Поиск элемента по ключу. Линейный поиск. Бинарный поиск. Экспоненциальный поиск *(galloping search)*.

Поиск элемента по ключу

- Дана последовательность из n ключей: $a_1, a_2, ..., a_n$
- Требуется найти номер *(индекс)* элемента, совпадающего с заданным ключом **key**

Линейный поиск

```
int LinearSearch(int A[], int n, int key) {

  for (int i = 1; i < n; i++) {

    if (A[i] == key)

      return i;

  }

  return -1;

}
```

Принцип работы:

- Просматриваем элементы, начиная с первого, и сравниваем ключи.
- В худшем случае искомый элемент находится в конце массива или отсутствует.
- Количество операций в худшем случае *(worst case)*: **T(n) = O(n)**.

Бинарный поиск *(binary search)*

- Имеется **упорядоченная** последовательность ключей

$$a_1 \le a_2 \le \ldots \le a_i \le \ldots \le a_n$$

- Требуется найти позицию элемента, ключ которого совпадает с заданным ключом *key*
- Бинарный поиск *(binary search)*
  1. Если центральный элемент **равен** искомому, **конец** алгоритма
  2. Если центральный элемент **меньше**, делаем текущей **правую** половину массива
  3. Если центральный элемент **больше**, делаем текущей **левую** половину массива

```
int BinarySearch(int A[], int n, int key) {
  // Устанавливаем начальные границы поиска
  int low = 1;        // нижняя граница (обычно начинают с 0)
  int high = n;       // верхняя граница

  // Пока границы не сомкнутся
  while (low <= high) {
    // Вычисляем средний элемент
    int mid = (low + high) / 2;  /* Внимание: возможно переполнение при
больших low и high! */
                                 /* Безопасная альтернатива: low + (high - low)
/ 2 */

    // Проверяем средний элемент
    if (A[mid] == key) return mid;          // Если нашли искомый элемент
                                            // Возвращаем его индекс
    else if (key > A[mid]) low = mid + 1; // Если искомое значение больше
среднего, сдвигаем нижнюю границу вправо
    else high = mid - 1;           // Если искомое значение меньше среднего,
сдвигаем верхнюю границу влево
  }

  // Если элемент не найден
  return -1;
}
```

Бинарный поиск неэффективно использует кеш-память процессора: доступ к элементам массива непоследовательный (прыжки по массиву)

Экспоненциальный или поиск от края *(galloping search)*

- Задан отсортированный массив A[n]
- Алгоритм поиска от края проверяет ключи с индексами

  $1, 3, 7, 15, \ldots, 2^i - 1, \ldots$
- Проверка идёт до тех пор, пока не будет найден элемент

  $A[2^i - 1] > key$
- Далее выполняется бинарный поиск в интервале

  $2^{i-1} - 1, \ldots, 2^i - 1$

```c
// Вспомогательная функция проверки сортировки
bool is_sorted(int A[], int n) {
    for (int i = 1; i < n; i++) {
        if (A[i-1] > A[i]) {
            return false;
        }
    }
    return true;
}

// Бинарный поиск (вспомогательная функция для галоп-поиска)
int binary_search(int A[], int left, int right, int key) {
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (A[mid] == key) return mid;
        if (A[mid] < key) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}

// Галоп-поиск (экспоненциальный + бинарный)
int gallop_search(int A[], int n, int key) {
    if (A[0] == key) return 0;

    int i = 1;
    while (i < n && A[i] <= key) {
        i *= 2;
    }
```

```
    return binary_search(A, i/2, (i < n) ? i : n-1, key);
}

// Основная функция поиска с проверкой сортировки
static bool issorted = false; // Статическая переменная для хранения состояния

int search(int A[], int n, int key) {
    if (!issorted) {
        if (!is_sorted(A, n)) {
            // Используем qsort из стандартной библиотеки:
            qsort(A, n, sizeof(int),
                [](const void* a, const void* b) {
                    return (*(int*)a - *(int*)b);
                });
        }
        issorted = true;
    }
    return gallop_search(A, n, key);
}
```

Поиск в массиве

- Задан неупорядоченный массив ключей, новые элементы добавляются крайне редко
- Требуется периодически осуществлять поиск в массиве
- Решение 1, «в лоб»– Каждый раз при поиске использовать линейный поиск за O(n)
- Решение 2, в среднем за O(logn)– Один раз отсортировать массив за O(nlogn) или за O(n + k)– Использовать экспоненциальный поиск (galloping search) за O(logn)
