Министерство цифрового развития, связи и массовых коммуникаций

Ордена Трудового Красного Знамени федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и информационных технологий

Отчёт по лабораторной работе № 1

«Методы сортировки»

по дисциплине «Сиаод»

Выполнил: студент группы

БВТ1902

Сорокин Никита Андреевич

Москва

2021

Цель работы

Написать генератор случайных матриц(многомерных), который принимает опциональные параметры m, n, min_limit, max_limit, где m и n указывают размер матрицы, а min_lim и max_lim - минимальное и максимальное значение для генерируемого числа . По умолчанию при отсутствии параметров принимать следующие значения: m = 50 n = 50 min_limit = -250 max_limit = 1000 + (номер своего варианта)

Реализовать методы сортировки строк числовой матрицы в соответствии с заданием. Оценить время работы каждого алгоритма сортировки и сравнить его со временем стандартной функции сортировки. Испытания проводить на сгенерированных матрицах. Методы: сортировка выбором, сортировка вставкой, сортировка обменом, сортировка шелла, быстрая сортировка, пирамидальная сортировка.

Код программы

```
from random import randint
import random
import time
import copy

xstr = input("Введите x: ")
ystr = input("Введите y: ")

if xstr == "":
    x = 50
else:
    x = int(xstr)

if ystr == "":
    y = 50
else:
    y = int(ystr)

minstr = input("Введите min: ")
maxstr = input("Введите max: ")

if minstr == "":
    min = -250
else:
    min = int(minstr)

if maxstr == "":
    max = 1021
else:
    max = int(maxstr)

print()
print("Результат: ")

mass = []

for i in range(y):
    mass2 = []
    for j in range(x):
        mass2.append(randint(min, max))
    print(mass2)
    mass.append(mass2)

def sel_sort(array):
    for i in range(len(array)):
        m = i
        j = i + 1
        while j < len(array):
```

```
            if array[j] < array[m]:
                m = j
            j = j + 1
        array[i], array[m] = array[m], array[i]
    print(array)


print("Результат сортировки выбором: ")

start_time = time.time()

for j in range(y):
    arr = []
    for i in range(x):
        arr.append(mass[j][i])
    sel_sort(arr)

print("Потраченное время(с): ")
sec1 = float('{:.8f}'.format(time.time() - start_time))
print(sec1)

def insertion_sort(nums):
    for i in range(1, len(nums)):
        item_to_insert = nums[i]
        j = i - 1
        while j >= 0 and nums[j] > item_to_insert:
            nums[j + 1] = nums[j]
            j -= 1
        nums[j + 1] = item_to_insert
    print(nums)


print("Результат сортировки вставкой: ")

start_time = time.time()

for j in range(y):
    arr = []
    for i in range(x):
        arr.append(mass[j][i])
    insertion_sort(arr)

print("Потраченное время(с): ")
sec2 = float('{:.6f}'.format(time.time() - start_time))
print(sec2)

def insertion_sort(nums):
    for i in range(1, len(nums)):
        item_to_insert = nums[i]
        j = i - 1
```

```
        while j >= 0 and nums[j] > item_to_insert:
            nums[j + 1] = nums[j]
            j -= 1
        nums[j + 1] = item_to_insert
    print(nums)


print("Результат сортировки вставкой: ")

start_time = time.time()

for j in range(y):
    arr = []
    for i in range(x):
        arr.append(mass[j][i])
    insertion_sort(arr)

print("Потраченное время(с): ")
sec2 = float('{:.6f}'.format(time.time() - start_time))
print(sec2)

def shell(mass):
    inc = len(mass) // 2
    while inc:
        for i, el in enumerate(mass):
            while i >= inc and mass[i - inc] > el:
                mass[i] = mass[i - inc]
                i -= inc
            mass[i] = el
        inc = 1 if inc == 2 else int(inc * 5.0 / 11)
    print(mass)


print("Результат сортировки шелла: ")

start_time = time.time()

for j in range(y):
    a = []
    for i in range(x):
        a.append(mass[j][i])
    shell(a)

print("Потраченное время(с): ")
sec4 = float('{:.6f}'.format(time.time() - start_time))
print(sec4)

def quicksort(nums):
    if len(nums) <= 1:
        return nums
```

```
    else:
        q = random.choice(nums)
        s_nums = []
        m_nums = []
        e_nums = []
        for n in nums:
            if n < q:
                s_nums.append(n)
            elif n > q:
                m_nums.append(n)
            else:
                e_nums.append(n)
        return quicksort(s_nums) + e_nums + quicksort(m_nums)


print("Результат быстрой сортировки: ")

start_time = time.time()

fast_sort = []

for j in range(y):
    nums = []
    for i in range(x):
        nums.append(mass[j][i])
    quicksort(nums)
    fast_sort.append(quicksort(nums))

for i in range(y):
    print(fast_sort[i])

print("")
print("Потраченное время(с): ")
sec5 = float('{:.6f}'.format(time.time() - start_time))
print(sec5)

def heapSort(li):
    def downHeap(li, k, n):
        new_elem = li[k]
        while 2 * k + 1 < n:
            child = 2 * k + 1
            if child + 1 < n and li[child] < li[child + 1]:
                child += 1
            if new_elem >= li[child]:
                break
            li[k] = li[child]
            k = child
        li[k] = new_elem

    size = len(li)
```

```
    for i in range(size // 2 - 1, -1, -1):
        downHeap(li, i, size)
    for i in range(size - 1, 0, -1):
        temp = li[i]
        li[i] = li[0]
        li[0] = temp
        downHeap(li, 0, i)
    print(li)


print("Результат пирамидальной сортировки: ")

start_time = time.time()

pir_sort = []

for j in range(y):
    li = []
    for i in range(x):
        li.append(mass[j][i])
    heapSort(li)
    pir_sort.append(quicksort(nums))

print("Потраченное время(с): ")
sec6 = float('{:.6f}'.format(time.time() - start_time))
print(sec6)
```

Скриншоты работы программы

```
Введите x: 10
Введите y: 10
Введите min: -100
Введите max: 100

Результат:
[39, -19, 9, 0, -76, 26, -15, 72, 45, -57]
[87, 45, -23, -77, 73, -71, 15, -34, 15, -96]
[-48, -19, -18, -93, -94, -5, 59, 34, 41, -3]
[61, -30, -20, -63, 33, -21, 87, 37, -92, -69]
[-61, -79, 91, 93, -12, 88, 29, -22, -72, -49]
[-33, -99, -42, 67, -10, -68, -44, -81, 35, -26]
[14, -17, 67, 80, 43, -72, -12, 6, -31, 90]
[-72, -43, 49, 61, 1, -93, -71, -66, -47, 21]
[-30, -94, 53, -8, 1, -42, 69, -75, -47, -36]
[-68, 88, -38, 45, -75, -94, -41, -4, -18, 60]
```

Рис. 1 – Генератор матрицы

```
Результат сортировки выбором:
[-76, -57, -19, -15, 0, 9, 26, 39, 45, 72]
[-96, -77, -71, -34, -23, 15, 15, 45, 73, 87]
[-94, -93, -48, -19, -18, -5, -3, 34, 41, 59]
[-92, -69, -63, -30, -21, -20, 33, 37, 61, 87]
[-79, -72, -61, -49, -22, -12, 29, 88, 91, 93]
[-99, -81, -68, -44, -42, -33, -26, -10, 35, 67]
[-72, -31, -17, -12, 6, 14, 43, 67, 80, 90]
[-93, -72, -71, -66, -47, -43, 1, 21, 49, 61]
[-94, -75, -47, -42, -36, -30, -8, 1, 53, 69]
[-94, -75, -68, -41, -38, -18, -4, 45, 60, 88]
Потраченное время(с):
0.00099659
```

Рис. 2 – Результат сортировки выбором

```
Результат сортировки вставкой:
[-76, -57, -19, -15, 0, 9, 26, 39, 45, 72]
[-96, -77, -71, -34, -23, 15, 15, 45, 73, 87]
[-94, -93, -48, -19, -18, -5, -3, 34, 41, 59]
[-92, -69, -63, -30, -21, -20, 33, 37, 61, 87]
[-79, -72, -61, -49, -22, -12, 29, 88, 91, 93]
[-99, -81, -68, -44, -42, -33, -26, -10, 35, 67]
[-72, -31, -17, -12, 6, 14, 43, 67, 80, 90]
[-93, -72, -71, -66, -47, -43, 1, 21, 49, 61]
[-94, -75, -47, -42, -36, -30, -8, 1, 53, 69]
[-94, -75, -68, -41, -38, -18, -4, 45, 60, 88]
Потраченное время(с):
0.000995
```

Рис. 3 – Результат сортировки вставкой

```
Результат сортировки шелла:
[-76, -57, -19, -15, 0, 9, 26, 39, 45, 72]
[-96, -77, -71, -34, -23, 15, 15, 45, 73, 87]
[-94, -93, -48, -19, -18, -5, -3, 34, 41, 59]
[-92, -69, -63, -30, -21, -20, 33, 37, 61, 87]
[-79, -72, -61, -49, -22, -12, 29, 88, 91, 93]
[-99, -81, -68, -44, -42, -33, -26, -10, 35, 67]
[-72, -31, -17, -12, 6, 14, 43, 67, 80, 90]
[-93, -72, -71, -66, -47, -43, 1, 21, 49, 61]
[-94, -75, -47, -42, -36, -30, -8, 1, 53, 69]
[-94, -75, -68, -41, -38, -18, -4, 45, 60, 88]
```

Рис. 4 – Результат сортировки шелла

```
Результат быстрой сортировки:
[-76, -57, -19, -15, 0, 9, 26, 39, 45, 72]
[-96, -77, -71, -34, -23, 15, 15, 45, 73, 87]
[-94, -93, -48, -19, -18, -5, -3, 34, 41, 59]
[-92, -69, -63, -30, -21, -20, 33, 37, 61, 87]
[-79, -72, -61, -49, -22, -12, 29, 88, 91, 93]
[-99, -81, -68, -44, -42, -33, -26, -10, 35, 67]
[-72, -31, -17, -12, 6, 14, 43, 67, 80, 90]
[-93, -72, -71, -66, -47, -43, 1, 21, 49, 61]
[-94, -75, -47, -42, -36, -30, -8, 1, 53, 69]
[-94, -75, -68, -41, -38, -18, -4, 45, 60, 88]

Потраченное время(с):
0.000994
```

Рис. 5 – Результат быстрой сортировки

```
Результат пирамидальной сортировки:
[-76, -57, -19, -15, 0, 9, 26, 39, 45, 72]
[-96, -77, -71, -34, -23, 15, 15, 45, 73, 87]
[-94, -93, -48, -19, -18, -5, -3, 34, 41, 59]
[-92, -69, -63, -30, -21, -20, 33, 37, 61, 87]
[-79, -72, -61, -49, -22, -12, 29, 88, 91, 93]
[-99, -81, -68, -44, -42, -33, -26, -10, 35, 67]
[-72, -31, -17, -12, 6, 14, 43, 67, 80, 90]
[-93, -72, -71, -66, -47, -43, 1, 21, 49, 61]
[-94, -75, -47, -42, -36, -30, -8, 1, 53, 69]
[-94, -75, -68, -41, -38, -18, -4, 45, 60, 88]
Потраченное время(с):
0.000997
```

Рис. 6 – Результат пирамидальной сортировки

Вывод

В ходе выполнения лабораторной работы, я научился реализовывать различные алгоритмы сортировок. В ходе тестов было понятно, что скорость зависит от количества исходный элементов, но самыми быстрыми являются “быстрая” и “пирамидальная” сортировка.