Binary Search

By:

Dustin Smith

Binary search is a useful algorithm for sorted arrays. Given some array and target value, we would like to find the index of the target, if it exists. Otherwise, we can return -1. We can approach the

1	3	4	7	8	10	13	14	18	19	21	24	37	40	45	71

Table 1: Sorted array for binary search.

binary search algorithm in two ways

1. Recursive
2. Iterative

First, we will consider the recursive case. Given the array in table 1 and a target value of 7, what would be our procedure?

1. Set start equal to 0 and end equal to the length of the array.
2. If start is greater than end, return -1.
3. Set the mid to the floor of start plus end divide by two.
4. If the array value at the mid point is our target value, return the mid point.
5. If the target is less than the value of the array at the mid point, we will call our function with parameters the array, target, start, and mid point minus 1 repeating the steps.
6. If not, we call our function with parameters the array, target, mid point plus 1, and end.

For the iterative procedure, we can just use a while the start index is less than or equal to the end incrementing start and decrementing end while we run the checks.

What is the space and time complexity of binary search? The space complexity is relatively easy to see. We are only keeping track of three pointers: start, end, and mid. That is, the space complexity of binary search is $\theta(1)$, constant space. For the time complexity, how do we step through process? First, we split the array so we have $n/2$ where n is the size of the array. The next step we have $n/4$ and so on. For any array, we have $n/2^k$ steps.

$$\frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \frac{n}{16}, \frac{n}{32}, \ldots \rightarrow \frac{n}{2^k}$$

We can bound $n/2^k$ by 1.

$$\begin{aligned} n/2^k &\leqslant 1 \\ n &\leqslant 2^k \\ \log_2 n &\leqslant k \end{aligned}$$

Therefore, our worst case time complexity is $\theta(\log_2 n)$ and best case would be the mid point is the value $\theta(1)$ constant time.

```python
# recursive
def binary_recrusive(
                array: List[int],
                target: int,
                start: int,
                end: int
                ) -> int:
  if start > end:
    return -1

  mid = (start + end) // 2

  if target == array[mid]:
        return mid

  if target < array[mid]:
        # search the left half from the next index
        # before the mid to the start
        return binary_recrusive(array, target, start, mid - 1)
  else:
        # search the right half from the next index
        # past mid to the end
        return binary_recrusive(array, target, mid + 1, end)


# iterative
def binary_iterative(array: List[int], target: int) -> int:
  mid, start = 0, 0
  end = len(array)

  while start <= end:
    mid = (start + end) // 2

    if target == array[mid]:
      return mid

    if target < array[mid]:
      end = mid - 1
    else:
      start = mid + 1
  return -1
```