با توجه به اینکه کارایی زمانی الگوریتم باید $O(\log n)$ باشد، به نظر می رسد که با نوعی الگوریتم جستجوی دودویی مواجه هستیم که در هر مرحله از اجرا، فضای جستجو رو نصف می کند.

find_biggest(A)

Require: a list $A[0, \ldots, n-1]$ containing n numbers
Ensure: returns maximum
  $left = 0$
  $right$ = len(A) - 1
  while $left \leq right$ **do** $middle = (left + right)//2$
    if $middle == 0$ **then**
      return max(A[0], A[1])
    else if $middle ==$ len(A) - 1 **then**
      return max($A[middle-1]$, $A[middle]$)
    else if $A[middle-1] < A[middle] > A[middle+1]$ **then**
      return $A[middle]$
    else if $A[middle-1] < A[middle] < A[middle+1]$ **then**
      $left = middle + 1$
    else if $A[middle-1] > A[middle] > A[middle+1]$ **then**
      $right = middle - 1$

الگوریتم فوق تا حدود زیادی مشابه الگوریتم جستجوی دودویی است. به طور کلی، در هر مرحله عضو مورد نظر را با عنصر قبلی و بعدی اش مقایسه کرده و در صورتی که از هر دو بیشتر باشد، آن را به عنوان عنصر ماکسیمم معرفی می کنیم. اما در صورتی که اشاره گر $middle$ روی عنصری قرار بگیرد که در شیب صعودی قرار دارد، اشاره گر سمت چپ را به عنصر بعدیِ $middle$ منتقل می کنیم. (چون اولا مطمئن هستیم خود $middle$ جواب مسئله نیست، و دوما می دانیم جواب قطعا بعد از $middle$ قرار دارد. چون روی شیب صعودی قرار گرفته ایم ولی هنوز به ماکسیمم نرسیده ایم) به همین شکل، در صورتی که اشاره گر $middle$ روی عناصر با شیب نزولی قرار بگیرد، اشاره گر $right$ روی عنصر قبلیِ $middle$ قرار می گیرد (چون اولا مطمئن هستیم خود $middle$ جواب مسئله نیست، و دوما می دانیم جواب قطعا قبل از $middle$ قرار دارد. چون روی شیب نزولی قرار گرفته ایم و ماکسیمم را رد کرده ایم) همچنین، در صورتی که اشاره گر $middle$ روی اولین را آخرین عنصر قرار بگیرد، می دانیم که فاصله $left$ و $right$ یک واحد است و صرفا دو عنصر برای مقایسه باقی مانده اند، پس ماکسیمم آنها را به عنوان جواب مسأله باز می گردانیم.

از طرفی، چون هر بار بازه، مسأله را به دو نیم تقسیم کرده و یک نیم را در نظر نمی گیریم، و عمل پایه را مقایسه در نظر می گیریم، چون تعداد ثابتی عمل پایه ای داریم، کارایی زمانی الگوریتم مضربی ثابت از $O(\log n)$ خواهد بود که با توجه به در نظر نگرفتن ثوابت، کارای زمانی الگوریتم همان $O(\log n)$ خواهد بود.