

# Binary Search

**Binary search.** Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

**Invariant.** Algorithm maintains  $a[\text{lo}] \leq \text{value} \leq a[\text{hi}]$ .

Ex. Binary search for 33.

[illegible]

# Binary Search

**Binary search.** Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

**Invariant.** Algorithm maintains  $a[lo] \leq value \leq a[hi]$ .

**Ex.** Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑							↑							↑
lo							mid							hi

# Binary Search

**Binary search.** Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

**Invariant.** Algorithm maintains  $a[lo] \leq value \leq a[hi]$ .

**Ex.** Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑						↑								
lo						hi								

# Binary Search

**Binary search.** Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

**Invariant.** Algorithm maintains  $a[\text{lo}] \leq \text{value} \leq a[\text{hi}]$ .

Ex. Binary search for 33.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑			↑			↑								
lo			mid			hi								

# Binary Search

**Binary search.** Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

**Invariant.** Algorithm maintains  $a[\text{lo}] \leq \text{value} \leq a[\text{hi}]$ .

Ex. Binary search for 33.

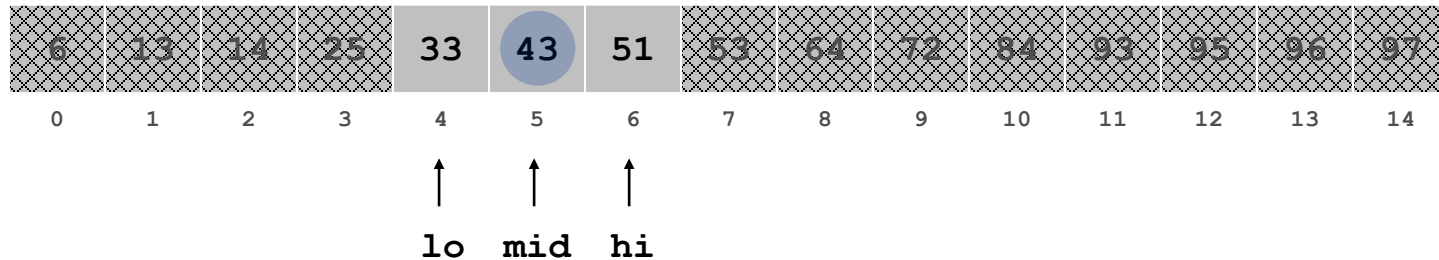
Diagram illustrating the initial state of the array for the binary search algorithm. The array contains 15 elements, indexed from 0 to 14. The elements are: 6, 13, 14, 25, 33, 43, 51, 53, 64, 72, 84, 93, 95, 96, 97. The elements 33, 43, and 51 are highlighted in a darker gray. The labels **lo** and **hi** are positioned below the array, with arrows pointing to the elements 33 and 51 respectively, indicating the current search range.

# Binary Search

**Binary search.** Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

**Invariant.** Algorithm maintains  $a[\text{lo}] \leq \text{value} \leq a[\text{hi}]$ .

**Ex.** Binary search for 33.

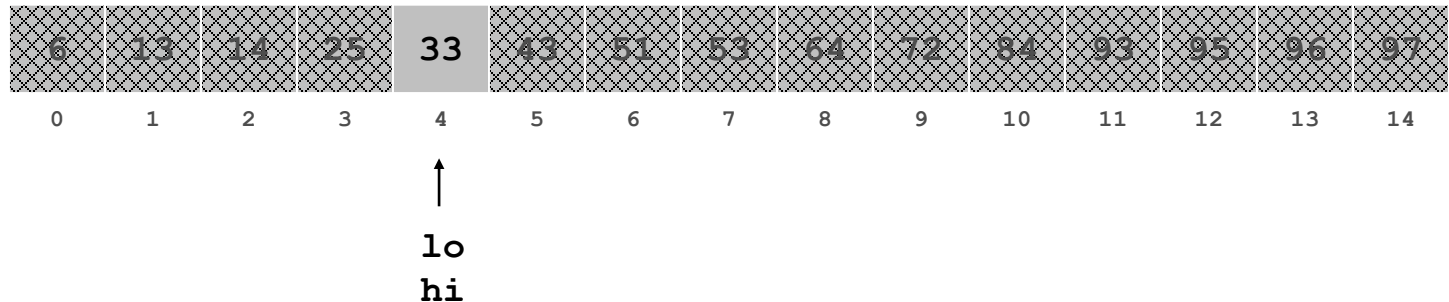


# Binary Search

**Binary search.** Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

**Invariant.** Algorithm maintains  $a[\text{lo}] \leq \text{value} \leq a[\text{hi}]$ .

**Ex.** Binary search for 33.

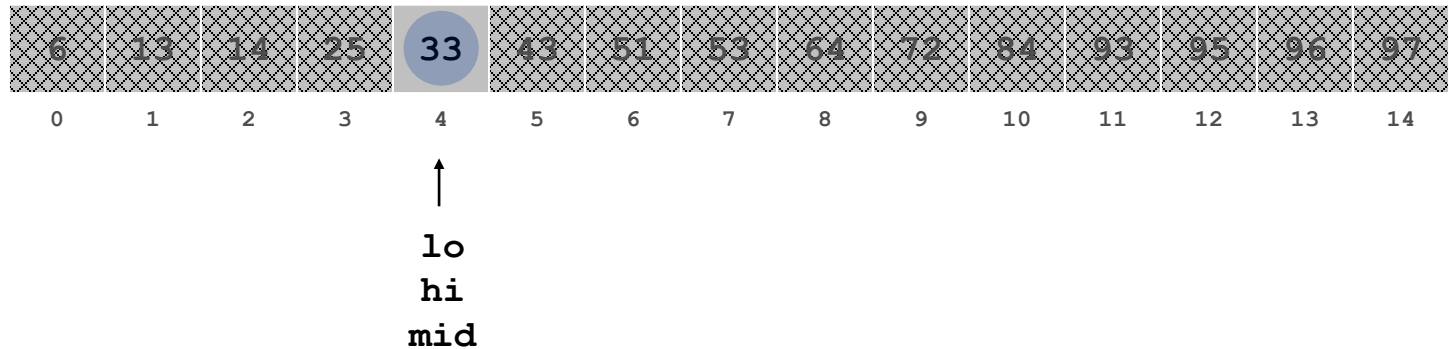


# Binary Search

**Binary search.** Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

**Invariant.** Algorithm maintains  $a[lo] \leq value \leq a[hi]$ .

**Ex.** Binary search for 33.



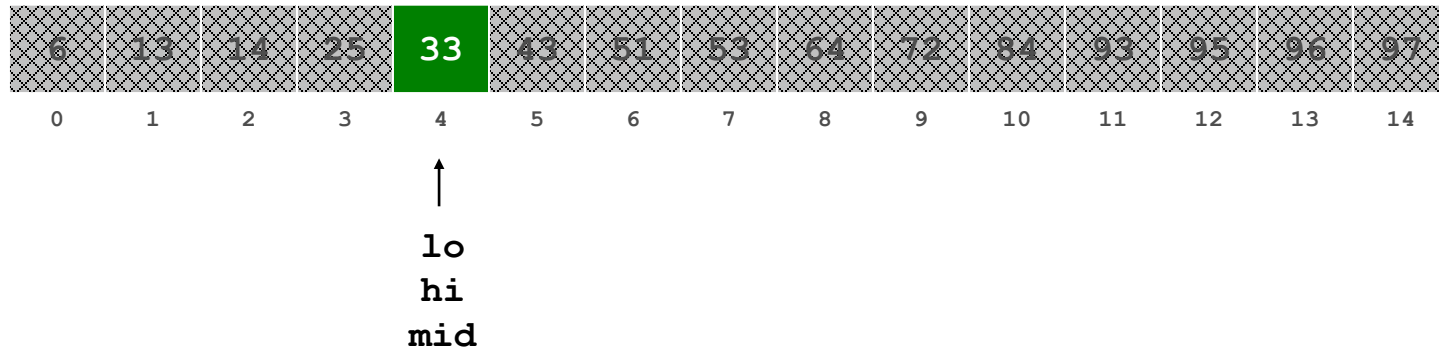


# Binary Search

**Binary search.** Given `value` and sorted array `a[]`, find index `i` such that `a[i] = value`, or report that no such index exists.

**Invariant.** Algorithm maintains  $a[lo] \leq value \leq a[hi]$ .

**Ex.** Binary search for 33.



# Binary Search Algorithm

```
int binarySearch(int[] a, int target) {  
    int min = 0;  
    int max = a.length - 1;  
  
    while (min <= max) {  
        int mid = (min + max) / 2;  
        if (a[mid] < target) {  
            min = mid + 1;  
        } else if (a[mid] > target) {  
            max = mid - 1;  
        } else {  
            return mid;    // target found  
        }  
    }  
  
    return -(min + 1);    // target not found  
}
```

## Binary Search with Recursion

```
int binarySearch(int[] a, int target, int min, int max) {  
    if (min > max) {  
        return -1;           // target not found  
    } else {  
        int mid = (min + max) / 2;  
        if (a[mid] < target) {           // go right  
            return binarySearch(a, target, mid + 1, max);  
        } else if (a[mid] > target) {    // go left  
            return binarySearch(a, target, min, mid - 1);  
        } else {  
            return mid;    // target found; a[mid] = target  
        }  
    }  
}
```