رویکرد تقسیم و غلبه یا تقسیم و حل

- نمونه ای از یک مساله را به صورت بازگشتی به تعدادی نمونه کوچکتر تقسیم کن تا زمانی که راه حل نمونه های کوچکتر به سادگی قابل تعیین باشند.
- رهیافت بالا به پایین که توسط روتین های بازگشتی به کار می رود.

جستجوی دودویی (Binary search)

این روش در آرایه های مرتب شده استفاده می شود. ابتدا کلید مورد نظر با عنصر وسط آرایه مقایسه می شود، اگر برابر بود جستجو تمام می شود. در غیر اینصورت اگر از آن کمتر بود جستجو در قسمت پایین آرایه وگرنه در قسمت بالای آن ادامه می یابد.

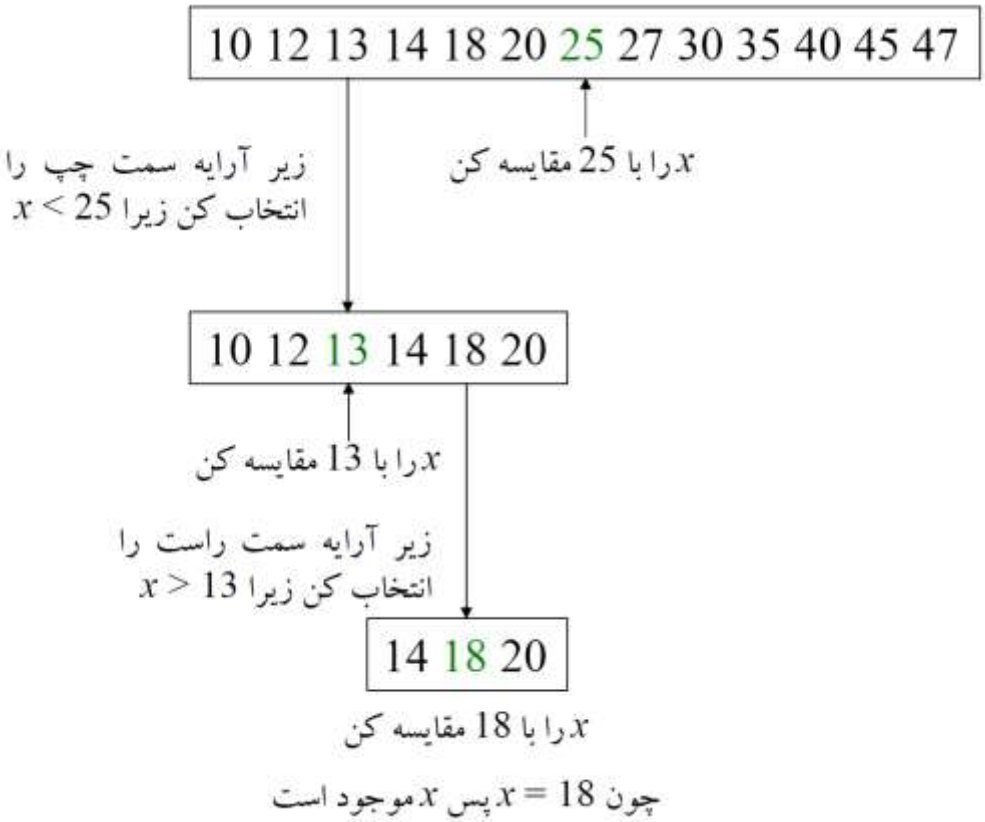
مثال 1:

- فرض کنید $x = 18$ و آرایه به صورت زیر باشد:

10 12 13 14 18 20 25 27 30 35 40 45 47

↑

عنصر وسط



نسخه غیر بازگشتی

```
int  bsearch (int  x [ ] , int  n , int m)
{
    int  low, high , mid;
    low = 0 ;    high  = n - 1 ;
   while  (low < = high ) {
          mid = (low + high) / 2 ;
          if  ( m < x [ mid ] )
                high  = mid - 1 ;
          else  if  (m > x [mid] )
                low = mid + 1 ;
          else   return  mid
    }
    return   - 1 ;
}
```

هدف شمارش تعداد مقایسه با عنصر میانی است. مرتبه اجرای الگوریتم فوق در حالات متوسط و بدترین $O(log\ n)$ است. در بهترین حالت نیز با یک مقایسه پایان می پذیرد.

- Case1: Element is present in the array
- Case2: Element is not present in the array.

There are **n** Case1 and **1** Case2. So total number of cases = **n+1**. Now notice the following:

- An element at index **n/2** can be found in **1** comparison
- Elements at index **n/4** and 3n/4 can be found in **2** comparisons.
- Elements at indices **n/8**, **3n/8**, **5n/8** and **7n/8** can be found in **3** comparisons and so on.

Based on this we can conclude that elements that require:

- 1 comparison = 1
- 2 comparisons = 2
- 3 comparisons = 4
- **i** comparisons = $\mathbf{2^{i-1}}$ where **x** belongs to the range **[1, logn]** because maximum comparisons = maximum time n can be halved = maximum comparisons to reach 1st element = logn.

So, total comparisons
= 1*(elements requiring 1 comparisons) + 2*(elements requiring 2 comparisons) + . . . + logn*(elements requiring logn comparisons)
= **1*1 + 2*2 + 3*4 + . . . + logn * ($\mathbf{2^{logn-1}}$)**

نسخه تقسیم و حل

```
int bsearch (int low, int high, int m, int x[ ])
{
 int  mid;
 if (low > high)     return  -1;
 else {
    mid = (low + high) / 2;
    if (m = = x[mid]    return   mid;
    else  if (m < x[mid])
              return bsearch (low, mid -1, m, x);
    else return bsearch (mid+1, high, m, x);
 }
}
```

پیچیدگی در بدترین حالت: این حالت زمانی اتفاق می افتد که عدد مورد جستجو آخرین عنصر آرایه باشد.

اگر n توانی از 2 باشد، داریم:

$$T(n) = T\left(\frac{n}{2}\right) + 1, T(1) = 1$$

جواب این رابطه بازگشتی

$$T(n) = \log n + 1 = \theta(\log n)$$

اگر n توانی از 2 نباشد، داریم:

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1, T(1) = 1$$

$\Theta(\log n)$

$$T(n) = \lfloor \log n \rfloor + 1$$

برای $n+1$ ؟

$T(n+1) = \lfloor \log(n+1) \rfloor + 1$?

$T(n+1) = T\left(\left\lfloor \frac{n+1}{2} \right\rfloor\right) + 1$ $\quad \left\lfloor \frac{n+1}{2} \right\rfloor = \frac{n}{2}$

$= T\left(\frac{n}{2}\right) + 1 = T(n) = \lfloor \log n \rfloor + 1$

$\lfloor \log(n+1) \rfloor = \lfloor \log n \rfloor$

$2^{k-1} < n < 2^k$

$2^{k-1} < n+1 < 2^k$

$(k-1)\log 2 < \log(n+1) < k \log 2$

$k-1 < \log(n+1) < k$

$\lfloor \log(n+1) \rfloor = k-1$

$2^{k-1} < n < 2^k$

$k-1 < \log n < k$

$\lfloor \log n \rfloor = k-1$