Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий и систем связи

Лабораторная работа №3

Вариант №1

Выполнил(и:)

Гусев Я.А.

Проверил

Мусаев А.А.

Санкт-Петербург,

2022

Задание 1

На рисунке 1 изображён написанный мной алгоритм пузырьковой сортировки.

```
import math

example = [24, -79, -85, -99, 79, -24, -22, -100, 24, 74, 75, 41, -12, -9, -55, -90, -1, -17, -16, -1]
need = True
steps = 3
while need:
    need = False
    steps += 3
    for i in range(len(example) - 1):
        steps += 2
        if example[i] > example[i + 1]:
            example[i], example[i + 1] = example[i + 1], example[i]
            need = True
            steps += 4
print(example)
print(steps)
print(len(example) * math.log(len(example)))
#Сложность n^2
```

```
main ×    bubble_real ×
C:\Users\GoldenJaden\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\GoldenJaden\Desktop\bubble_real.py
[-100, -99, -90, -85, -79, -55, -24, -22, -17, -16, -12, -9, -1, -1, 24, 24, 41, 74, 75, 79]
929
59.914645471079815
```

Рисунок 1 – Пузырьковая сортировка

Считаем шаги, находим сложность нашего алгоритма (929).

Далее считаем сложность метода sort(). Согласно википедии его сложность составляет n * log(n) = 59,9.

Получается, что метод sort() быстрее моего алгоритма пузырьковой сортировки в 15 с половиной раз.

Задание 2

Реализовать алгоритмы, имеющие сложность:

O(3n):

```
a = [76, -2, 75, 60, 5, 87, 41, 33, 81, 32, 70, -43, 21, -59, -68]
suma = 0
umn = 1
umn_kv = 1
for i in range(len(a)):
    suma += a[i]
for i in range(len(a)):
    umn *= a[i]
for i in range(len(a)):
    umn_kv *= a[i]**2
print(suma, umn, umn_kv)
```

Рисунок 2 – алгоритм со сложностью O(3n)

O(nlogn):

```
a = [-74, -68, -62, -59, -43, -30, -22, -13]
for i in range(len(a)-1, -1, -1):
    x = a[i]
    mx = len(a)
    mn = 0
    mid = (mx + mn) // 2
    while a[mid] != x:
        if a[mid] < x:
            mn = mid
        elif a[mid] > x:
            mx = mid
        mid = (mx + mn) // 2
    print(a[mid])
```

Рисунок 3 – алгоритм со сложностью O(nlogn)

O(n!):

```python
n = int(input())
def factorials(n):
    if n <= 0: return 0
    k = 1
    for i in range(1, n + 1):
        k *= i
    print(k)
    factorials(n-1)
factorials(n)
```

Рисунок 4 – алгоритм со сложностью O(n!)

O(n^3):

```python
n = int(input())
cnt = 0
for i in range(n):
    for j in range(n):
        for p in range(n):
            cnt += 1
print(cnt)
```

Рисунок 5 – алгоритм со сложностью O(n^3)

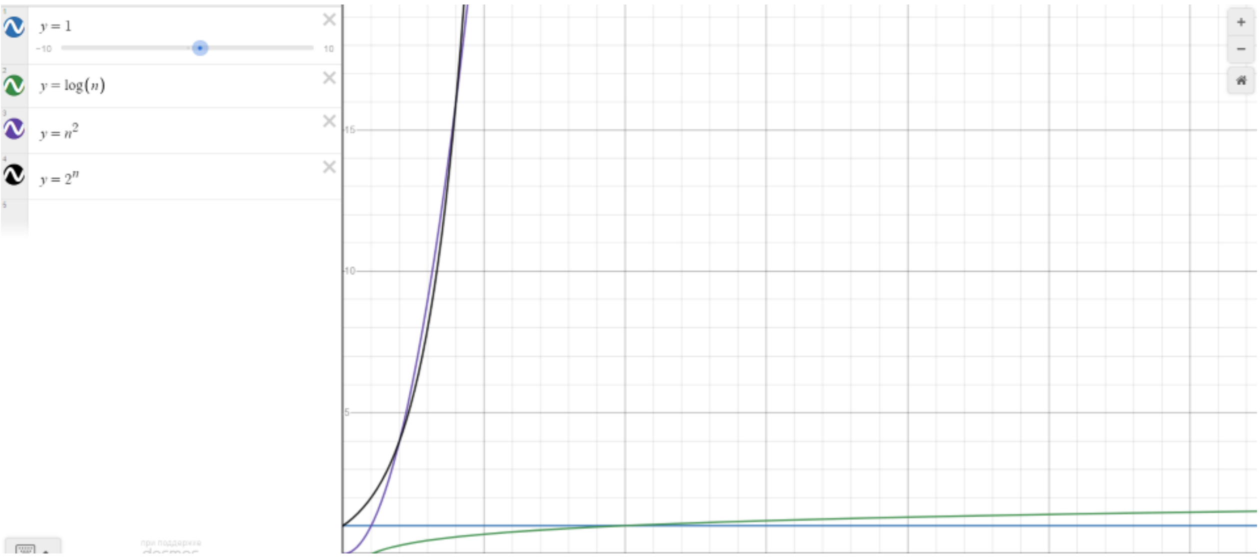
O(3logn):

```python
from random import randint as r

a = [-74, -68, -62, -59, -43, -30, -22, -13]
for i in range(3):
    x = a[r(0, len(a))-1]
    print(f'Ищем {x}')
    mx = len(a)
    mn = 0
    mid = (mx + mn) // 2
    while a[mid] != x:
        if a[mid] < x:
            mn = mid
        elif a[mid] > x:
            mx = mid
        mid = (mx + mn) // 2
    print(a[mid], mid)
```

Рисунок 6 – алгоритм со сложностью 3logn

Задание 3



При помощи графического калькулятора на сайте Desmos визуализируем зависимость сложности наших алгоритмов от количества элементов.

Как мы видим, O(1) постоянный, так как не зависит от количества элементов.

O(log(n)) возрастает медленно, в определенный момент обгоняя O(1).

O(n^2) и O(2^n) сначала идут почти наравне, но потом O(n^2) вырывается вперёд и растёт быстрее.

Список литературы.

1. Desmos: официальный сайт. – URL: https://www.desmos.com/calculator?lang=ru (дата обращения: 26.10.2022)