```
func(n){

if (base case)
    return result =>    O(1)
else if (….)
    func(n/2) …. //search key in half-sized of n => T(n/2) + c (time to compute half-
                                                                sized n, plus some
                                                                standard operations)
else
    func(n/2)…. //search key in another half-sized of n => T(n/2) + c, similar case
}
```

Let T(n) be the time complexity

T(n) = T(n/2) + c (We are only concerned 1/2 of original size, n.)

since T(n/2) = T(n/4) + c, therefore:

T(n) = (T(n/4) + c) + c

= T(n/4) + 2c

= T(n/8) + 3c

…

$T(n) = T(n/2^k) + kc$ (Generalized form)

We know T(1) = c, because only 1 possibility -> base case, let $k = \log_2 n$

$T(n) = T(n/2^{\log_2 n}) + c \log_2 n$

$= T(n/n) + c \log_2 n$

$= T(1) + c \log_2 n$

$= c + c \log_2 n$

Therefore: $T(n) = O(\lg n)$ or $O(\log_2 n)$