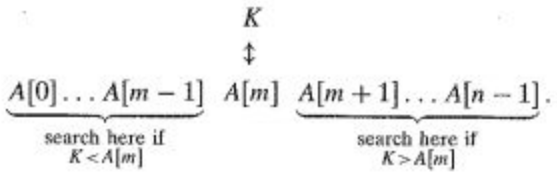
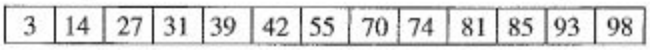
Binary Search:
Binary search is a 0 (log n) algorithm for searching in sorted arrays. It is an atypical example of an application of the divide-and-conquer technique because it needs to solve just one problem of half the size on each of its iterations.

It works by comparing a search key K with the array's middle element A[m].

- If they match, the algorithm stops;
- otherwise, the same operation is repeated recursively for the first half of the array if K < A[m], and for the second half if K > A[m]



As an example, let us apply binary search to searching for $K = 70$ in the array



The iterations of the algorithm are given in the following table:

index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	3	14	27	31	39	42	55	70	74	81	85	93	98
iteration 1	*l*						*m*						*r*
iteration 2								*l*		*m*			*r*
iteration 3								*l,m*	*r*				

ALGORITHM BinarySearch(A[O .. n -1], K)
//Implements non recursive binary search
//Input: An array A[O .. n - 1] sorted in ascending order and
II a search key K
//Output: An index of the array's element that is equal to K
II or -1 if there is no such element
l=0;r=n-1
while l <= r do
    m = (l + r)/2
    if K =A[m] return m
    elseif K < A[m] r=m-1

```
        else l = m+1
return -1
```

Analysis:

The standard way to analyze the efficiency of binary search is to count the number of times the search key is compared with an element of the array. Moreover, for the sake of simplicity, we will count the so-called three-way comparisons. This assumes that after one comparison of K with A[m], the algorithm can determine whether K is smaller, equal to, or larger than A[m].

comparisons depends not only on n but also on the specifics of a particular instance of the problem. Let us find the number of key comparisons in the worst case C(n). The worst-case inputs include all arrays that do not contain a given search key

Recurrence relation for worst case $C_{worst}(n)=C_{worst}(n)(n/2)+1$ for $n>1$, $C_{worst}(1)=1$

assume that $n = 2^K$ and solve the resulting recurrence by backward substitutions

Analysis:

Best-case: ~~Element found in the first attempt~~ Only one comparison.

$C_{best}(n) = 1$

Worst-case: key comparion is done with half of the size.

$C_w(1) = 1 \qquad n = 1$

$C_w(n) = C_w(n/2) + 1 \quad \text{for } n > 1$

$n = 2^k$

$C_w(2^k) = C_w(2^k/2) + 1$

$C_w(2^k) = C_w(2^{k-1}) + 1$

$C_w(2^k) = C_w(2^{k-2}) + 2$

$= C_w(2^{k-3}) + 3$

...

...

$= C_w(2^{k-i}) + i$

$k - i = 0 \qquad i = k$

$C_w(2^k) = C_w(1) + k$

$= 1 + k$

$= 1 + \log n$

$\Rightarrow C_w(2^{k-1}) = C_w(2^{k-2}) + 1$

$C_w(2^{k-2}) = C_w(2^{k-3}) + 2$

Successful search			Unsuccessful search
1	$\log n$	$\log n$	$\log n$
best	average	worst	best, average, worst