

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»  
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных  
систем

**Лабораторная работа №2.1**  
**Алгоритмы порождения комбинаторных объектов**  
по дисциплине: Дискретная математика

Выполнил: студент ПВ-233  
Мороз Роман Алексеевич

Проверил: Островский Алексей  
Мичеславович

Белгород 2024 г.

**Цель работы:** изучить основные комбинаторные объекты, алгоритмы их порождения, программно реализовать и оценить временную сложность алгоритмов.

## Задания

1. Реализовать алгоритм порождения подмножеств.

```
def generate_binary_vectors(n, current_vector=[], result=[]):
    if len(current_vector) == n:
        result.append(current_vector.copy())
        return
    for bit in (0, 1):
        current_vector.append(bit)
        generate_binary_vectors(n, current_vector, result)
        current_vector.pop()

    return result

def binary_vector_to_subset(binary_vector, original_set):
    subset = [element for i, element in
enumerate(original_set) if binary_vector[i] == 1]
    return subset

def generate_subsets_from_binary_vectors(binary_vectors,
original_set):
    subsets = [binary_vector_to_subset(vector, original_set)
for vector in binary_vectors]
    return subsets

n = 3
source = [i for i in range(1, n + 1)]
all_vectors = generate_binary_vectors(n)
print(all_vectors)
subsets = generate_subsets_from_binary_vectors(all_vectors,
source)
print(subsets)
```

```
"C:\Users\Мороз Роман\PycharmProjects\python_Project1\.venv\Scripts\python.exe" "C:\Users\М
[[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]
[[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]

Process finished with exit code 0
```

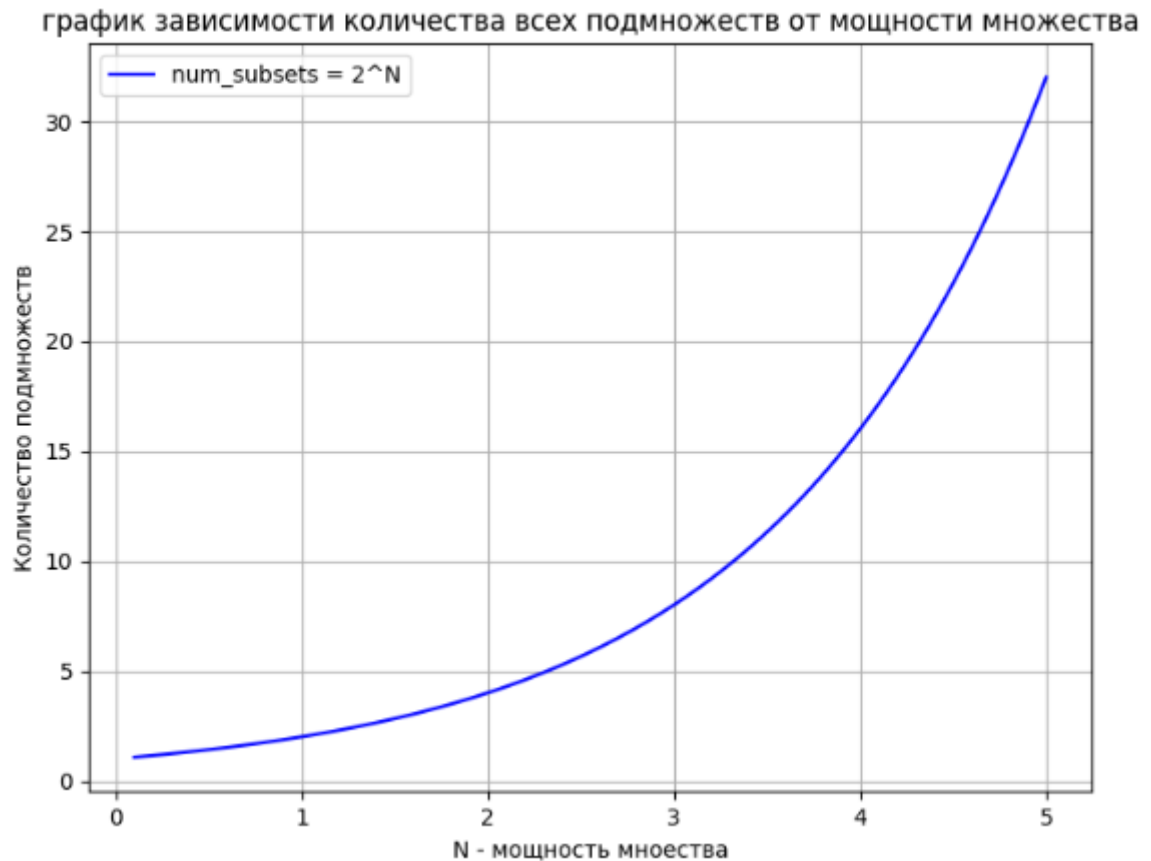
2. Построить график зависимости количества всех подмножеств от мощности множества.

Функция  $y = 2^N$ , где  $y$  – кол-во всех подмножеств, а  $N$  – мощность множества

```
import numpy as np
import matplotlib.pyplot as plt

N = np.linspace(0.1, 5, 500)
num_subsets = 2 ** N

plt.figure(figsize=(8, 6))
plt.plot(N, num_subsets, label='num_subsets = 2^N', color='b')
plt.xlabel('N - мощность мноества')
plt.ylabel('Количество подмножеств')
plt.title('график зависимости количества всех подмножеств от
мощности множества')
plt.grid(True)
plt.legend()
plt.show()
```



3. Построить графики зависимости времени выполнения алгоритмов п.1 на вашей ЭВМ от мощности множества.

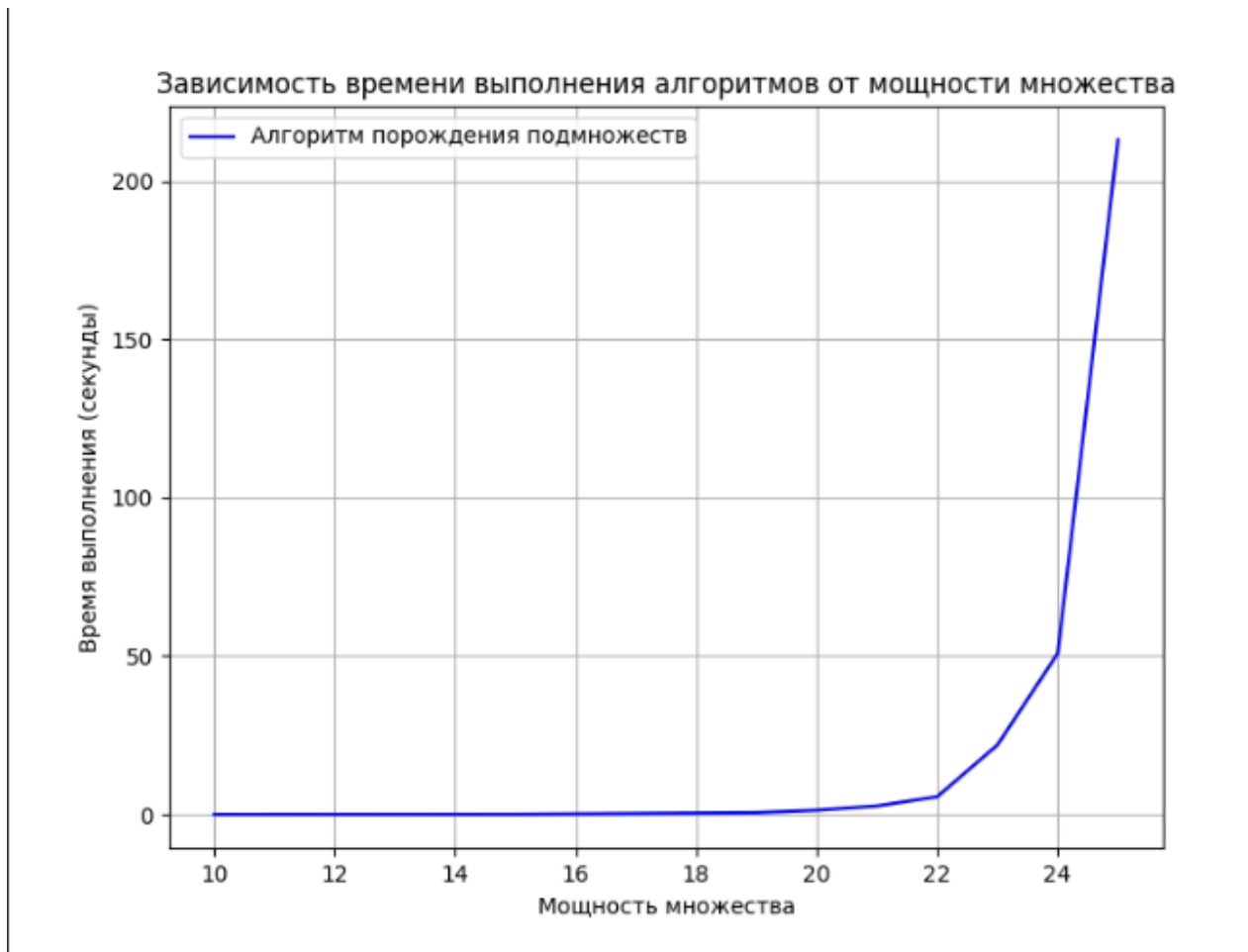
```
import matplotlib.pyplot as plt

powers = [10, 15, 19, 20, 21, 22, 23, 24, 25]

times = [0.00299, 0.0259, 0.557, 1.325, 2.656, 5.586, 21.957,
50.869, 213.0630]

plt.figure(figsize=(8, 6))
plt.plot(powers, times, label='Алгоритм порождения подмножеств',
color='b')

plt.xlabel('Мощность множества')
plt.ylabel('Время выполнения (секунды)')
plt.title('Зависимость времени выполнения алгоритмов от мощности
множества')
plt.grid(True)
plt.legend()
plt.show()
```



4. Определить максимальную мощность множества, для которого можно получить все подмножества не более чем за час, сутки, месяц, год на вашей ЭВМ.

Час = 3600 секунд

Сутки = 86400 секунд

Год = 31536000 секунд

Количество всех различных подмножеств  $n$ -элементного множества равно  $2^n$

Если мощность множества увеличим на единицу, то количество всех подмножеств увеличится в два раза.

Если мощность множества увеличим на  $k$ , то количество всех подмножеств увеличится в  $2^k$  раз.

Если мощность множества увеличим на  $k$ , то время порождения всех подмножеств увеличится не менее, чем в  $2^k$  раз.

За 50 секунд порождаются все подмножества  $2^4$  элементного множества

Для порождения всех подмножеств  $2^{4+6} = 2^{10}$  -элементного множества ему понадобится менее, чем  $2^6 = 3600$  секунд, т. е. менее часа.

Для порождения всех подмножеств  $(2^{4+6})^{+4}$ -элементного множества ему понадобится не менее, чем  $2^4 = 16$  часов, т. е. значительно менее суток.

Для порождения всех подмножеств  $((2^{4+6})^{+4})^{+5}$ -элементного множества ему понадобится менее, чем  $\frac{3}{4} * 2^5 = 24$ , т. е. значительно менее месяца.

*(Так как за менее чем 16 часов =  $\frac{3}{4}$  суток выполняется порождение всех подмножеств  $(2^{4+6})^{+4}$ -элементного множества)*

Для порождения всех подмножеств  $((((2^{4+6})^{+4})^{+5})^{+4})$ -элементного множества ему понадобится менее, чем  $\frac{3}{4} * 2^4 = 12$  месяцев.

5. Определить максимальную мощность множества, для которого можно получить все подмножества не более чем за час, сутки, месяц, год на ЭВМ, в 10 и в 100 раз быстрее вашей.

**ЭВМ в 10 раз быстрее. За 5 секунд порождаются все подмножества  $2^4$  элементного множества**

$$5 * 2^k = 3600$$

$$2^k = 720$$

*$K = 9$ , значит менее чем за час можно породить все подмножества  $9+24 = 33$  элементного множества*

$$5 * 2^k = 86400$$

$$2^k = 17280$$

*$K = 14$ , значит менее чем за сутки можно породить все подмножества 38 элементного множества*

$$5 * 2^k = 31536000$$

$$2^k = 6307200$$

*$K = 22$ , значит менее чем за год можно породить все подмножества 46 элементного множества*

**ЭВМ в 100 раз быстрее. За 0,5 секунд порождаются все подмножества 24 элементного множества**

$$0,5 * 2^k = 3600$$

$$2^k = 7200$$

***K = 12, значит менее чем за час можно породить все подмножества 36 элементного множества***

$$0,5 * 2^k = 86400$$

$$2^k = 172800$$

***K = 17, значит менее чем за сутки можно породить все подмножества 41 элементного множества***

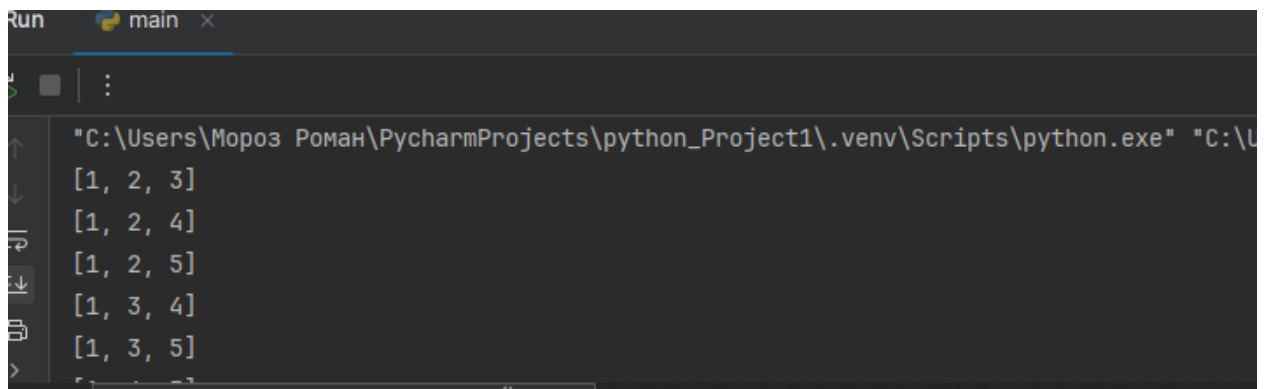
$$0,5 * 2^k = 31536000$$

$$2^k = 63072000$$

***K = 25, значит менее чем за год можно породить все подмножества 49 элементного множества***

6. Реализовать алгоритм порождения сочетаний.

```
def get_combinations(combination, k, original_set, i=0, b=0):  
    if i == k:  
        print(combination)  
    else:  
        for x in range(b, len(original_set) - k + i + 1):  
            combination[i] = original_set[x]  
            get_combinations(combination, k, original_set, i +  
1, x + 1)
```



Run main x

```
"C:\Users\Мороз Роман\PycharmProjects\python_Project1\.venv\Scripts\python.exe" "C:\U  
[1, 2, 3]  
[1, 2, 4]  
[1, 2, 5]  
[1, 3, 4]  
[1, 3, 5]  
[1, 4, 5]
```

7. Построить графики зависимости количества всех сочетаний из n по k от k при n= (5, 6, 7, 8, 9).

```

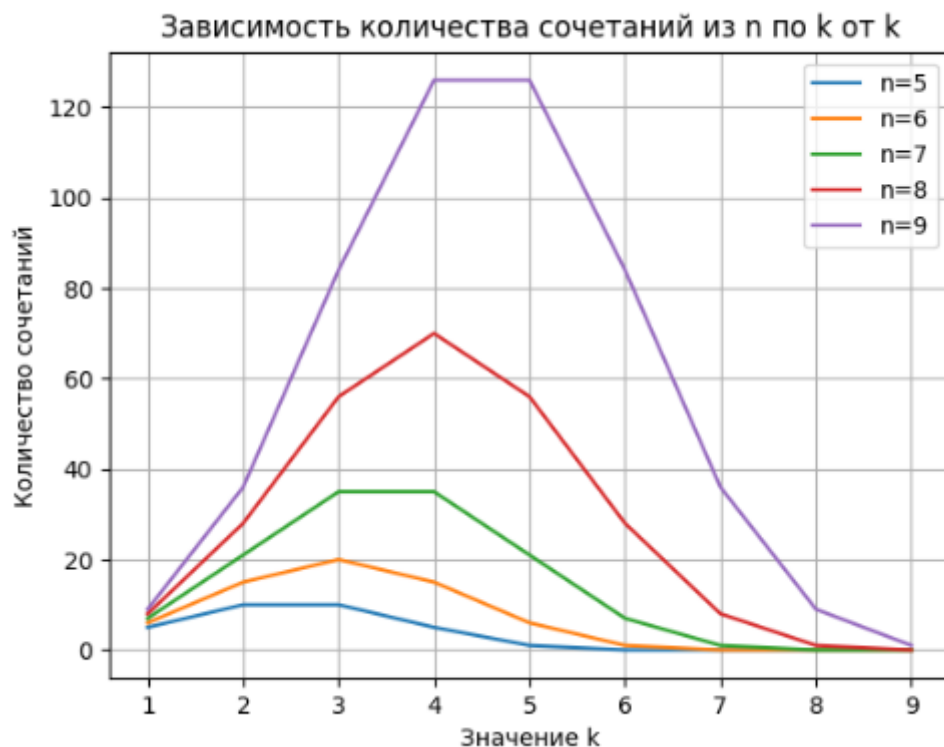
import matplotlib.pyplot as plt
from math import comb

n_values = [5, 6, 7, 8, 9]
k_values = range(1, 10)

for n in n_values:
    combinations = [comb(n, k) for k in k_values]
    plt.plot(k_values, combinations, label=f'n={n}')

plt.xlabel('Значение k')
plt.ylabel('Количество сочетаний')
plt.title('График зависимости количества сочетаний из n по k от k')
plt.legend()
plt.grid(True)
plt.show()

```



8. Реализовать алгоритм порождения перестановок.

```

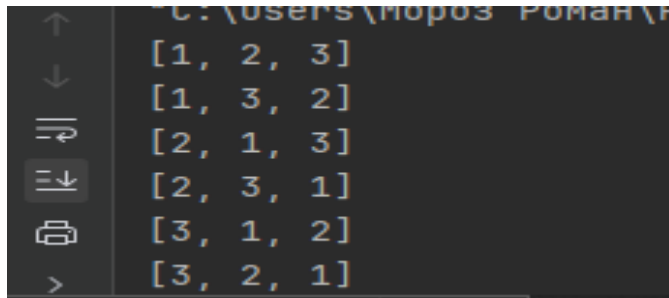
def del_el(original_set, element):
    if element in original_set:
        pos = original_set.index(element)
        s_copy = original_set.copy()
        s_copy.pop(pos)
    return s_copy

def get_permutations(original_set, index, permutation,

```



```
size_set):
    for x in range(len(original_set)):
        permutation[index] = original_set[x]
        if index == size_set - 1:
            print(permutation)
        else:
            get_permutations(del_el(original_set,
original_set[x]), index + 1, permutation, size_set)
```



```
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]
[3, 2, 1]
```

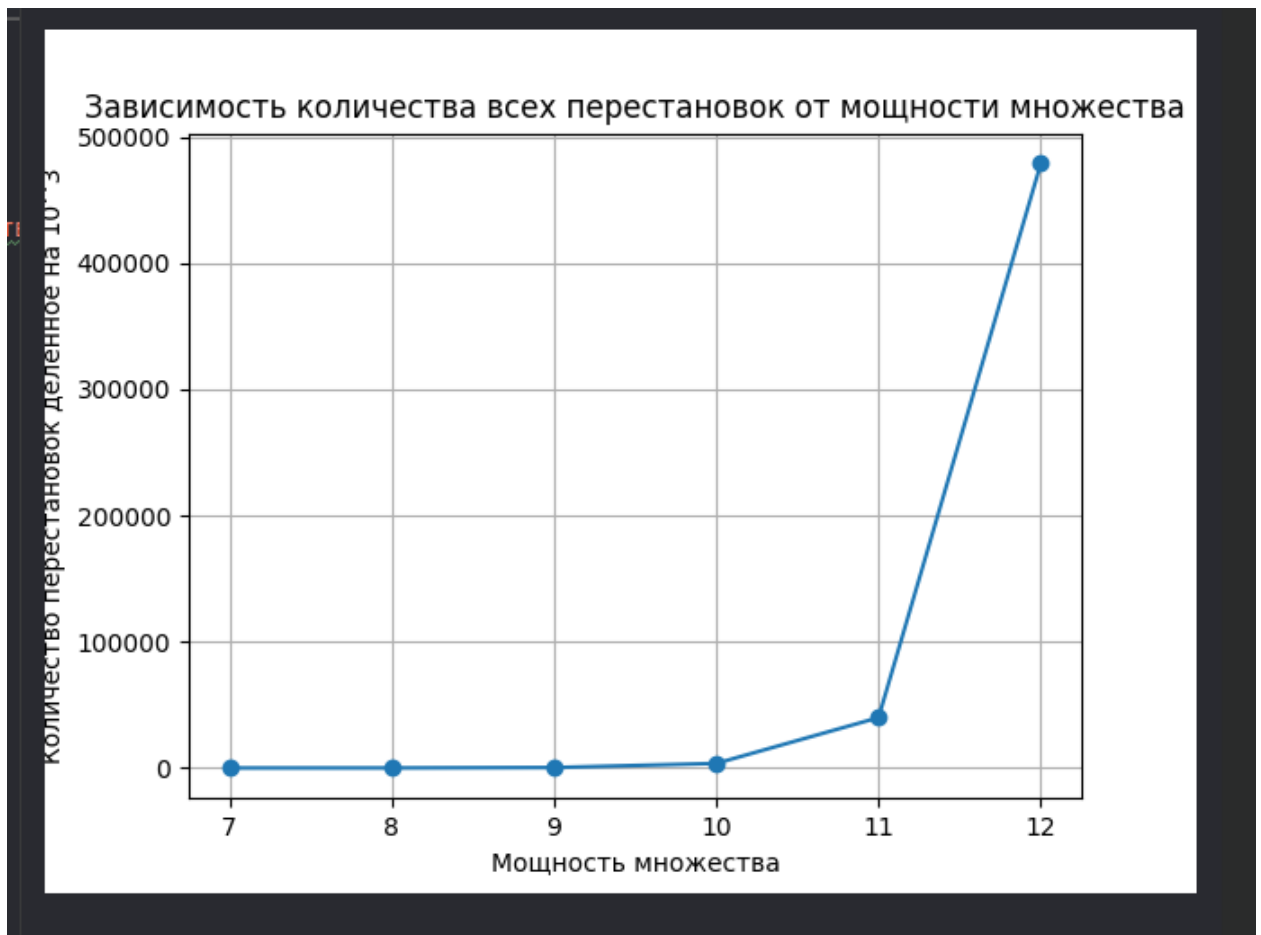
9. Построить график зависимости количества всех перестановок от мощности множества.

Количество перестановок рассчитывается по формуле  $P = n!$  где  $n$ -мощность множества

```
import matplotlib.pyplot as plt
from math import factorial

x = list(range(7, 13))
y = [factorial(i) / 10**3 for i in x]

plt.plot(x, y, marker='o')
plt.xlabel('Мощность множества')
plt.ylabel('Количество перестановок деленное на 10^3')
plt.title('Зависимость количества всех перестановок от мощности множества')
plt.grid(True)
plt.show()
```



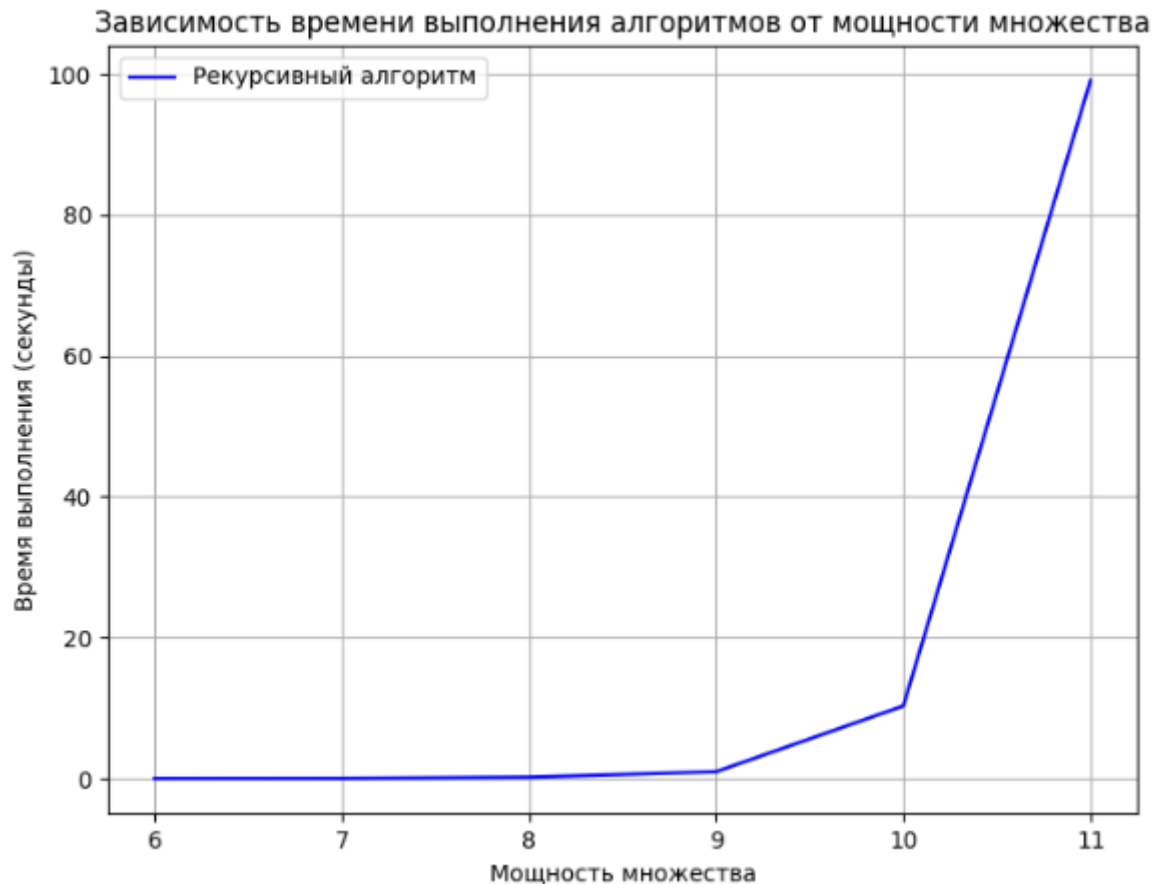
10. Построить графики зависимости времени выполнения алгоритма п.8 на вашей ЭВМ от мощности множества.

```
import matplotlib.pyplot as plt

powers = [6, 7, 8, 9, 10, 11]

times = [0.00199, 0.0099, 0.1795, 0.9573, 10.2934, 99.1028]

plt.figure(figsize=(8, 6))
plt.plot(powers, times, label='Рекурсивный алгоритм', color='b')
plt.xlabel('Мощность множества')
plt.ylabel('Время выполнения (секунды)')
plt.title('Зависимость времени выполнения алгоритмов от мощности множества')
plt.grid(True)
plt.legend()
plt.show()
```



11. Определить максимальную мощность множества, для которого можно получить все перестановки не более чем за час, сутки, месяц, год на вашей ЭВМ.

Час = 3600 секунд

Сутки = 86400 секунд

Год = 31536000 секунд

Количество всех различных перестановок  $n$ -элементного множества  $M$  (количество способов упорядочивания множества) определяется формулой  $P = n!$ .

За 10 секунд находятся все перестановки множества мощностью 10.

Если мощность множества увеличить на единицу, то количество перестановок увеличится в 11 раз, следовательно, времени на порождение всех перестановок увеличится не менее, чем в 11 раз.

Если мощность множества увеличить на 2, то количество перестановок увеличится в  $11*12=132$  раза, следовательно, времени на порождение всех перестановок увеличится не менее, чем в 132 раза.

*Значит менее чем за час ЭВМ может найти все перестановки множества мощностью 12 – примерно  $10 * 132 = 1320$  секунд*

А если мощность множества увеличить на 3, то количество перестановок увеличится в  $11*12*13=1716$  раз, следовательно, времени на порождение всех перестановок увеличится не менее, чем в 1716 раз.

*Значит менее чем за сутки ЭВМ может найти все перестановки множества мощностью 13 – примерно  $10 * 1716 = 17160$  секунд*

А если мощность множества увеличить на 5, то количество перестановок увеличится в  $11*12*13*14*15= 360360$  раз 7207200

*Значит менее чем за год ЭВМ может найти все перестановки множества мощностью 15 – примерно  $10 * 360360 = 3603600$  секунд*

12. Определить максимальную мощность множества, для которого можно получить все перестановки не более чем за час, сутки, месяц, год на ЭВМ, в 10 и в 100 раз быстрее вашей.

**ЭВМ в 10 раз быстрее. За 1 секунду порождаются все перестановки 10 элементного множества**

Если мощность множества увеличить на 3, то количество перестановок увеличится в  $11*12*13=1716$  раз, следовательно, времени на порождение всех перестановок увеличится не менее, чем в 1716 раза.

*Значит менее чем за час ЭВМ в 10 раз мощнее может найти все перестановки множества мощностью 13 – примерно  $1 * 1716 = 1716$  секунд*

А если мощность множества увеличить на 4, то количество перестановок увеличится в  $11*12*13*14= 24024$  раз, следовательно, времени на порождение всех перестановок увеличится не менее, чем в 24024раз.

*Значит менее чем за сутки ЭВМ в 10 раз мощнее может найти все перестановки множества мощностью 14 – примерно  $1 * 24024= 24024$  секунд*

А если мощность множества увеличить на 6, то количество перестановок увеличится в  $11*12*13*14*15*16=5765760$  раз

*Значит менее чем за год ЭВМ в 10 раз мощнее может найти все перестановки множества мощностью 14 – примерно  $1 * 5765760 = 5765760$  секунд*

**ЭВМ в 100 раз быстрее. За 0,1 секунду порождаются все перестановки 10 элементного множества**

Если мощность множества увеличить на 3, то количество перестановок увеличится в  $11*12*13*14=24024$  раз, следовательно, времени на порождение всех перестановок увеличится не менее, чем в 24024 раза.

*Значит менее чем за час ЭВМ в 100 раз мощнее может найти все перестановки множества мощностью 13 – примерно  $0,1 * 24024 = 2402,4$  секунд.*

А если мощность множества увеличить на 5, то количество перестановок увеличится в  $11*12*13*14*15=360360$  раз, следовательно, времени на порождение всех перестановок увеличится не менее, чем в 360360 раз.

*Значит менее чем за сутки ЭВМ в 100 раз мощнее может найти все перестановки множества мощностью 15 – примерно  $0,1 * 360360 = 36036$  секунд*

А если мощность множества увеличить на 7, то количество перестановок увеличится в  $11*12*13*14*15*16*17=98017920$  раз

*Значит менее чем за год ЭВМ в 100 раз мощнее может найти все перестановки множества мощностью 14 – примерно  $0,1 * 98017920 = 9801792$  секунд*

13. Реализовать алгоритм порождения размещений.

```
def del_el(original_set, element):
    if element in original_set:
        pos = original_set.index(element)
        s_copy = original_set.copy()
        s_copy.pop(pos)
    return s_copy

def get_arrangements(original_set, index, arrangement, k):
    for x in range(len(original_set)):
        arrangement[index] = original_set[x]
```

```

        if index == k - 1:
            print(arrangement)
        else:
            get_arrangements(del_el(original_set,
original_set[x]), index + 1, arrangement, k)

source = [1, 2, 3, 4]
k = 2
res = [0] * k
get_arrangements(source, 0, res, k)

```

```

"C:\Users\Мороз Роман\PycharmProjects\python_Proj
[1, 2]
[1, 3]
[1, 4]
[2, 1]
[2, 3]
[2, 4]

```

14. Построить графики зависимости количества всех размещений из  $n$  по  $k$  от  $k$  при  $n = (5, 6, 7, 8, 9)$

```

import matplotlib.pyplot as plt
from math import factorial

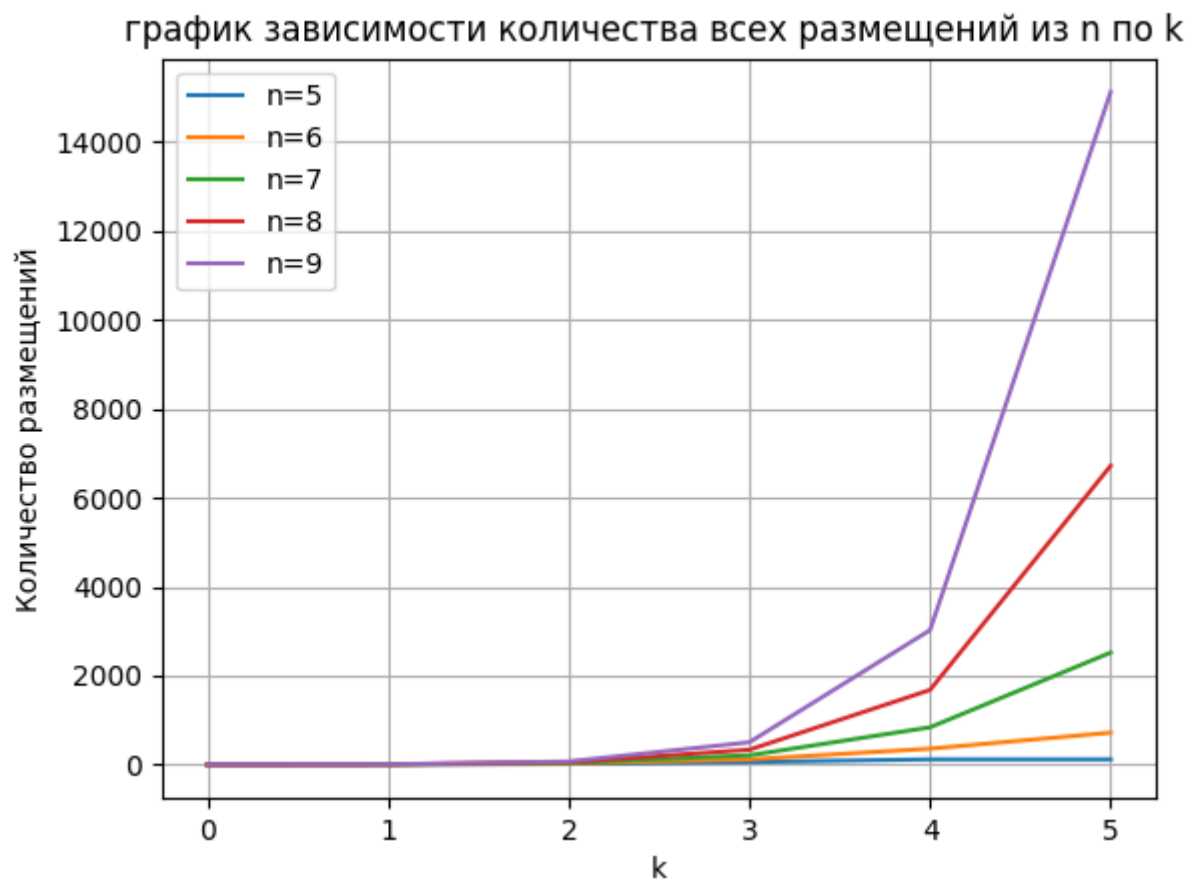
def count_arrangements(n, k):
    return factorial(n) / (factorial(n - k))

n_values = [5, 6, 7, 8, 9]
k_values = range(6)

for n in n_values:
    arrangements_count = [count_arrangements(n, k) for k in
k_values]
    plt.plot(k_values, arrangements_count, label=f'n={n}')

plt.xlabel('k')
plt.ylabel('Количество размещений')
plt.title('график зависимости количества всех размещений из n по
k ')
plt.legend()
plt.grid(True)
plt.show()

```



**Вывод:** изучили основные комбинаторные объекты, алгоритмы их порождения, программно реализовали и оценили временную сложность алгоритмов.