



Ask a Question



Introduction to Modified Binary Search

Let's go over the Modified Binary Search pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following... ▼

About the pattern


The **modified binary search** pattern is an extension of the traditional binary search algorithm and can be applied to a wide range of problems. Before we delve into the modified version, let's first recap the classic binary search algorithm.

Classic Binary Search

Binary search is an efficient search algorithm for searching a target value in sorted arrays or sorted lists that support direct addressing (also known as random access). It follows a divide-and-conquer approach, significantly reducing the search space with each iteration. The algorithm uses three indexes—start, end, and middle—and proceeds as follows:

1. Set the start and end indexes to the first and last elements of the array, respectively.



- 
2. Calculate the position of the middle index by taking the average of the start and end indexes. For example, if $start = 0$ and $end = 7$, then $middle = \lfloor (0 + 7) / 2 \rfloor = 3$.
 3. Compare the target value with the element at the middle index.
 4. If the target value is equal to the middle index element, we have found the target, and the search terminates.
 5. If the target value is less than the middle index element, update the end index to $middle - 1$ and repeat from step 2 onwards. Because the array is sorted, all the values between the middle and the end indexes will also be greater than the target value. Therefore, there's no reason to consider that half of the search space.
 6. If the target value is greater than the middle index element, update the start index to $middle + 1$ and repeat from step 2. Again, because the array is sorted, all the values between the start and the middle indexes will also be less than the target value. Therefore, there's no reason to consider that half of the search space.
 7. Continue the process until the target value is found or if the search space is exhausted, that is, if the start index has crossed the end index. This means that the algorithm has explored all possible values, which implies that the search space is now empty and the target value is not present.

Note: We're assuming the array is sorted in ascending order. If it's sorted in descending order, we'll do the opposite when changing the positions of the start and end pointers. Also, to avoid integer overflow when calculating the middle index, especially in languages with limited integer ranges, we can use $start + (end - start) / 2$ instead of $(start + end) / 2$.

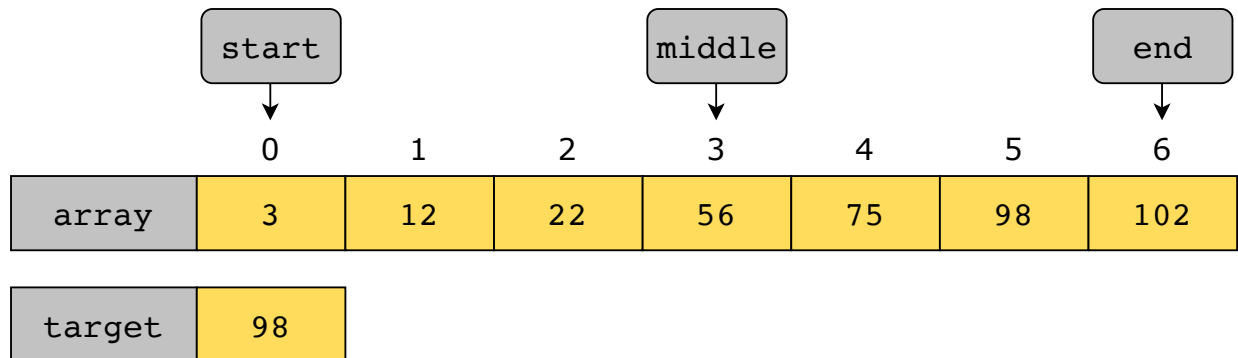
?

Tt



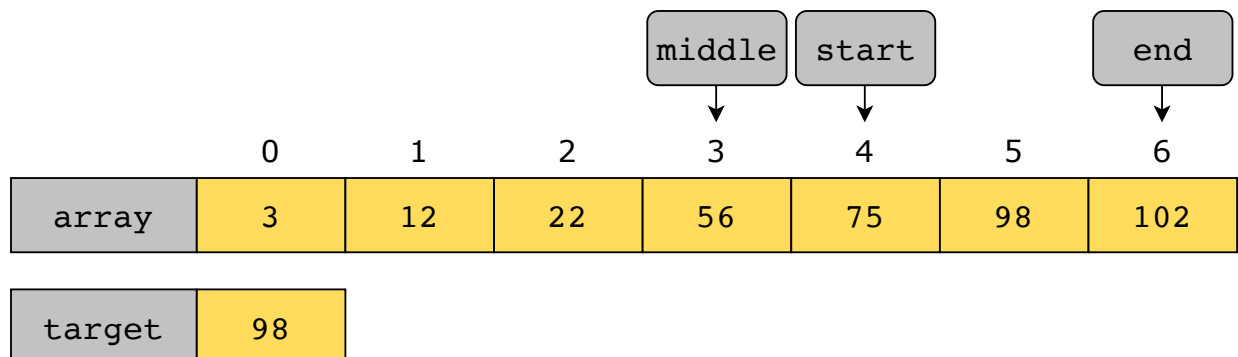
The following illustration shows how a binary search operation works using the techniques described above:

We calculate the position of **middle** by taking the floor of $(\text{start} + \text{end}) / 2$.



1 of 4

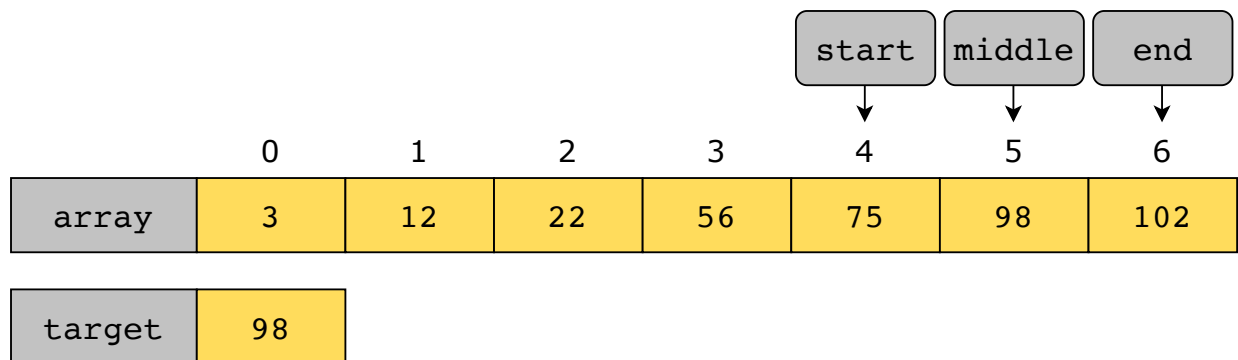
array[middle] < target, so the **start** now points to **middle + 1**.



2 of 4

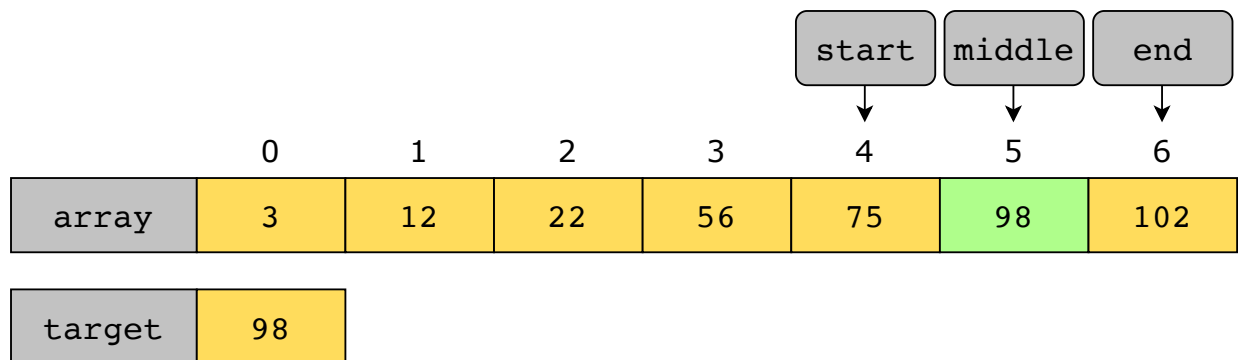


Update the position of **middle** by taking the floor of $(\text{start} + \text{end}) / 2$.



3 of 4

The element at **middle** is equal to the target value, so the algorithm ends here.



4 of 4

—

[]

Binary search reaches the target value in $O(\log(n))$ time because we divide the array into two halves at each step and then focus on only one of these halves. If we had opted for the brute-force approach, we would have had to traverse the entire array without any partitioning to search for the target value, which would take $O(n)$ in the worst case.

Modified Binary Search

The modified binary search pattern builds upon the basic binary search algorithm discussed above. It involves adapting the traditional binary search approach by applying certain conditions or transformations, allowing us to solve problems in which input data are modified in a certain way.

A few common variations of the modified binary search pattern are:

1. **Binary search on a modified array:** Sometimes, the array may be modified in a certain way, which affects the search process. For example, the array might be sorted and then rotated around some unknown pivot. Alternatively, some elements in a sorted array might be modified based on a specific condition. To handle such scenarios, we can modify the basic binary search technique to detect anomalies in the sorted order.
2. **Binary search with multiple requirements:** When searching for a target satisfying multiple requirements, a modified binary search can be used. It involves adapting the comparison logic within the binary search to accommodate multiple specifications. Examples include finding a target range rather than a single target or finding the leftmost or the rightmost occurrence of a target value.

Examples

The following examples illustrate some problems that can be solved with this approach:

1. **First and last position of an element in a sorted array:** For a given integer array sorted in increasing order, find the starting and ending position of a given target value.

We perform two binary searches—one to find the first occurrence and the other to find the last occurrence.

	0	1	2	3	4	5
array	3	9	9	9	11	12

target = 9

1 of 3



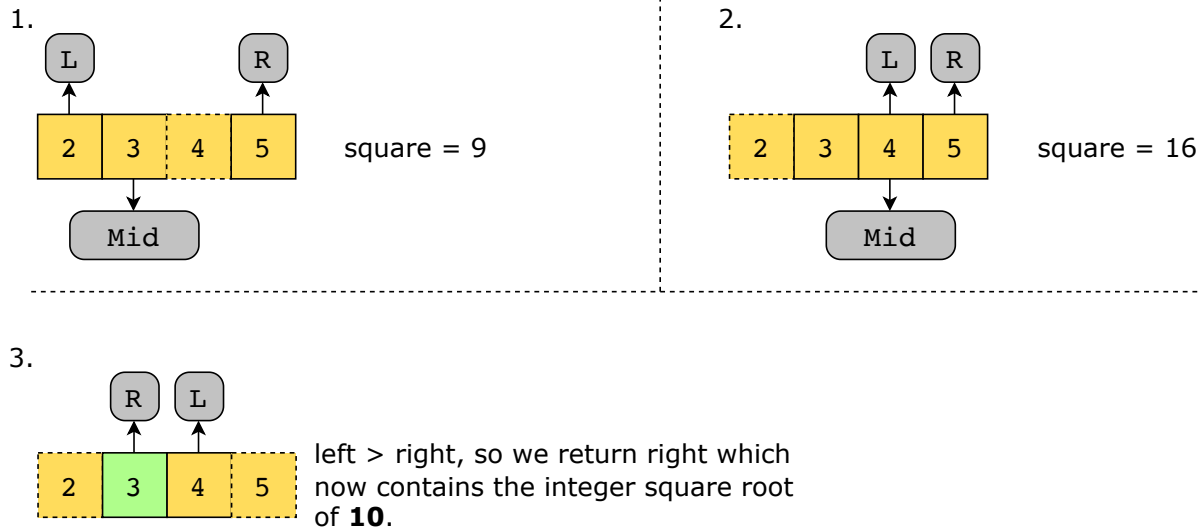
2. **Sqrt(x):** Find the integer square root of a positive number, x .



If the number is 0 or 1, we return the number. Otherwise, we set **left=2** and **right=floor(x/2)**. While **left<=right**, we calculate **mid=floor((left+right)/2)**. Let's store the value of **mid^2** in a variable called **square**.

If **square==x**, we return **mid**. Else, if the **square<x**, we set **left=mid+1** (see panels 1 and 2). Otherwise, if the **square>x**, we set **right=mid-1** (see panels 2 and 3). Finally, if our loop ends, i.e., when **left>right**, we simply return **right** (see panel 3).

x = 10




Does your problem match this pattern?

- Yes, if either of these conditions is fulfilled:
 - **Target value in sorted data:** The problem involves locating a specific target value—or identifying its first or last occurrence—within a sorted array or list. This pattern applies to data structures that support direct addressing.
 - **Partially sorted segments:** We may use this pattern when segments of an input array are sorted.
- No, if either of these conditions is fulfilled:
 - **Lack of direct addressing:** The input data structure does not support direct addressing.

?

Tt



- 
- **Unsorted or inappropriately sorted data:** The data to search is not sorted according to criteria relevant to the search. For example, if we're looking for a particular date in a list of dates, but the list is sorted in alphabetical order (and not chronologically), we cannot use binary search.
 - **Non-value-based solutions:** The problem does not require identifying a specific value or range of values. For example, if we have a list of student names only and we want to search for a student, modified binary search won't be a correct choice.

Real-world problems

Many problems in the real world use the modified binary search pattern. Let's look at some examples.

- **Dictionary searches:** A dictionary contains words that are alphabetically sorted. Therefore, we can use a classic binary search to find the required word quickly. If we wanted to find all the words in the dictionary that share a common prefix, we could modify the comparison operation used in the classic binary search algorithm.
- **Range-based filtering:** We can find range-based information by applying a modified binary search algorithm. For example, if we want to filter out YouTube videos on the basis of a certain time range, we can modify our classic binary search algorithm. Another example can also be filtering bank transactions by dates or finding test scores of students within a certain range.
- **String searching algorithms:** Modified binary search can be adapted for string searching algorithms such as search on the suffix array or search on the longest common prefix (LCP) array. These techniques are used in bioinformatics for DNA sequence analysis, text processing, and search engines.

?

Tt



Strategy time!

Match the problems that can be solved using the modified binary search pattern.

Note: Select a problem in the left-hand column by clicking it, and click one of the two options in the right-hand column.

Match The Answer

① Select an option from the left-hand side

Detect a cycle in a graph.

Modified Binary Search

Check if a number is the majority element in a sorted array.

Some other pattern

