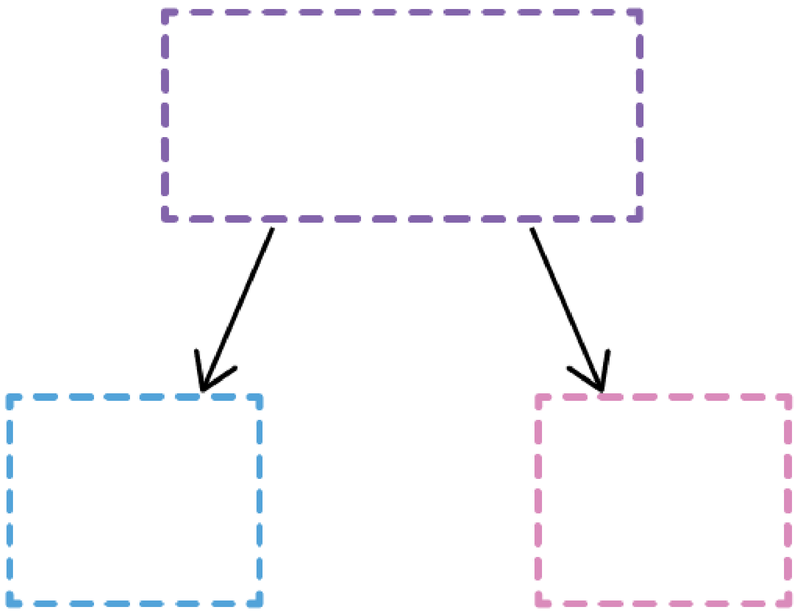
Binary Search

Binary Search is used with sorted array or list. In binary search, we follow the following steps:

1. We start by comparing the element to be searched with the element in the middle of the list/array.
2. If we get a match, we return the index of the middle element.
3. If we do not get a match, we check whether the element to be searched is less or greater than in value than the middle element.
4. If the element/number to be searched has a greater in value than the middle number, then we pick the elements on the right side of the middle element(as the list/array is sorted, hence on the right, we will have all the numbers greater than the middle number), and start again from the step 1.
5. If the element/number to be searched is lesser in value than the middle number, then we pick the elements on the left side of the middle element, and start again from the step 1.

   Binary Search is useful when there are large number of elements in an array and they are sorted. So a necessary condition for the Binary search to work is that the list/array should be sorted.



Binary Search

Binary search is In this example we are looking for the value 6 in a array of integer.

Features of Binary Search Algorithm

1. It is great to search through large sorted arrays.
2. It has a time complexity of O(log n) which is a very good time complexity. We will discuss this in detail in the Binary Search tutorial.
3. It has a simple implementation.

Binary search

This is the iterative version of the binary search:

```
FUNCTION binary_search(arr : ARRAY_OF INTEGER) : INTEGER
VAR
   left, right, mid : INTEGER;
BEGIN
```

```
    left := 0;
    right := arr.length-1;
    WHILE (left < right) DO
        mid := left + (right - left)/2;
        // check if x is present in the mid
        IF (arr[mid] = x) THEN
            RETURN mid;
        END_IF
        // if x grater, ignore the left half
        IF (arr[mid] < x) THEN
            left := mid+1;
        ELSE
        // if x is smaller, ignore the right half
        right := mid-1;
        END_IF
    END_WHILE
    // if we reached here then the element is not present
    RETURN -1   ;
END
```

‹ Previous   next ›