**УСЛОВИЕ ЗАДАЧИ**

Нам нужно написать функцию, которая принимает отсортированный массив чисел и возвращает индекс найденного числа. Если индекс не найден, тогда возвращается -1.

Сразу уделим внимание на то, что длинна массива может быть любой. Массив может состоять из любых чисел и искомое число так же может быть любым.

Предположим у нас есть массив чисел от 1 до 100:

**const** numberArray [  
 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,  
 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,  
 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,  
 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,  
 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,  
 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,  
 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,  
 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96,  
 97, 98, 99, 100]

**ЛИНЕЙНЫЙ ПОИСК**

**Линейный поиск** — это простой алгоритм поиска элемента в структуре данных (например в массиве), который последовательно проверяет каждый элемент в структуре данных на совпадение с целевым значением. Он начинает с первого элемента и продолжает проверку до тех пор, пока не будет найден искомый элемент или пока не закончится весь набор данных.

Вот пример кода линейного поиска:

int linearSearch (const std::vector<int>& arr, int target) {

for (int i = 0; i < arr.size(); i++) {

if (arr[i] == target) {

return i; // Элемент найден

}

}

return -1; // Элемент не найден

}

Если в качестве target мы передадим 1, тогда нам потребуется одна итерация цикла, чтобы найти число.

Если в качестве target мы передадим 100, тогда нам потребуется 100 итераций цикла, чтобы найти число.

В данном алгоритме сложность **O(n)**. Такую сложность так же называют линейной сложностью.

Такой поиск крайне неэффективный при работе с большим набором данных.

### БИНАРНЫЙ ПОИСК

**Бинарный поиск** гораздо более эффективный в сравнении с **линейным поиском**.

Бинарный поиск основан на идее деления данных на половины и последующем поиске в одной из них с последующим делением.

#### **Принцип бинарного поиска:**

Предположим, что в нашем отсортированном списке чисел от 1 до 100 мы будем искать число 87.

**Первое действие:**

* Разделим наш массив пополам. У нас получилось 2 массива one и two. В one числа от 1 до 50, а в two с 51 до 100.

**const** one = [  
 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,   
 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,   
 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,  
 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,   
 41, 42, 43, 44, 45, 46, 47, 48, 50]  
  
**const** two = [  
 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,  
 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,  
 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,  
 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,  
 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

* Посмотрим последний элемент массива one. 49 больше, чем 87? Нет.
* Тогда смотрим последний элемент массива two. 100 больше, чем 87? Да, значит искать число будет массиве two. При первом действии мы уже отсекли половину данных.

**Второе действие:**

* Массив two снова раздели на два. Получаем массив three c числами от 50 до 75 и four с числами от 76 до 100.

**const** three = [  
 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,  
 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71,  
 72, 73, 74, 75 ]  
  
**const** four = [  
 76, 77, 78, 79, 80, 81, 82, 83, 84, 95, 85,  
 86, 87, 88, 89, 90, 91, 92, 93, 94, 96, 97,  
 98, 99, 100]

* Посмотрим последний элемент массива three. 75 больше, чем 87? Нет.
* Тогда смотрим последний элемент массива four. 100 больше, чем 87? Да, значит искать число будет в массиве four. При втором действии мы отсекли еще половину данных.

**Третье действие:**

* Массив four снова раздели на два. Получаем массив five с числами от 76 до 87 и массив six с числами от 88 до 100.

**const** five = [  
 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, ]  
  
**const** six = [  
 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

* Посмотрим последний элемент массива five.
* 87 больше, чем 87? Нет. Они равны. **Мы нашли наше число на третьем действии.**

Для линейного поиска максимальное количество итераций совпадает с размером массива.

С бинарным поиском дело обстоит иначе. Если список состоит из 100 эле­ментов, потребуется не более 7 попыток. Для списка из 4 миллиардов эле­ментов потребуется не более 32 попыток. Впечатляет, верно?

Бинарный поиск выполняется за логарифмическое время и имеет сложность **O(log n)**.