Метод итераций

Вычислим C из n по k:
$C_n^k = \frac{n!}{k!*(n-k)!} = \frac{(n-k+1)*(n-k+2)*...*n}{k*(k-1)*...*1}$
Оценка сложности: $k * O(T(n))$, где $T(n)$ – оценки одного шага

$$\begin{array}{ccc} X_1 & A_1 & C_n^1 \\ X_2 & A_1 * A_2 & C_n^2 \\ ... & \cdots & ... \\ X_k & A_1 * A_2 * ... * A_k & C_n^k \end{array}$$

Задача: проверить существует ли сумма элементов подпоследовательности численно равная элементу последовательности, который больше каждого элемента такой подпоследовательности.

```
for (i: 1..n) {
s=0;
for(j:i..n)
{ s+=a[j];
if (S>D) return 1; } }
```

Оценки сложности: $O(n) + O(n-1) + ... + O(1) = O(\sum_{i=1}^{n} i = O(\frac{n*(n+1)}{2}) = O(n^2)$

Более выгодный алгоритм:

```
for (i: 1..n)
s=0;
for(j:i..n)
s+=a[j];
if (S>D) return 1;
if (S<0) i=j; break
```

Оценка сложности: $O(i_1) + O(i_2) + ... + O(i_k) = O(n)$

Задача: найти общие элементы двух упорядоченных массивов.

```
i=1;
j=1;
k=i+j-1;
C[k]=min(ai,bj);
if (a[i]<=b[j]) i++;
else j++
```

Задача: проверить содержится ли элемент X в упорядоченном массиве.
Бинарный поиск: Сравним $a_{\frac{n}{2}}$ и X:

1. если $a_k < X$ смотрим правую часть

2. если $a_k > X$ смотрим левую часть

3. если $a_k = X$ нашли решение

Ищем до тех пор пока либо не нашли решение, либо пока рассматриваемый промежуток стал по модулю меньше 1. Сложность алгоритма $O(\frac{n}{2^k})$.

Задача: отсортировать массив (merge sort).
Делим массив на 2.
Делим подмассив на 2 и т.д.
Сортируем минимальную единицу.
Поочередно сливаем массивы в один.

1. 2 массива для n/2 $O(n)$

2. 4 массива для n/4 $O(n/2) + O(n/2) = O(n)$

3. 8 массивов для n/8 $O(n/4) + O(n/4) + O(n/4) + O(n/4) = O(n)$

4. n массивов длины 1 $O(n)$