**Линейная**

Если в массиве N элементов, то основной цикл for выполнится ровно N раз.

for (size\_t i = 0; i < arr.size(); ++i) { // Этот цикл выполнится N раз

// ... тело цикла ...

}

В цикле:

if (arr[i] > maxElement) { // 1 операция сравнения

maxElement = arr[i]; // 1 операция присваивания

}

Общее количество операций T(N) - это произведение количества операций за одну итерацию (K) на общее количество итераций (N).  
T(N) = K \* N

**Логарифмическая**

**Расчет в лучшем случае: T(N) = K (или O(1))**

Шаг 1: Определяем количество шагов (итераций цикла).

Лучший случай - элемент находится ровно в середине массива

вычисляем mid, делаем одно сравнение sorted\_arr[mid] == target. Это константное число операций K

T(N) = K \* 1 = K

**Расчет в худшем случае: T(N) = K \* log(N)**

Худший случай — искомого элемента нет в массиве, или он находится на самом "дне" рекурсии.

**Факториальная сложность (N!)**

Алгоритмы генерации всех перестановок N элементов

**void** generatePermutations(vector<**int**>& arr, **int** start) {

**if** (start == arr.size() - 1) { *// 1 операция сравнения*

printArray(arr); *// K операций печати*

**return**; *// 1 операция возврата*

}

**for** (**int** i = start; i < arr.size(); i++) { *// Цикл выполнится n раз*

swap(arr[start], arr[i]); *// 1 операция обмена*

generatePermutations(arr, start + 1); *// Рекурсивный вызов*

swap(arr[start], arr[i]); *// 1 операция обратного обмена*

}

}

**Подсчет операций:**

* На уровне 0: N рекурсивных вызовов
* На уровне 1: каждый из N вызовов порождает (N-1) вызовов
* На уровне k: каждый вызов порождает (N-k) вызовов

**T(N) = N × (N-1) × (N-2) × ... × 1 = N!**

**Экспоненциальная сложность (K^N)**

Алгоритмы генерации всех подмножеств массива. Каждый элемент может быть включен или не включен в подмножество.

**void** generateSubsets(vector<**int**>& arr, **int** index, vector<**int**>& current) {

**if** (index == arr.size()) { *// 1 операция сравнения*

printSubset(current); *// K операций печати*

**return**; *// 1 операция возврата*

}

*// Вариант 1: не включаем элемент*

generateSubsets(arr, index + 1, current); *// 1 рекурсивный вызов*

*// Вариант 2: включаем элемент*

current.push\_back(arr[index]); *// 1 операция добавления*

generateSubsets(arr, index + 1, current); *// 1 рекурсивный вызов*

current.pop\_back(); *// 1 операция удаления*

}

**Подсчет операций:**

* На каждом уровне рекурсии: 2 варианта (включить/не включить элемент)
* Глубина рекурсии: N уровней
* Общее количество листьев дерева рекурсии: 2^N

**T(N) = K × 2^N, где K = 2**

**Полиномиальная сложность (N^K)**

Алгоритм поиска пары элементов с заданной суммой

**bool** findPairWithSum(vector<**int**>& arr, **int** target) {

**for** (size\_t i = 0; i < arr.size(); i++) { *// Внешний цикл выполнится N раз*

**for** (size\_t j = i + 1; j < arr.size(); j++) { *// Внутренний цикл выполнится в среднем N/2 раз*

**if** (arr[i] + arr[j] == target) { *// 1 операция сложения + 1 сравнения = 2 операции*

**return** true; *// 1 операция возврата*

}

}

}

**return** false; *// 1 операция возврата*

}

**Подсчет операций:**

* Внешний цикл: N итераций
* Внутренний цикл: для каждой итерации внешнего в среднем N/2 итераций
* Операций в теле внутреннего цикла: 2 операции

**T(N) = N × (N/2) × 2 = N^2**

**Линейный логарифм (N × log(N))**

Сортировка слиянием

**void** mergeSort(vector<**int**>& arr, **int** left, **int** right) {

**if** (left < right) { *// 1 операция сравнения*

**int** mid = (left + right) / 2; *// 1 сложение + 1 деление = 2 операции*

mergeSort(arr, left, mid); *// Рекурсивный вызов левой половины*

mergeSort(arr, mid + 1, right); *// Рекурсивный вызов правой половины*

merge(arr, left, mid, right); *// N операций слияния*

}

}

**void** merge(vector<**int**>& arr, **int** left, **int** mid, **int** right) {

*// Слияние занимает ровно (right - left + 1) операций*

**for** (**int** i = left; i <= right; i++) { *// Цикл выполнится N раз на данном уровне*

*// ... операции слияния ... // Константное число операций K*

}

}

**Подсчет операций:**

* Глубина рекурсии: log(N) уровней (каждый раз делим массив пополам)
* На каждом уровне рекурсии: обрабатываем все N элементов при слиянии
* Операций слияния на уровне: N × K операций

**T(N) = N × log(N) × K**