Prepared for submission to JCAP

Binary Search

Mohammed Rizin [1]

[1]Unemployed, GitHub Accounts

E-mail: mrizin2013@gmail.com

Abstract. Binary Search is efficient Searching algorithm.

Keywords: binary, search, algorithm, data structures

ArXiv ePrint: 2434.3253

Dedicated to Just myself

[1]He's amazing

1 Binary Search Algorithm.

Earlier we learned about Divide and Conquer Method and discussed about calculating recurrence relation. In this we would be exploring Binary Search and we will code them in two methods : Iterative and Recursive Method.
Let us consider this array

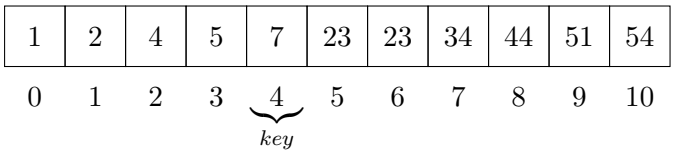

Figure 1: Searching for key = 7 after 4 steps

1.1 Algorithm : Interatively

Algorithm 1 Binary Search Algorithm : Iterative

```
procedure BinarySearch(A, n, key)
    l ← 0
    h ← n
    while l ≤ h do
        mid ← (l+h)/2
        if key = A[mid] then
            Return mid
        end if
        if key > A[mid] then
            l ← mid − 1
        else
            h ← mid + 1
        end if
    end while
    Return −1
end procedure
```

Let me explain this in detail. At first we set low and high pointer. We calculate the mid point of this and take floor value of it. If A[mid] is what we looking for then the procedure terminates and Returns the key. else. We will check if the key belong to right hand side of mid point value (as the array is already sorted) or left hand side and expand search only to that side. Doing this Interatively would give us a value of key if found else it will give not Found.
The Time Complexities :
If the searching value is the midpoint of the array $\boxed{O_{min}(n) = 1}$

If the searching value is at the leaf node tree of Binary Search $\boxed{O_{max}(n) = \log n}$
which is the height of the tree and that would be the number of comparisons

```python
def binarySearch(A: list, key: int) -> int:
    low = 0
    n = len(A)
    high = n - 1
    while (low <= high):
        mid = (low + high)//2
        if A[mid] == key:
            return mid
        elif key > A[mid]:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

1.2 Algorithm : Recursively

Algorithm 2 Binary Search Algorithm : Recursive

procedure BINARYSEARCH(l, h, key)
  if $l == h$ **then**
    if $key = A[l]$ **then**
      Return l
    else
      Return -1
    end if
  else
    $mid \leftarrow \left(\frac{l+h}{2}\right)$
    if $key = A[mid]$ **then**
      Return mid
    end if
    if $key > A[mid]$ **then**
      $BinarySearch(mid + 1, h, key)$
    else
      $BinarySearch(l, mid - 1, key)$
    end if
  end if
end procedure

To Build a recursive algorithm (aka. Divide and Conquer), we need to know what is a small problem. Of course, a small problem should of same task as the parent problem. But we need the smallest problem to which we know answer of. In this case, if $l == h$ is true, we know how to solve this case if $A[l] == key$ Then that's the element we looking for, else Not Found. If the problem still need to get break it down, then do the Binary search on the broken problem. If we calculate the Time Complexity:

if $key > A[mid]$ **then**
  $BinarySearch(mid + 1, h, key)$
else
  $BinarySearch(l, mid - 1, key)$

end if

¿What is the Time taken for small problem?

if $l == h$ **then**
  if $key = A[l]$ **then**
    Return l
  else
    Return -1
  end if
end if

Since all of them take constant amount of time, this is $O(1)$
Since this is a If statement, only one of recursive function work which is of Complexity : $T(\frac{n}{2})$. Along with some more constant statement which adds to $T(n) = T(\frac{n}{2}) + 1$ Then the function would be :

$$T(n) = \begin{cases} 1 & n \\ T\left(\frac{n}{2}\right) + 1 & n > 1 \end{cases} \tag{1.1}$$

According to Masters Theorem of Dividing Function:

$$a = 1 b = 2$$
$$\log_b a = 0 k = 0$$
$$\text{Since } \log_b a = k = 0 \text{ and } p = 0$$
$$\text{Then } T(n) \text{ is } O(\log n)$$

Here's the Python Code:

```python
def recursiveBinary(A: list, key: int, offset : int = 0) -> int:
    n = len(A)
    if n == 0:
        return -1
    mid = n//2
    if A[mid] == key:
        return offset + mid
    if key > A[mid]:
        return recursiveBinary(A[mid + 1:], key, offset + mid + 1)
    else:
        return recursiveBinary(A[:mid], key, offset)
```

Here's the C++ Code:

```cpp
int binarySearchRec(int l, int h, int key){
    if (l == h){
        if (key == A[l]) return l;
        else return -1;
    }
    int mid = (l+h)/2;
```

```
    if (A[mid] == key){
        return mid;
    } else if (key > A[mid]){
        return binarySearchRec(mid + 1, h, key);
    } else{
        return binarySearchRec(l, mid - 1, key);
    }
}
```