Лабораторная работа 1. Методы сортировки.

Выполнил: Щитов В.М.
Группа: БВТ1903
Вариант: 22

Задание.

Реализовать метод сортировки Шелла строк числовой матрицы в соответствии с индивидуальным заданием. Добавить реализацию быстрой сортировки (quicksort). Оценить время работы каждого алгоритма сортировки и сравнить его со временем стандартной функции сортировки, используемой в Python.

Необходимые библиотеки.

Для реализации двумерного массива (матрицы) можно использовать библиотеку **NumPy**. Для замеров времени выполнения алгоритмов используется модуль **time** стандартной библиотеки языка.

In [1]:
```
# Импортирование библиотеки
import numpy as np
import time
import random
```

In [2]:
```
# Реализация матрицы с помощью двумерного массива NumPy
SIZE = 1000
matrix = np.zeros((SIZE, SIZE))

# Генерация случайно матрицы
for i in range (0, SIZE):
    for j in range (0, SIZE):
        matrix[i][j] = random.randrange(-1000, 1000)

# Вывод сгенерированной матрицы
print(matrix)
```

```
[[-445.  230. -300. ... -425.  273. -446.]
 [ 811.  -16.  903. ...  246. -123. -194.]
 [-827. -597.  663. ... -601.  102. -671.]
 ...
 [ 770.  705.  566. ...  481.  239. -910.]
 [ 944.  131. -548. ... -338.  882.  352.]
 [ 103. -229.  740. ... -693. -244. -134.]]
```

In [3]:
```
# Функция сортировки Шелла для каждой строки матрицы
def shell_sort(row):
    last_index = len(row) - 1
    step = len(row) // 2
    sort_row = row.copy()
    while step > 0:
        for i in range(step, last_index + 1, 1):
            j = i
            delta = j - step
            while delta >= 0 and sort_row[delta] > sort_row[j]:
                sort_row[delta], sort_row[j] = sort_row[j], sort_row[delta]
                j = delta
                delta = j - step
        step //= 2
    return sort_row
```

In [4]:
```
# Функция возвращает индекс опорного элемента
def partition(row, low, high):
    x = row[high]
    i = low - 1
    j = low
    tmp = 0
    while j < high:
        if row[j] <= x:
            i = i + 1
            tmp = row[i]
            row[i] = row[j]
            row[j] = tmp
        j = j + 1
    tmp = row[high]
    row[high] = row[i + 1]
    row[i + 1] = tmp
    return i + 1

# Функция быстрой сортировки
def quicksort(row, low, high):
    if low < high:
        p = partition(row, low, high)
        quicksort(row, low, p - 1)
        quicksort(row, p + 1, high)
```

In [5]:
```
# Функция, формирующая отсортированную матрицу
def using_sort(matrix, sort_type = 'none'):
    sort_matrix = np.zeros(matrix.shape)
    algo_time = 0;

    # Если выбрана сортировка Шелла
    if sort_type == 'shell':
        tic = time.process_time()
        for i in range (0, len(matrix)):
            sort_matrix[i] = shell_sort(matrix[i,:])
        toc = time.process_time()
        algo_time = toc - tic

    # Если выбрана быстрая сортировка
    elif sort_type == 'quick':
        sort_matrix = matrix.copy()
        tic = time.process_time()
        for i in range (0, len(matrix)):
            quicksort(sort_matrix[i,:], 0, sort_matrix.shape[1] - 1)
        toc = time.process_time()
        algo_time = toc - tic

    # В противном случае выбирается стандартная функция сортировки NumPy
    else:
        tic = time.process_time()
        for i in range (0, len(matrix)):
            sort_matrix[i] = np.sort(matrix[i,:])
        toc = time.process_time()
        algo_time = toc - tic

    # Вывод результатов на экран и их возвращение функцией
    print('Время выполнения сортировки (в секундах): ', algo_time)
    print('Результат:\n', sort_matrix, "\n")
    return sort_matrix
```

In [6]:
```
# Вызов каждого метода сортировки с выводом времени выполнения и отсортированной матрицы

# Сортировка Шелла
print('Сортировка Шелла')
a = using_sort(matrix, 'shell')

# Быстрая сортировка
print('Быстрая сортировка')
a = using_sort(matrix, 'quick')

# Стандартная функция сортировки
print('Стандартная функция сортировки NumPy')
a = using_sort(matrix)
```

```
Сортировка Шелла
Время выполнения сортировки (в секундах):  17.578125
Результат:
 [[ -998.  -997.  -994. ...   991.   996.   996.]
 [-1000.  -999.  -998. ...   997.   999.   999.]
 [ -997.  -994.  -994. ...   996.   997.   998.]
 ...
 [ -995.  -992.  -992. ...   999.   999.   999.]
 [ -998.  -994.  -992. ...   994.   998.   999.]
 [ -998.  -995.  -994. ...   998.   998.   999.]]

Быстрая сортировка
Время выполнения сортировки (в секундах):  10.78125
Результат:
 [[ -998.  -997.  -994. ...   991.   996.   996.]
 [-1000.  -999.  -998. ...   997.   999.   999.]
 [ -997.  -994.  -994. ...   996.   997.   998.]
 ...
 [ -995.  -992.  -992. ...   999.   999.   999.]
 [ -998.  -994.  -992. ...   994.   998.   999.]
 [ -998.  -995.  -994. ...   998.   998.   999.]]

Стандартная функция сортировки NumPy
Время выполнения сортировки (в секундах):  0.09375
Результат:
 [[ -998.  -997.  -994. ...   991.   996.   996.]
 [-1000.  -999.  -998. ...   997.   999.   999.]
 [ -997.  -994.  -994. ...   996.   997.   998.]
 ...
 [ -995.  -992.  -992. ...   999.   999.   999.]
 [ -998.  -994.  -992. ...   994.   998.   999.]
 [ -998.  -995.  -994. ...   998.   998.   999.]]
```