Вопрос 1, задание 1:

Вопрос: Какие есть ограничения в программе?

Для ответа на этот вопрос рассмотрим, какие аргументы в принципе использует эта программа. Функция-обертка print_all_subsets задействует только один параметр, set – множество, на основе которого генерируются подмножества и которое может быть любым. Запускающаяся в ходе ее выполнения рекурсивная функция print_all_subsets_ использует три параметра: все то же исходное множество set, которое может быть любым, текущее значение вектора cur_vector и номер обрабатываемого разряда cur_digit. Эти аргументы уже имеют ограничения: cur_digit не должен превышать мощность исходного множества, как и длина двоичного вектора cur_vector. Но стартовое значение cur_digit всегда равно 1 (а вне функции-обертки print_all_subsets_ просто не используется) и после каждой итерации возрастает строго на 1, и если при равенстве cur_digit мощности множества set_size цепь рекурсивных вызовов прекращается, то больше этот параметр стать просто не может.

Таким образом, на предмет ограничений целесообразнее рассмотреть функцию преобразования вектора в подмножество, create_subset_by_binary_vector. И в нее действительно не помешает внести ограничение на длину двоичного вектора, vector: количество двоичных разрядов в нем не должно превышать размер множества set, на основе которого происходит генерация. Проверить это можно следующей строкой:

```
assert(1<<set.size > vector);
```

Для наглядности можем представить себе, что у нас есть исходное множество мощности 4, и два вектора, на основе которых мы должны сгенерировать подмножество. С помощью побитового сдвига сместив единицу на 4 влево, получим вектор 10000, который численно автоматически будет превышать все возможные векторы длины 4: векторы, больше или равные 10000, будут отбрасываться после проверки и вызывать исключение.

Единица					1
Единица, смещенная на мощность множества	1	0	0	0	0
Вектор 1		1	1	0	1
Вектор 2	1	0	0	1	0

И в конечном итоге исправленная функция будет выглядеть так (остальные части кода останутся неизменными):

```
//Возвращает подмножество множества set, заданное двоичным
вектором vector
ordered_array_set create_subset_by_binary_vector(ordered_array_set
set, int vector) {
    assert(1<<set.size > vector);
    ordered_array_set result =
ordered_array_set_create(set.capacity);
    for (int i = 0; i < set.capacity; i++) {
        if (vector >> i & 1) {
            ordered_array_set_insert(&result, set.data[i]);
        }
    }
    ordered_array_set_shrinkToFit(&result);
    return result;
}
```

Задание на защиту:

Для того, чтобы программа порождала подмножества в порядке неуменьшения их мощности, можно воспользоваться функцией порождения сочетаний: ведь сочетания – это и есть подмножества некоторого множества, но обладающие фиксированной мощностью. Перебирая k, по которым сочетания будут порождаться, в пределах от нуля до n, и запуская функцию их порождения, можно получить все подмножества заданного множества в порядке неуменьшения мощности.

```
//Выводит на экран все подмножества множества set в порядке
неубывания мощности
void print_all_subsets_power_increment(ordered_array_set set) {
    for (int k = 0; k <= set.size; k++) {
        print_all_combinations(k, set);
    }
}

int main() {
    int a[4] = {1, 2, 3, 4};
    ordered_array_set A = ordered_array_set_create_from_array(a,
4);
    print_all_subsets_power_increment(A);
}
```

Запустив этот код, получим следующий вывод:

```
"C:\Users\sovac\Desktop\дискретная математика\ДИСКРЕТКА ЛАБА 2.1\combination_creation.exe"
Empty set
{1}
{2}
{3}
{4}
{1 2}
{1 3}
{1 4}
{2 3}
{2 4}
{3 4}
{1 2 3}
{1 2 4}
{1 3 4}
{2 3 4}
{1 2 3 4}

Process finished with exit code 0
```

Как видим, подмножества действительно упорядочены по неубыванию мощностей.