$RT = T(n)$

```
power ( x , n ): // compute x^n, x > 1, n ≥ 0
  if n = 0 then return 1                  — 1
  else
      p ← power ( x*x, n/2 )              — T(n/2)
      return odd( n ) ? p*x : p           — 1
```

$1 + \max(1, T(n/2) + 1)$

$$T(n) \le T(n/2) + 2, \quad n > 0$$

$$T(0) = 1$$

<u>Theorem:</u> $T(n) \le a \log n + b = O(\log n)$ for some constants a, b.

<u>Proof:</u> By induction on n.

<u>Base:</u> $n = 0$ $\quad T(0) = 1$

We need $1 \le a \cdot \log(0) + b$ — Not possible.

change theorem's base case to $n = 1$

$$T(1) \le T(0) + 2 = 1 + 2 = 3$$

<u>Base:</u> $n = 1$: $T(1) \le 3 \le a \log(1) + b$

$\Rightarrow \quad 3 \le a \cdot 0 + b = b$

<u>step:</u> Consider $n > 1$. Induction hypothesis (IH): Theorem is true for $1, 2, \dots, n-1$

$T(n) \le T(n/2) + 2$.

If $n > 1$, then $n/2 < n$

By IH, $T(n/2) \leq a\log(n/2) + b$

Combining the two inequalities,

$$T(n) \leq \left[a\log(n/2) + b\right] + 2$$
$$= a(\log n - 1) + b + 2$$
$$= a\log n + (-a + b + 2)$$

For proof to work, we need

$$a\log n + (-a+b+2) \leq a\log n + b$$
$$\Rightarrow \quad -a + 2 \leq 0$$
$$\Rightarrow \quad 2 \leq a$$

choose $a = 2$, $b = 3$

By induction,

$$T(n) \leq 2\log n + 3 = O(\log n)$$

Master method (Special case):

Theorem: Let $T(n) \leq aT(n/b) + cn^k$
for constants a, b, c, k. $a, b, c > 0$, $k \geq 0$

Then: Compare k and $\log_b a$

Case 1: If $k < \log_b a$ then $T(n) = O(n^{\log_b a})$

Case 2: If $k = \log_b a$ then $T(n) = O(n^{\log_b a} \cdot \log n) = O(n^k \log n)$

Case 3: If $k > \log_b a$ then $T(n) = O(n^k)$

```
boolean binarySearch ( A, x ): // Search for x in sorted array A, n = A.length
   return binarySearch ( A, 0, n − 1, x )

binarySearch ( A, p, r, x ): // Helper method to search for x in sorted array A[p..r]
if p > r then return false
else
    q ← (p + r) / 2
    if x < A[q] then
        return binarySearch ( A, p, q − 1, x )
    else if x = A[q] then
        return true
    else // x > A[q]
        return binarySearch ( A, q + 1, r, x )
```

RT recurrence:

$T(n) \le T(n/2) + c$

Case 2 of M.M.

$T(n) = O(\log n)$.

__Proof of correctness:__ binSrch(A, p, r, x) outputs true if there exists $q \in [p, r]$ such that $A[q] = x$, false otherwise.

__Proof:__ By induction on size of array, $s = r - p + 1$

__Base:__ if $p > r$ then $s = r - p + 1 \le 0$.
$s = 0$ then subarray $A[p..r]$ has no elements
So, $x \notin A[p..r]$ — algorithm returns false which is correct.

__step:__ Consider $s > 0$.
Program calculates $q = (p+r)/2$. $p \le q \le r$.
Algorithm compares x with $A[q]$.

__Case 1:__ $x < A[q]$: Since array is sorted,

$x < A[q] \le A[q+1] \le \dots \le A[r]$.

$\Rightarrow x \notin A[q..r]$.

If $x \in A[p..r]$, then x must be only in $A[p..q-1]$

Algorithm recursively calls

$BinSrch(A, p, q-1, x)$

size of subproblem $= q-1-p+1 = q-p$

Since $q \le r$, size of subproblem $\le r-p < s$

By IH: $BinSrch(A, p, q-1, x)$ correctly returns true if $x \in A[p..q-1]$, false otherwise.

Therefore, algorithm correctly returns true if $x \in A[p..r]$, false otherwise.

Case 2: $x = A[q]$

Since $p \le q \le r$, $x \in A[p..r]$

Algorithm returns true — which is correct.

Case 3: $x > A[q]$ — analogous to case 1. Switch roles of left / right subarrays.

By induction, $BSrch(A, p, r, x)$ is correct.

$\Rightarrow$ $BSrch(A, x)$ is correct.

```
binarySearch ( A, x ):
// Search for x in array A. Precondition: A[i−1] ≤ A[i] for i ∈ [ 1, A.length − 1]
left ← 0, right ← A.length − 1
while left ≤ right do
    mid ← ( left + right ) / 2
    if x < A [ mid ] then
        right ← mid − 1
    else if x = A [ mid ] then
        return mid
    else // x > A [ mid ]
        left ← mid + 1
return −1
```

$n = A.length$

Loop invariant (LI): $x \notin A[0..left-1]$,

$x \notin A[right+1..n-1]$.

Proof of correctness:

Initialization: when entering loop for the first time,

$left = 0$, $right = n-1$

$A[0..left-1]$, $A[right+1..n]$ are both empty arrays

x is not in them.

Maintenance: Case 1: $x < A[mid]$ - from prev $x \notin A[mid..right]$ (by induction)

Algorithm assigns $right \leftarrow mid-1$ - Verify LI for next iteration

Case 2: $x = A[mid]$ — loop is ended

Algorithm returns mid - correct.

Case 3: $x > A[mid]$ — analogous to case 1.

Termination: $left > right$ LI: $x \notin A[0..left-1]$, $x \notin A[right+1..n-1]$

$\Rightarrow x \notin A[0..n-1]$ - Alg returns -1 - correct.

Lists : Sequence, Ordered collection

{ 1, 7, 2, 18, 27, 7, - }

Duplicates, null values are allowed.

Linked List   Array list

List operations: 1 add (x) – add new element x at the end of the list

also known as insert

2 iterator : go through elements of list, one at a time

hasNext(), next(), remove( )

3 remove (x) : remove first occurrence of x from list

4 Contains (x) : does x appear in list ?

(find)

Common ops : size( ), isEmpty( ), clear( )

Additional ops : addFirst (x), addLast (x)

removeFirst( ), removeLast ( )

Indexing operations : (initial element is at index 0)

get (index), set (index, x), add (index, x),

remove (index)