 4 1 8<br>5 6 4 7<br>1 6 1 8<br>7 8 2 3<br>9 10 3 6<br>7 10 2 6<br>1 10 1 8<br>1 1 2 2 3 3 4 6 7 8 | 0.00046552019654367723<br>секунд |

## 4 задача. Бинарный поиск

В этой задаче вы реализуете алгоритм бинарного поиска, который позволяет очень эффективно искать (даже в огромных) списках при условии, что список отсортирован. Цель - реализация алгоритма двоичного (бинарного) поиска.
• Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \le n \le 105$) — число элементов в массиве, и последовательность $a0 < a1 < ... < an-1$ из n различных положительных целых чисел в порядке возрастания, $1 \le ai \le 109$ для всех $0 \le i < n$. Следующая строка содержит число k, $1 \le k \le 105$ и k положительных целых чисел b0, ...bk−1, $1 \le bj \le 109$ для всех $0 \le j < k$.
• Формат выходного файла (output.txt). Для всех i от 0 до k − 1 вывести индекс $0 \le j \le n - 1$, такой что ai = bj или -1, если такого числа в массиве нет.
• Ограничение по времени. 2сек.
• Ограничение по памяти. 256 мб.

Исходный код:

```
fi = open("Input", "r")
fo = open("Output", "w")
lines = fi.readlines()
n = int(lines[0])
a_list = list(lines[1].split())
a_array = []
for e in a_list:
   a_array.append(int(e))
k = int(lines[2])
b_list = list(lines[3].split())
b_array = []
for e in b_list:
   b_array.append(int(e))
```

```
def binary_search(A, low, high, V):
    if high <= low:
        return fo.write("-1"), fo.write(" ")
    mid = low + (high - low) // 2
    if V == A[mid]:
        return fo.write(str(mid)), fo.write(" ")
    elif V < A[mid]:
        return binary_search(A, low, mid - 1, V)
    else:
        return binary_search(A, mid + 1, high, V)


for i in b_array:
    binary_search(a_array, 0, n, i)

fi.close()
fo.close()
```

Тесты:

| Input | Output | Time |
|---|---|---|
| 5<br>1 5 8 12 13<br>5<br>8 1 23 1 11 | 2 0 -1 0 -1 | 0.00042512099753366783 секунд |
| 10<br>1 4 5 7 10 23 28 30 31 36<br>7<br>40 1 10 2 28 36 10 | -1 0 4 -1 6 9 4 | 0.000559291998797562 секунд |

## 6 задача. Поиск максимальной прибыли

Используя псевдокод процедур Find Maximum Subarray и Find Max Crossing Subarray из презентации к Лекции 2, напишите программу поиска максимального подмассива.

Примените ваш алгоритм для ответа на следующий вопрос. Допустим, у нас есть данные по акциям какой-либо фирмы за последний месяц (год, или иной срок).
Проанализируйте этот срок и выдайте ответ, в какой из дней при покупке единицы акции данной фирмы, и в какой из дней продажи, вы бы получили максимальную прибыль? Выдайте дату покупки, дату продажи и максимальную прибыль.

Соответственно, вам нужно только выбрать данные, посчитать изменение цены и применить алгоритм поиска максимального подмассива.

• Формат входного файла в данном случае на ваше усмотрение.

• Формат выходного файла (output.txt). Выведите название фирмы, рассматриваемый вами срок изменения акций, дату покупки и дату продажи единицы акции, чтобы получилась максимальная выгода; и сумма этой прибыли.

Исходный код:

```
fi = open("TeslaStocks", "r")
fo = open("Output", "w")
lines = fi.readlines()
company = lines[0]
dates = lines[1]
array = []
d_array = []
for a in range(2, len(lines)):
   date, price = lines[a].split()
   array.append(float(price))
   d_array.append(str(date))
n = len(array)



def find_max_crossing_subarray(A, low, mid, high):
   leftsum = float("-inf")
   sum = 0
   maxleft = mid
   for i in range(mid, low - 1, -1):
      sum += A[i]
      if sum > leftsum:
         leftsum = sum
         maxleft = i
   rightsum = float("-inf")
   sum = 0
   maxright = mid
   for j in range(mid + 1, high + 1):
      sum += A[j]
      if sum > rightsum:
         rightsum = sum
         maxright = j
   return maxleft, maxright, leftsum + rightsum



def find_maximum_subarray(A, low, high):
   if high == low:
      return low, high, A[low]
   else:
      mid = (low + high) // 2
      leftlow, lefthigh, leftsum = find_maximum_subarray(A, low, mid)
      rightlow, righthigh, rightsum = find_maximum_subarray(A, mid + 1, high)
      crosslow, crosshigh, crosssum = find_max_crossing_subarray(A, low, mid, high)
      if leftsum >= rightsum and leftsum >= crosssum:
         return leftlow, lefthigh, leftsum
      elif rightsum >= leftsum and rightsum >= crosssum:
         return rightlow, righthigh, rightsum
      else:
         return crosslow, crosshigh, crosssum



start, finish, summ = find_maximum_subarray(array, 0, n-2)

fo.write(str(company))
fo.write(str(dates))
fo.write(str(d_array[start]))
fo.write(" ")
fo.write(str(d_array[finish]))
fo.write(" ")
fo.write(str(summ))
```

```
fi.close()
fo.close()
```

Тесты:

| TeslaStocks | Output | time |
|---|---|---|
| Tesla<br>01/07/2021 - 31/07/2021<br>01/07/2021 677.92<br>02/07/2021 678.90<br>06/07/2021 659.58<br>07/07/2021 644.65<br>08/07/2021 652.81<br>09/07/2021 656.95<br>12/07/2021 685.70<br>13/07/2021 668.54<br>14/07/2021 653.38<br>15/07/2021 650.60<br>16/07/2021 644.22<br>19/07/2021 646.22<br>20/07/2021 660.50<br>21/07/2021 655.29<br>22/07/2021 649.26<br>23/07/2021 643.38<br>26/07/2021 657.62<br>27/07/2021 644.78<br>28/07/2021 646.98<br>29/07/2021 677.35<br>30/07/2021 687.20 | Tesla<br>01/07/2021 - 31/07/2021<br>01/07/2021 29/07/2021 13154.630000000001 | 0.0007135579999157926 секунд |

## 9 задача. Метод Штрассена для умножения матриц

Цель. Применить метод Штрассена для умножения матриц и сравнить его с простым методом. Найти размер матриц n, при котором метод Штрассена работает существенно быстрее простого метода.

• Формат входа. Стандартный ввод или input.txt. Первая строка - размер квадратных матриц n для умножения. Следующие строки соответсвенно сами значения матриц A и B.
• Формат выхода. Стандартный вывод или output.txt. Матрица C = A · B.

Исходный код для простого метода:

```
fi = open("Input", "r")
fo = open("Output", "w")
lines = fi.readlines()
n = int(lines[0])
matrix_1 = []
matrix_2 = []
for a in range(1, n+1):
  f = lines[a].split()
  b = []
  for e in f:
    b.append(int(e))
  row_1 = []
  row_2 = []
```

```
    for c in enumerate(b):
        if c[0] < n:
            row_1.append(c[1])
        else:
            row_2.append(c[1])
    matrix_1.append(row_1)
    matrix_2.append(row_2)


def matrix_multiply(x, y):
    z = []
    for d in range(n):
        z.append(list(range(n)))
    for i in range(n):
        for j in range(n):
            z[i][j] = 0
            for k in range(n):
                z[i][j] = z[i][j] + x[i][k]*y[k][j]
    return z


for i in matrix_multiply(matrix_1,matrix_2):
    for j in i:
        fo.write(str(j))
        fo.write(" ")
    fo.write("\n")

fi.close()
fo.close()
```

Исходный код для метода Штрассена:

```
import math


def decimal(s):
    return int(s) if s.is_integer() else s


def two_exp(matrix):
    if type(decimal(math.log2(len(matrix)))) != int:
        for f in matrix:
            f.append(0)
        matrix.append([0] * (len(matrix) + 1))
        two_exp(matrix)
    else:
        return matrix


def breakdown(matrix):
    a = []
    for i in range(len(matrix) // 2):
        inner = []
        for j in range(len(matrix[i]) // 2):
            inner.append(matrix[i][j])
        a.append(inner)
    b = []
    for i in range(len(matrix) // 2):
```

```
        inner = []
        for j in range(len(matrix[i]) // 2, len(matrix)):
            inner.append(matrix[i][j])
        b.append(inner)
    c = []
    for i in range(len(matrix) // 2, len(matrix)):
        inner = []
        for j in range(len(matrix[i]) // 2):
            inner.append(matrix[i][j])
        c.append(inner)
    d = []
    for i in range(len(matrix) // 2, len(matrix)):
        inner = []
        for j in range(len(matrix[i]) // 2, len(matrix)):
            inner.append(matrix[i][j])
        d.append(inner)
    return a, b, c, d


def minus(x, y):
    z = []
    for d in range(len(x)):
        z.append(list(range(len(x))))
    for i in range(len(x)):
        for j in range(len(x)):
            z[i][j] = x[i][j] - y[i][j]
    return z


def plus(x, y):
    z = []
    for d in range(len(x)):
        z.append(list(range(len(x))))
    for i in range(len(x)):
        for j in range(len(x)):
            z[i][j] = x[i][j] + y[i][j]
    return z


def no_zeros(matrix):
    while n < len(matrix):
        for f in matrix:
            f.pop(-1)
        matrix.pop(-1)
    else:
        return matrix


def strassen(x, y):
    two_exp(x)
    two_exp(y)
    if len(x) <= 2:
        a, b, c, d = x[0][0], x[0][1], x[1][0], x[1][1]
        e, f, g, h = y[0][0], y[0][1], y[1][0], y[1][1]
        p1 = a * (f - h)
        p2 = (a + b) * h
        p3 = (c + d) * e
        p4 = d * (g - e)
        p5 = (a + d) * (e + h)
        p6 = (b - d) * (g + h)
```

```
    p7 = (a - c) * (e + f)
    z = []
    for p in range(len(x)):
        z.append(list(range(len(x))))
    z[0][0] = p5 + p4 - p2 + p6
    z[0][1] = p1 + p2
    z[1][0] = p3 + p4
    z[1][1]= p1 + p5 - p3 - p7
  else:
    a,b,c,d = breakdown(x)
    e,f,g,h = breakdown(y)
    p1 = strassen(a, minus(f, h))
    p2 = strassen(plus(a, b), h)
    p3 = strassen(plus(c, d), e)
    p4 = strassen(d, minus(g, e))
    p5 = strassen(plus(a, d), plus(e, h))
    p6 = strassen(minus(b, d), plus(g, h))
    p7 = strassen(minus(a, c), plus(e, f))
    z1 = minus(plus(p5, p4),minus(p2, p6))
    z2 = plus(p1, p2)
    z3 = plus(p3, p4)
    z4 = minus(plus(p1, p5),plus(p3, p7))
    z = []
    for p in range(len(x)):
        z.append(list(range(len(x))))
    for i in range(len(z) // 2):
        for j in range(len(z[i]) // 2):
            z[i][j] = z1[i][j]
    for i in range(len(z) // 2):
        for j in range(len(z[i]) // 2, len(z)):
            z[i][j] = z2[i][j - len(z[1]) // 2]
    for i in range(len(z) // 2, len(z)):
        for j in range(len(z[i]) // 2):
            z[i][j] = z3[i - len(z) // 2][j]
    for i in range(len(z) // 2, len(z)):
        for j in range(len(z[i]) // 2, len(z)):
            z[i][j] = z4[i - len(z) // 2][j - len(z) // 2]
    no_zeros(z)
  return z
```

Тесты:

| input | output | time |
|---|---|---|
| 3<br>1 -3 2 4 -1 -3<br>4 2 5 -6 7 -7<br>0 9 -8 -3 4 -5 | 16 -14 8<br>-11 30 -51<br>-30 31 -23 | 0.001149636000263854 секунд |
| 5<br>1 2 -3 0 1 -2 0 3 -1 0<br>3 1 0 -2 -1 -1 -3 0 1 1<br>0 3 -2 -1 2 2 3 -2 0 -3<br>-2 0 1 3 -3 0 1 2 -3 0<br>-3 1 0 0 -2 -1 -1 3 2 -2 | -11 -16 12 3 9<br>-6 -4 2 2 3<br>-9 -18 8 10 5<br>9 9 -11 -13 3<br>7 -1 -15 0 5 | 0.004219508010223609 секунд |

## Вывод:

Рекурсия – штука очень увлекательная, непонятная и эффективная.