- Обработка входных данных, программа должна работать как с двумя аргументами так и с любыми другими ***(argc >= 2***), на вход принимаются только числа (т.е. ./push_swap “1 4 2” == ./push_swap 1 4 2).
- Я делал этот проект через односвязный список (по алгоритму Бражника), поэтому следующим этапом у меня было создание и заполнение списка **A**. К этому же этапу я отношу обработку (это могут быть только числа, числа не должны повторяться и должны быть в диапазоне *int*) и индексацию листов, после которой вся работа алгоритма уже идет только с индексами.

```
typedef struct s_list
{
    int             content;
    int             mvs;
    int             ind;
    unsigned char   des;
    struct s_list   *next;
}           t_list;
```

Вот из чего состоял мой лист (***content*** – входные данный, ***mvs*** – количество шагов, для того что бы стать первым элементом в списке **A**(нужно в дальнейшем для работы алгоритма), ***ind*** – говорит нам о том каким по порядку элемент должен быть по завершению программы от *0* до *n*, где *n* количество элементов, ***des*** – я использовал для расстановки флагов(*true* и *false* или в моем случае *0* и *1*) и наконец ***next*** – следующий элемент списка)

- Для значений $n < 6$ отдельное решение
- И наконец сам алгоритм, на сколько я его понял его можно разделить на 2 шага

1. Найти в списке **A** упорядоченный ряд максимальной длинны (для этого нужно найти начало этого ряда и шаг, я для каждого элемента списка **A** пробегался с разными шагами и запоминал лучший вариант) на примерах:

```
A: 1 3 5 6 4 8 2 (индексы листов)
шаг = 1 и начинаем с 0 элемента rez: 1 2
шаг = 2 и начинаем с 0 элемента rez: 1 3 5 6 8
...
в первом случае у нас длинна ряда 2 во воторм 5
```

Как только нашли лучший случай расставляем *true* и *false*. Далее все элементы с *false* перекидываем в стек **B** (тут есть один момент, когда проверяется можно ли увеличить количество элементов с *true* если свапнуть 2 первых элемента в **A**, если можно, то свапаем иначе отправляем элемент в стек **B**)

2. Перед вторым шагом у нас есть упорядоченный список **A** и рандомный список **B**. Второй шаг — это, по сути, цикл из 2-х действий:

1) Находится «лучший» элемент в стеке **B**
2) «лучший» элемент отправляется в стек **A**

Цикл повторяется до тех пор, пока в **B** что-то есть.

«лучший» элемент – это элемент в стеке **B**, который за наименьшее количество шагов может быть перемещен в правильную позицию в стеке **A**.

Пару слов о там как найти «лучший» элемент, я это делал следующим образом:

- Расставлял ***mvs*** в стеке **A** от *0* в начале примерно до ≈ *n/2* d в середине, и *1* в конце (количество шагов, для того, что бы стать первым элементом в списке **A**)
- Затем расставлял ***mvs*** в стеке **B** от *1* в начале примерно до ≈ *n/2* d в середине, и *2* в конце (количество шагов, для того, что бы стать первым элементом в списке **A**)
- И последнее складывал нужные ***mvs***

```
A:      1 3 5 6 8 - индексы в A
mvs_A:  0 1 2 2 1 - значение mvs для каждого листа
(значения идут с 0 т.к. первый элемент уже в нужной позиции)

B:      2 4 - индексы в B
mvs_B:  1 2 - значение mvs для каждого листа
(значения идут с 1 т.к. первый элемент нужно совершить "pa" что бы оказаться в A)

Затем мы просто складываем нужные индексы т.е.:

mvs_B[2] = mvs_B[2] + mvs_A[3] = 2      т.к. 2 должна встать перед 3
mvs_B[4] = mvs_B[4] + mvs_A[5] = 4      т.к. 4 должна встать перед 5

Итого:
B:      2 4 - индексы в B                       |
mvs_B:  2 4 - значение mvs для каждого листа    |-> лучший элемен - 2
```

Ну и в самом конце в большинстве случаев придется прокрутить стек **A** до 0 элемента.

```
A: 1 3 5 6 8
B: 2 4

что бы переместить 2 между 1 и 3 : ra, pa - 2 действия
что бы переместить 4 между 3 и 5 : ra, (ra, rb), pa - 4(3 если сделать rr, но это я не учитывал) действия

A: 2 3 5 6 8 1
B: 4

что бы переместить 4 между 3 и 5 : ra, ra, pa - 3 действия

A: 4 5 6 8 1 2 3
в конце може понадобится прокрутить стек A(rra, rra, rra) |-> A: 1 2 3 4 5 6 8
```

По итогу этот алгоритм дает решений для 100 элементов примерно на 570 действий, а для 500 элементов около 5300 действий.