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

по дисциплине: Дискретная математика
тема: «Алгоритмы порождения комбинаторных объектов»

Выполнил: ст. группы ПВ-202
Аладиб язан

Проверил:

Рязанов Юрий Дмитриевич
Бондаренко Татьяна
Владимировна

Лабораторная работа № 2.1
«Алгоритмы порождения комбинаторных объектов»

Цель работы:

изучить основные комбинаторные объекты, алго-ритмы их порождения, программно реализовать и оценить временную сложность алгоритмов

Задания к работе:

1. Реализовать алгоритм порождения подмножеств.
2. Построить график зависимости количества всех подмножеств от мощности множества.
3. Построить графики зависимости времени выполнения алго-ритмов п.1 на вашей ЭВМ от мощности множества.
4. Определить максимальную мощность множества, для которо-го можно получить все подмножества не более чем за час, сутки, месяц, год на вашей ЭВМ.
5. Определить максимальную мощность множества, для которо-го можно получить все подмножества не более чем за час, сутки, месяц, год на ЭВМ, в 10 и в 100 раз быстрее вашей.
6. Реализовать алгоритм порождения сочетаний.
7. Построить графики зависимости количества всех сочетаний из n по k от k при n = (5, 7, 9).
8. Реализовать алгоритм порождения перестановок.
9. Построить график зависимости количества всех перестановок от мощности множества.
10. Построить графики зависимости времени выполнения алго-ритма п.8 на вашей ЭВМ от мощности множества.
11. Определить максимальную мощность множества, для кото-рого можно получить все перестановки не более чем за час, сутки, месяц, год на вашей ЭВМ.
12. Определить максимальную мощность множества, для кото-рого можно получить все перестановки не более чем за час, сутки, месяц, год на ЭВМ, в 10 и в 100 раз быстрее вашей.
13. Реализовать алгоритм порождения размещений.
14. Построить графики зависимости количества всех размеще-ний из n по k от k при n = (5, 7, 9).

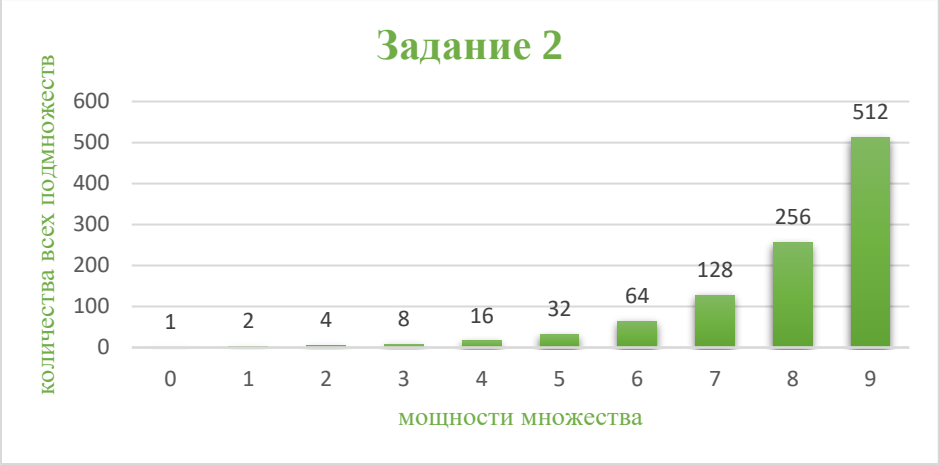
1. Реализовать алгоритм порождения подмножеств.

```
#include <stdio.h>
#include <stdint.h>
#include <malloc.h>

typedef struct
{
    int* values;
    size_t n;
    size_t size;
} sortset;

sortset create_sortset(size_t size)
{
    sortset res;
    res.n = 0;
    res.size = size;
    res.values = (int*)malloc(size * sizeof(int));
    return res;
}

void delete_sortset(sortset* p_a)
{
    free(p_a->values);
}

void append(sortset* p_a, int value)
{
    if (p_a->n == p_a->size)
    {
        p_a->size *= 2;
        p_a->values = (int*)realloc(p_a->values, p_a->size * sizeof(int));
    }
    (p_a->values)[p_a->n] = value;
    ++(p_a->n);
}

void append_values(sortset* p_a, size_t n)
{
    for (size_t i = 0; i < n; ++i)
    {
        int value;
        scanf("%i", &value);
        append(p_a, value);
    }
}

void print_sortset(sortset a)
{
```

```
    for (size_t i = 0; i < a.n; ++i)
        printf("%i ", a.values[i]);
}

void print_by_vect(sortset a, uint8_t* vect)
{
    printf("{");
    uint8_t f_to_clear = 0; // for nice output
    for (size_t i = 0; i < a.n; ++i)
        if (vect[i]) {
            f_to_clear = 1;
            printf("%i, ", a.values[i]);
        }
    if (f_to_clear)
        printf("\b\b");
    printf("}\n");
}

void print_subsets_inner(sortset a, size_t i, uint8_t* vect, size_t* p_n)
{
    if (!a.n) {
        (*p_n) = 1;
        printf("{}\n");
        return;
    }

    for (uint8_t x = 0; x <= 1; ++x) {
        vect[i] = x;
        if (i == a.n - 1) {
            (*p_n)++; // increases the number of subsets by 1
            print_by_vect(a, vect);
        }
        else
            print_subsets_inner(a, i + 1, vect, p_n);
    }
}

// returns the number of subsets of set a and prints them
size_t print_subsets(sortset a)
{
    uint8_t* vect = (uint8_t*)malloc(a.size);
    size_t res = 0;
    print_subsets_inner(a, 0, vect, &res);
    free(vect);
    return res;
}

int main()
{
    printf("Enter the size of set A: ");
    size_t size;
```

```
    scanf("%zu", &size);
    printf("Enter set A:\n");
    sortset a = create_sortset(size);
    append_values(&a, size);
    printf("Subsets of set A:\n");
    size_t n = print_subsets(a);
    printf("Total subsets: %zu", n);
    return 0;
}
```

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



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

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

максимальную мощность множества = $\log_2\left(\frac{\text{время(сек)}}{\text{Время вычисления подмножества на ЭВМ (сек)}}\right)$

Время вычисления подмножества на ЭВМ (сек) = $\frac{27.06}{32768} = 0{,}000825$

За час $= \log_2\left(\frac{60*60}{0{,}000825}\right) = \log_2\left(\frac{3600}{0{,}000825}\right) \approx 22$

За сутки $= \log_2\left(\frac{60*60*24}{0{,}000825}\right) = \log_2\left(\frac{86400}{0{,}000825}\right) \approx 26$

За месяц $= \log_2\left(\frac{60*60*24*30}{0{,}000825}\right) = \log_2\left(\frac{2592000}{0{,}000825}\right) \approx 31$

За год $= \log_2\left(\frac{60*60*24*30*12}{0{,}000825}\right) = \log_2\left(\frac{31104000}{0{,}000825}\right) \approx 35$

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.

чтобы получить требуемую скорость, мы делим Время вычисления подмножества на ЭВМ (сек) на количество раз.

в 10 раз быстрее :

За час $= \log_2\left(\frac{60*60}{0{,}000825/10}\right) = \log_2\left(\frac{3600}{0{,}0000825}\right) \approx 25$

За сутки $= \log_2\left(\frac{60*60*24}{0{,}000825/10}\right) = \log_2\left(\frac{86400}{0{,}0000825}\right) \approx 30$

За месяц $= \log_2\left(\frac{60*60*24*30}{0{,}000825/10}\right) = \log_2\left(\frac{2592000}{0{,}0000825}\right) \approx 35$

За год $= \log_2\left(\frac{60*60*24*30*12}{0{,}000825/10}\right) = \log_2\left(\frac{31104000}{0{,}0000825}\right) \approx 38$

в 100 раз быстрее :

За час $= \log_2\left(\frac{60*60}{0{,}000825/100}\right) = \log_2\left(\frac{3600}{0{,}00000825}\right) \approx 29$

За сутки $= \log_2\left(\frac{60*60*24}{0{,}000825/100}\right) = \log_2\left(\frac{86400}{0{,}00000825}\right) \approx 33$

За месяц $= \log_2\left(\frac{60*60*24*30}{0{,}000825/100}\right) = \log_2\left(\frac{2592000}{0{,}00000825}\right) \approx 38$

За год $= \log_2\left(\frac{60*60*24*30*12}{0{,}000825/100}\right) = \log_2\left(\frac{31104000}{0{,}00000825}\right) \approx 42$

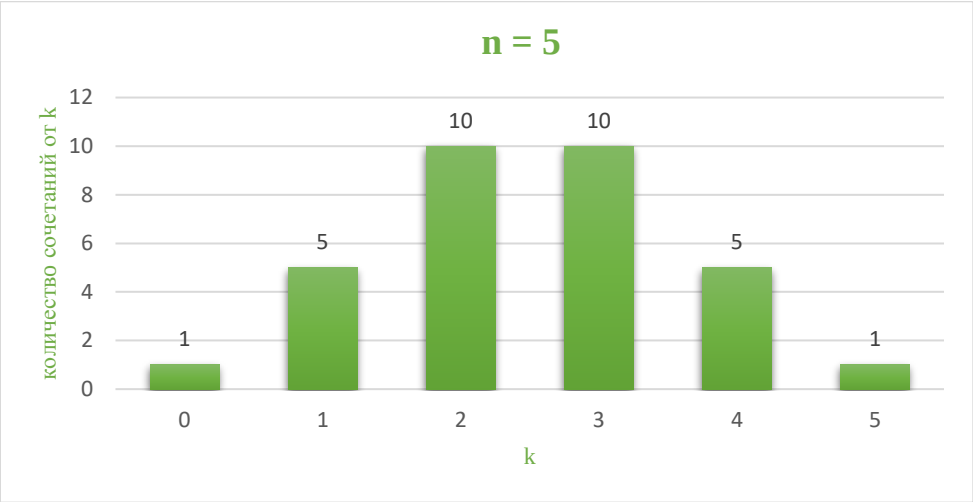
6. Реализовать алгоритм порождения сочетаний.

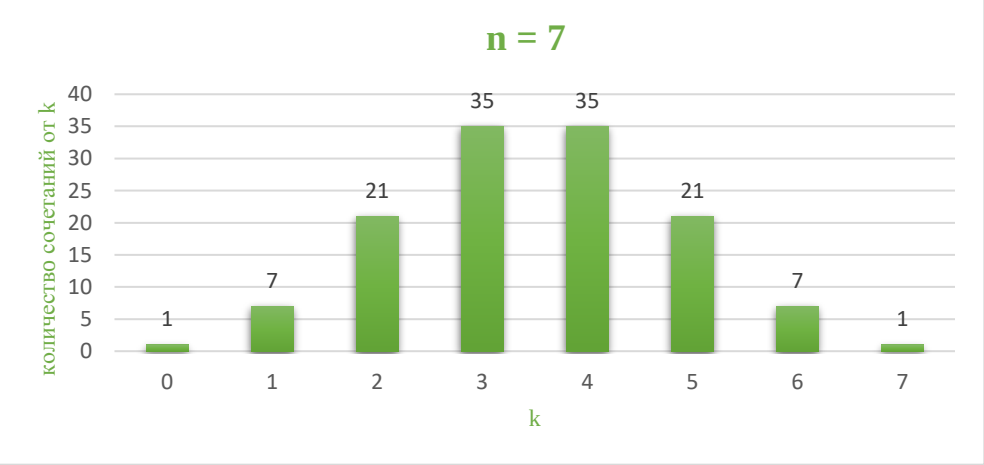
```
#include <stdio.h>
#include <stdint.h>
#include <malloc.h>
#include <locale.h>

typedef struct
{
    int* values;
    size_t n;
    size_t size;
} sortset;

sortset create_sortset(size_t size)
{
    sortset res;
    res.n = 0;
    res.size = size;
    res.values = (int*)malloc(size * sizeof(int));
    return res;
}

void delete_sortset(sortset* p_a)
{
    free(p_a->values);
}

void append(sortset* p_a, int value)
{
    if (p_a->n == p_a->size)
    {
        p_a->size *= 2;
        p_a->values = (int*)realloc(p_a->values, p_a->size * sizeof(int));
    }
    (p_a->values)[p_a->n] = value;
    ++(p_a->n);
}

void append_values(sortset* p_a, size_t n)
{
    for (size_t i = 0; i < n; ++i)
    {
        int value;
        scanf("%i", &value);
```

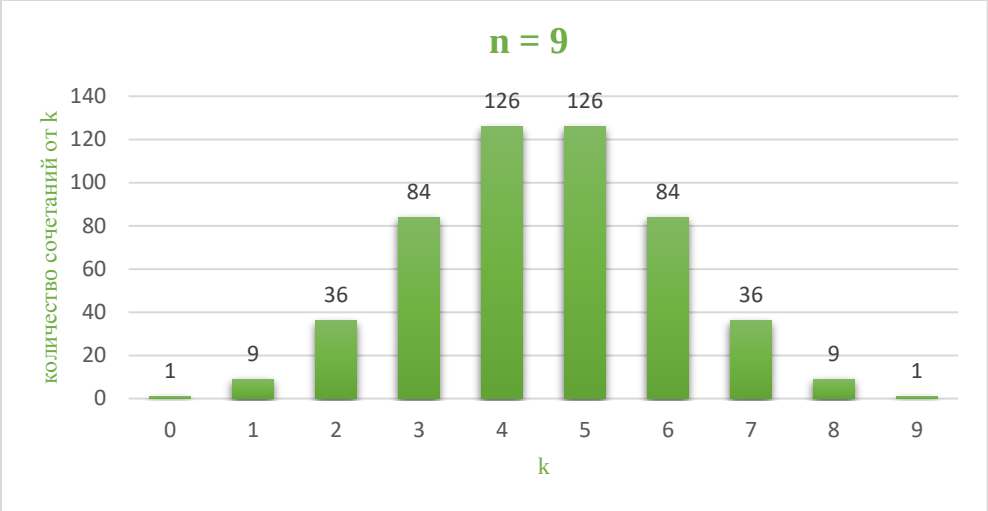
```
        append(p_a, value);
    }
}

void print_sortset(sortset a)
{
    if (!a.n) {
        printf("{}");
        return;
    }
    printf("{");
    for (size_t i = 0; i < a.n; ++i)
        printf("%i, ", a.values[i]);
    printf("\b\b}\n");
}

void print_combinations_inner(size_t i, size_t b_i, sortset a, // b_i - index of the
minimum element
                              size_t k, sortset C, size_t* p_n)
{
    for (size_t x_i = b_i; x_i <= a.n - k + i; ++x_i) {
        C.values[i] = a.values[x_i];
        if (i == k-1) {
            (*p_n)++; // increases the number of subsets by 1
            print_sortset(C);
        }
        else
            print_combinations_inner(i + 1, x_i + 1, a, k, C, p_n);
    }
}

// returns the number of subsets of set a, prints them
size_t print_combinations(sortset a, size_t k)
{
    if (!k) {
        printf("{}\n");
        return 1;
    }
    sortset C = create_sortset(k);
    C.n = k;
    size_t res = 0;
    print_combinations_inner(0, 0, a, k, C, &res);
    delete_sortset(&C);
    return res;
}

int main()
{
    printf("Enter the size of set A: ");
    size_t size;
    scanf("%zu", &size);
```

```
    printf("Enter set A:\n");
    sortset a = create_sortset(size);
    append_values(&a, size);
    for (size_t i = 0; i <= size; ++i) {
        printf("Combinations of set A of size %zu:\n", i);
        size_t n = print_combinations(a, i);
        printf("Total combinations of set A of size %zu: %zu\n\n", i, n);
    }
    delete_sortset(&a);
    return 0;
}
```

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







8. Реализовать алгоритм порождения перестановок.

```
#include <stdio.h>
#include <stdint.h>
#include <malloc.h>
#include <locale.h>

typedef struct
{
    int* values;
    size_t n;
    size_t size;
} sortset;

sortset create_sortset(size_t size)
{
    sortset res;
    res.n = 0;
    res.size = size;
    res.values = (int*)malloc(size * sizeof(int));
    return res;
}

void delete_sortset(sortset* p_a)
{
    free(p_a->values);
}

void append(sortset* p_a, int value)
{
    if (p_a->n == p_a->size)
```

```
    {
        p_a->size *= 2;
        p_a->values = (int*)realloc(p_a->values, p_a->size * sizeof(int));
    }
    (p_a->values)[p_a->n] = value;
    ++(p_a->n);
}

void append_values(sortset* p_a, size_t n)
{
    for (size_t i = 0; i < n; ++i)
    {
        int value;
        scanf("%i", &value);
        append(p_a, value);
    }
}

sortset copy_sortset(sortset a)
{
    sortset res = create_sortset(a.size);
    for (size_t i = 0; i < a.n; ++i)
        append(&res, a.values[i]);
    return res;
}

sortset diff_sortsets(sortset a, sortset b)
{
    sortset res = create_sortset(a.size);
    int i = 0;
    int j = 0;
    while (i < a.n && j < b.n)
    {
        if (a.values[i] == b.values[j])
        {
            ++i;
            ++j;
        }
        else if (a.values[i] > b.values[j])
        {
            ++j;
        }
        else
        {
            append(&res, a.values[i]);
```

```
            ++i;
        }
    }

    while (i < a.n)
    {
        append(&res, a.values[i]);
        ++i;
    }

    return res;
}

sortset diff_sortset_with_value(sortset a, int value)
{
    sortset s_value = create_sortset(1);
    append(&s_value, value);
    return diff_sortsets(a, s_value);
}

void print_sortset(sortset a)
{
    if (!a.n) {
        printf("{}");
        return;
    }
    printf("{");
    for (size_t i = 0; i < a.n; ++i)
        printf("%i, ", a.values[i]);
    printf("\b\b}\n");
}

void print_permutations_inner(sortset M, size_t i, sortset P, size_t* p_n)
{
    for (size_t x_i = 0; x_i < M.n; ++x_i) {
        P.values[i] = M.values[x_i];
        if (i == P.n - 1) {
            (*p_n)++; // increases the number of subsets by 1
            print_sortset(P);
        }
        else
            print_permutations_inner(diff_sortset_with_value(M, M.values[x_i]),
i+1, P, p_n);
    }
}
```

```
// returns the number of subsets of set a, prints them
size_t print_permutations(sortset a)
{
    if (!a.n) {
        printf("{}\n");
        return 1;
    }

    sortset P = create_sortset(a.n);
    P.n = a.n;
    sortset M_first = copy_sortset(a);
    size_t res = 0;
    print_permutations_inner(M_first, 0, P, &res);
    delete_sortset(&P);
    delete_sortset(&M_first);
    return res;
}

int main()
{
    printf("Enter the size of set A: ");
    size_t size;
    scanf("%zu", &size);
    printf("Enter set A:\n");
    sortset a = create_sortset(size);
    append_values(&a, size);
    printf("Permutations of set A:\n");
    size_t res = print_permutations(a);
    printf("Total permutations: %zu", res);
    delete_sortset(&a);
    return 0;
}
```

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

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

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

Количество перестановок множества = $\frac{\text{время(сек)}}{\text{Время перестановки подмножества на ЭВМ (сек)}}$

Время перестановки подмножества на ЭВМ (сек) = $\frac{146.04}{9!} = \frac{146.04}{362880} = 0{,}000402$

За час = $\frac{60*60}{0{,}000402} = \frac{3600}{0{,}000402} \approx 8955223$

N=10

N! = 3628800

За сутки = $\frac{60*60*24}{0{,}000402} = \frac{86400}{0{,}000402} \approx 214925373$

N=11

N! = 39916800

За месяц = $\frac{60*60*24*30}{0{,}000402} = \frac{2592000}{0{,}000402} \approx 6447761194$

N=13

N! = 6227020800

За год = $\frac{60*60*24*30*12}{0{,}000402} = \frac{31104000}{0{,}000402} \approx 77373134328$

N=14

N! = 87178291200

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.

чтобы получить требуемую скорость, мы делим Время перестановки подмножества на ЭВМ (сек) на количество раз.

в 10 раз быстрее :

За час = $\frac{60*60}{0{,}000402/10} = \frac{3600}{0{,}0000402} \approx 89552238$

N=11

N! = 39916800

За сутки $= \frac{60*60*24}{0{,}000402/10} = \frac{86400}{0{,}0000402} \approx 2149253731$

N=12

N! = 479001600

За месяц $= \frac{60*60*24*30}{0{,}000402/10} = \frac{2592000}{0{,}0000402} \approx 64477611940$

N=14

N! = 87178291200

За год $= \frac{60*60*24*30*12}{0{,}000402/10} = \frac{31104000}{0{,}0000402} \approx 773731343283$

N=14

N! = 87178291200

в 100 раз быстрее :

За час $= \frac{60*60}{0{,}000402/100} = \frac{3600}{0{,}00000402} \approx 895522388$

N=12

N! = 479001600

За сутки $= \frac{60*60*24}{0{,}000402/100} = \frac{86400}{0{,}000000402} \approx 21492537313$

N=13

N! = 6227020800

За месяц $= \frac{60*60*24*30}{0{,}000402/100} = \frac{2592000}{0{,}00000402} \approx 644776119402$

N=14

N! = 87178291200

За год $= \frac{60*60*24*30*12}{0{,}000402/100} = \frac{31104000}{0{,}00000402} \approx 7737313432835$

N=15

N! = 1307674368000

13. Реализовать алгоритм порождения размещений.

```
#include <stdio.h>
#include <stdint.h>
#include <malloc.h
```

```
#include <locale.h>

typedef struct
{
    int* values;
    size_t n;
    size_t size;
} sortset;

sortset create_sortset(size_t size)
{
    sortset res;
    res.n = 0;
    res.size = size;
    res.values = (int*)malloc(size * sizeof(int));
    return res;
}

void delete_sortset(sortset* p_a)
{
    free(p_a->values);
}

void append(sortset* p_a, int value)
{
    if (p_a->n == p_a->size)
    {
        p_a->size *= 2;
        p_a->values = (int*)realloc(p_a->values, p_a->size * sizeof(int));
    }
    (p_a->values)[p_a->n] = value;
    ++(p_a->n);
}

void append_values(sortset* p_a, size_t n)
{
    for (size_t i = 0; i < n; ++i)
    {
        int value;
        scanf("%i", &value);
        append(p_a, value);
    }
}

sortset copy_sortset(sortset a)
{
    sortset res = create_sortset(a.size);
    for (size_t i = 0; i < a.n; ++i)
        append(&res, a.values[i]);
    return res;
}
```

```
sortset diff_sortsets(sortset a, sortset b)
{
    sortset res = create_sortset(a.size);
    int i = 0;
    int j = 0;
    while (i < a.n && j < b.n)
    {
        if (a.values[i] == b.values[j])
        {
            ++i;
            ++j;
        }
        else if (a.values[i] > b.values[j])
        {
            ++j;
        }
        else
        {
            append(&res, a.values[i]);
            ++i;
        }
    }

    while (i < a.n)
    {
        append(&res, a.values[i]);
        ++i;
    }

    return res;
}

sortset diff_sortset_with_value(sortset a, int value)
{
    sortset s_value = create_sortset(1);
    append(&s_value, value);
    return diff_sortsets(a, s_value);
}

void print_sortset(sortset a)
{
    if (!a.n) {
        printf("{}");
        return;
    }
    printf("{");
    for (size_t i = 0; i < a.n; ++i)
        printf("%i, ", a.values[i]);
    printf("\b\b}\n");
}
```

```
void print_accommodations_inner(sortset M, size_t i, sortset A, size_t k, size_t*
p_n)
{
    for (size_t x_i = 0; x_i < M.n; ++x_i) {
        A.values[i] = M.values[x_i];
        if (i == k-1) {
            (*p_n)++; // increases the number of subsets by 1
            print_sortset(A);
        }
        else
            print_accommodations_inner(diff_sortset_with_value(M, M.values[x_i]),
i+1, A, k, p_n);
    }
}

// returns the number of subsets of set a, prints them
size_t print_accomodations(sortset a, size_t k)
{
    if (!k) {
        printf("{}\n");
        return 1;
    }

    sortset A = create_sortset(a.n);
    A.n = k;
    sortset M_first = copy_sortset(a);
    size_t res = 0;
    print_accommodations_inner(M_first, 0, A, k, &res);
    delete_sortset(&A);
    delete_sortset(&M_first);
    return res;
}

int main()
{
    printf("Enter the size of set A: ");
    size_t size;
    scanf("%zu", &size);
    sortset a = create_sortset(size);
    printf("Enter set A: \n");
    append_values(&a, size);
    for (size_t i = 0; i <= size; ++i) {
        printf("Accommodations of set A by %zu:\n", i);
        size_t n = print_accomodations(a, i);
        printf("Total accommodations of set A by %zu: %zu\n\n", i, n);
    }
    delete_sortset(&a);
    return 0;
}
```

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