الگوریتم زیر را در نظر بگیرید:

find_k(A, k)

```
Require: a list A[0, ..., m − 1]
Ensure: returns index of k in A
  left = 1
  right = len(A) - 1
  while A[left] < k do
      left× = 2
  right = left
  left// = 2
  while left ≤ right do middle = (left + right) // 2
      if A[middle] == k then
          return middle
      else if A[middle] < k then
          left = middle + 1
      else
          right = middle − 1
  return -1
```

به طور خلاصه، آنقدر نشانه گر سمت چپ را دوبرابر می کنیم تا به مقداری برسیم که از کلید k بیشتر است (خواه این مقدار عدد باشد، خواه بینهایت). آنگاه، اشاره گر راست را برابر با اشاره گر چپ قرار داده و سپس اشاره گر چپ را نصف می کنیم (ما هربار اشاره گر چپ را دوبرابر کرده بودیم، پس بعد از یافتن اشاره گر راست، مطمئن هستیم که قبل از $left//2$ عنصری نبوده که از کلید k بیشتر باشد، چون در این صورت حلقه همان جا متوقف می شد.)

اگر دقت کنیم، تا اینجا ما به اندازه $\lceil \log k \rceil$ یا همان $\lfloor \log k \rfloor + 1$ عمل مقایسه انجام داده ایم، پس کارایی الگوریتم تا اینجا برابر $O(\log n)$ خواهد بود چون در بدترین حالت k همان n خواهد بود.

اکنون که چپ و راستِ بازه ای که k در آن قرار دارد را یافته ایم، با اعمال الگوریتم جستجوی دودویی به یافتن محل دقیق کلید k می پردازیم.

در بدترین حالت، $right = n$ و $left = n//2$ است که باعث می شود طول بازه ی جستجو $n - \frac{n}{2} = \frac{n}{2}$ باشد و کارایی زمانی الگوریتم جستجوی دودویی $O(\log \frac{n}{2})$ شود که همان $O(\log n)$ است.