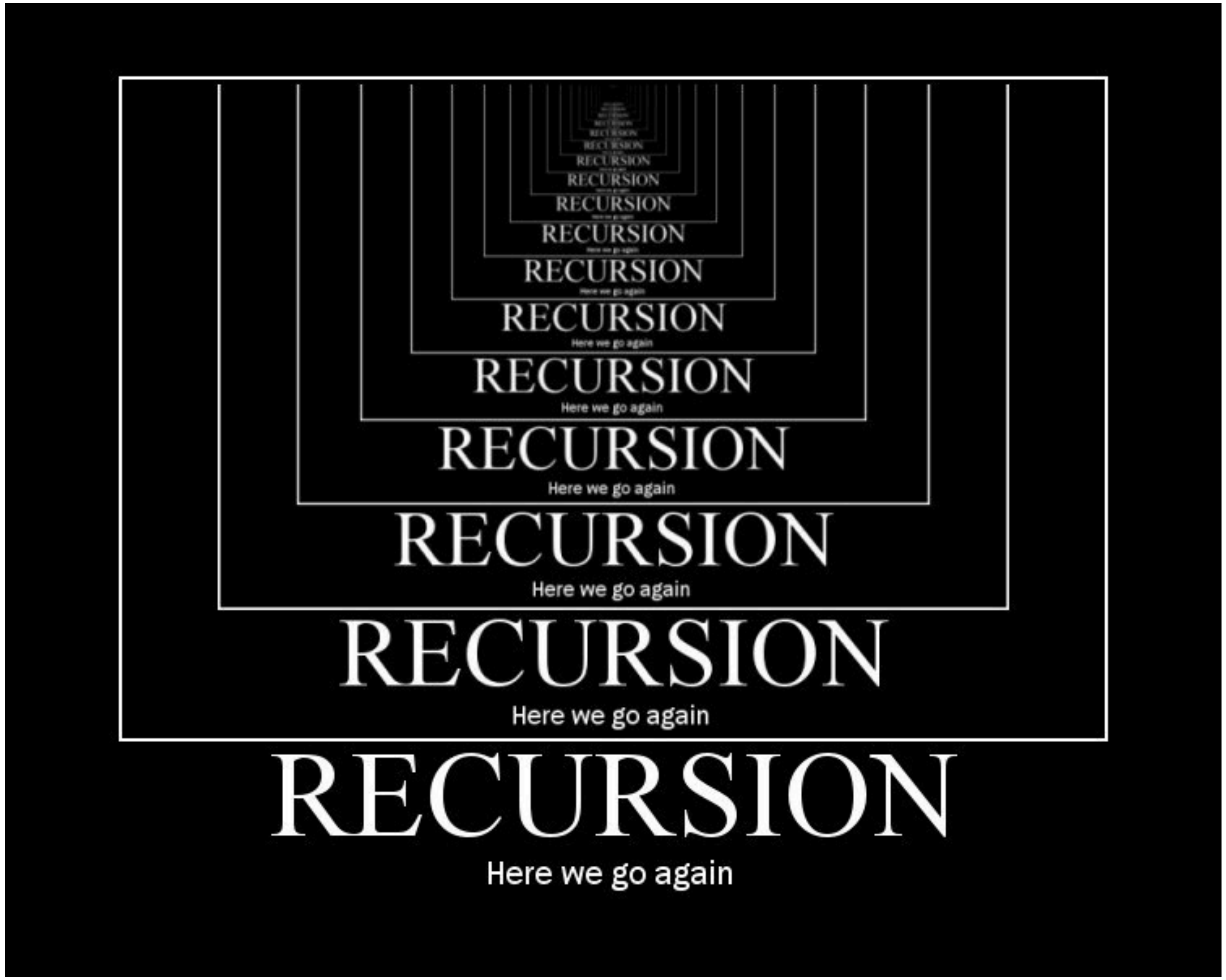



# Recursion

Jin Hyun Kim
Autumn 2019

# In this class

- How recursion works in program?

# Prob 1. Factorial Function

- Two statements for factorial function

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 & \text{if } n \geq 1. \end{cases}$$

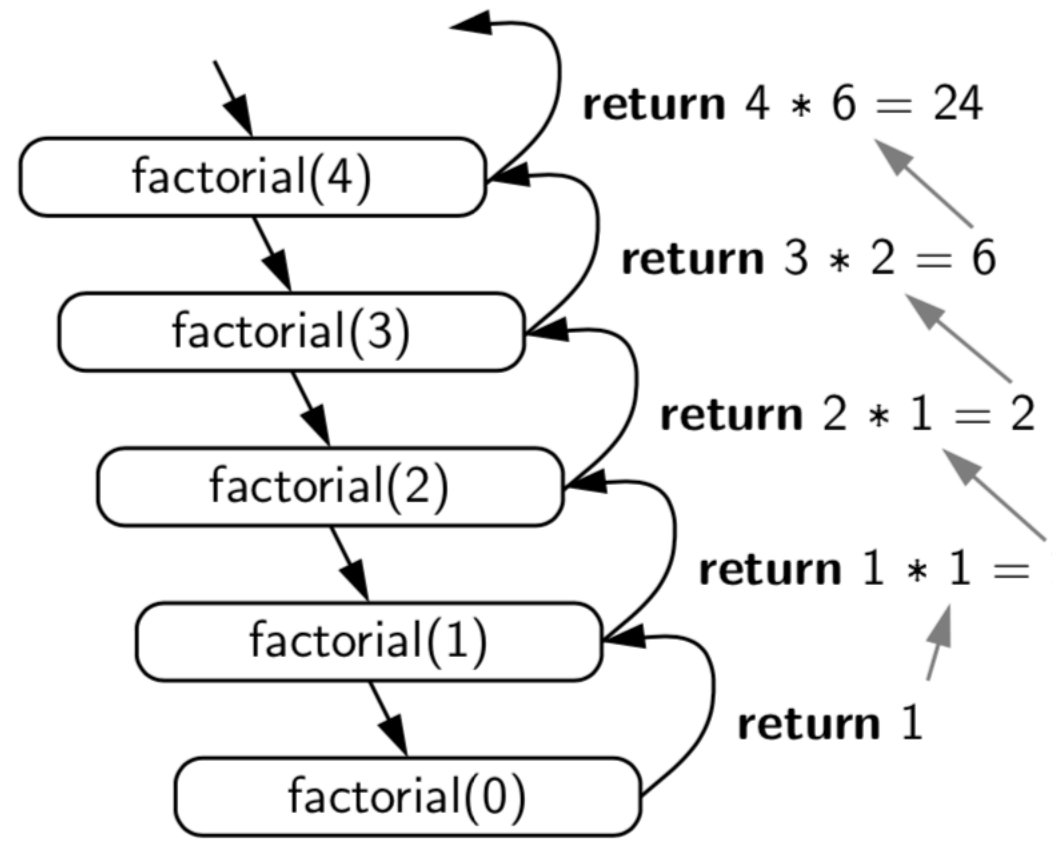
$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n \geq 1. \end{cases}$$

Non-Recursion Program

```
def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n−1)
```

# Recursion

```
def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n−1)
```

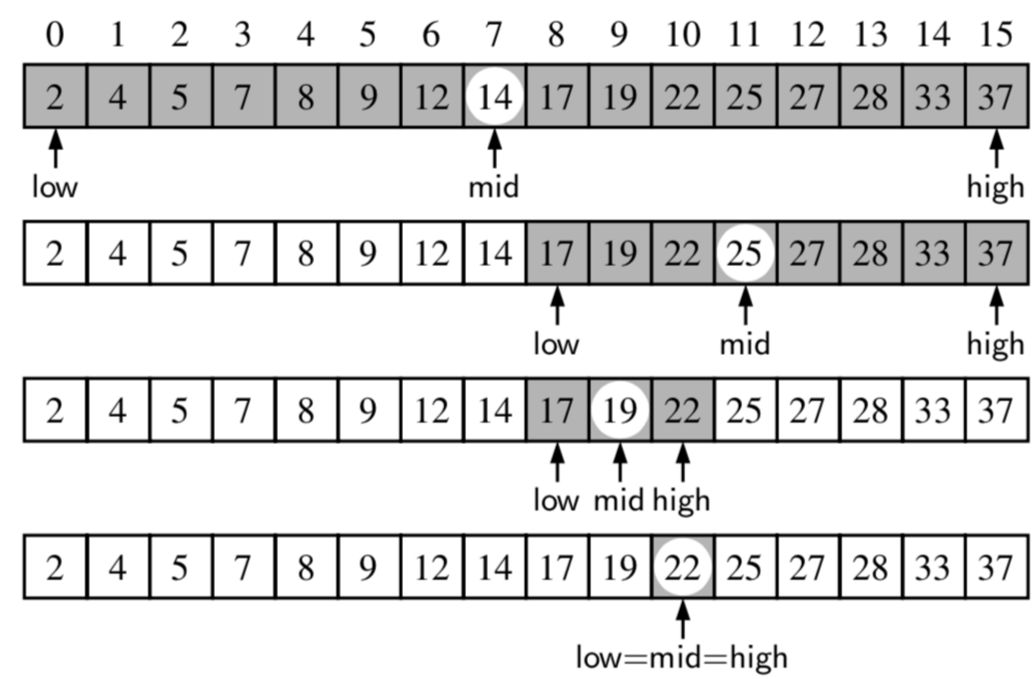


# Prob 2. Binary Search

- To locate a target value within a stored sequence
- Ex) Search 22

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 5 | 7 | 8 | 9 | 12 | 14 | 17 | 19 | 22 | 25 | 27 | 28 | 33 | 37 |

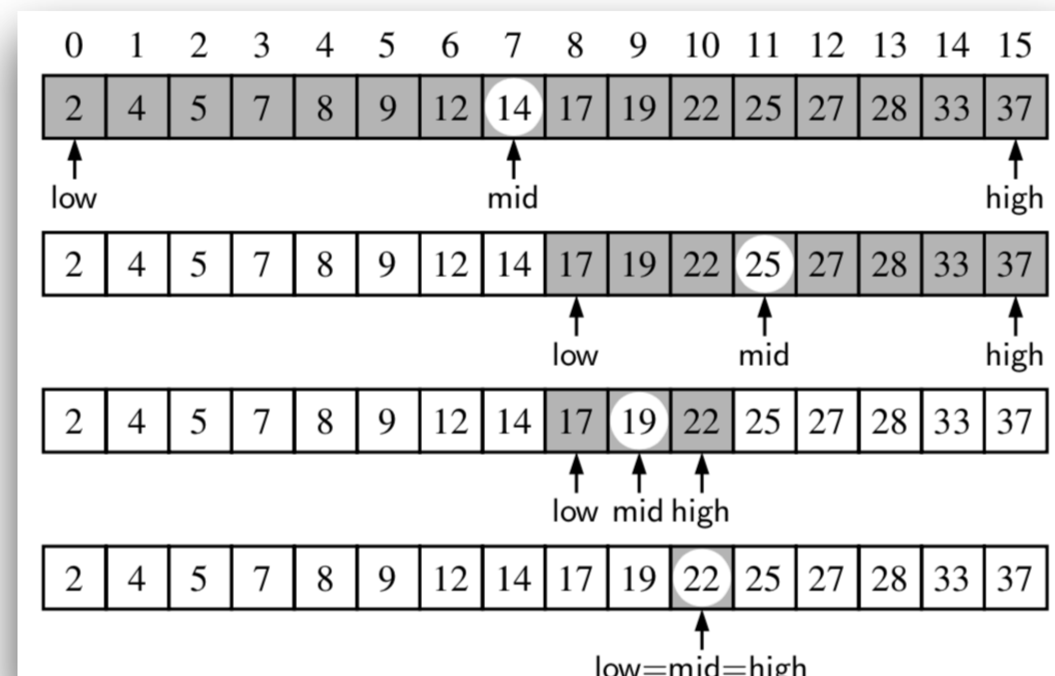
$$\mathrm{mid} = \lfloor (\mathrm{low} + \mathrm{high})/2 \rfloor .$$



# Binary Search

```
def binary_search(data, target, low, high):
  """Return True if target is found in indicated portion of a Python list.

  The search only considers the portion from data[low] to data[high] inclusive.
  """
  if low > high:
    return False                              # interval is empty; no match
  else:
    mid = (low + high) // 2
    if target == data[mid]:                   # found a match
      return True
    elif target < data[mid]:
      # recur on the portion left of the middle
      return binary_search(data, target, low, mid − 1)
    else:
      # recur on the portion right of the middle
      return binary_search(data, target, mid + 1, high)
```



# Complexity Analysis of B-Search in Recursion

**Proposition 4.2:** *The binary search algorithm runs in $O(\log n)$ time for a sorted sequence with $n$ elements.*

**Justification:** To prove this claim, a crucial fact is that with each recursive call the number of candidate entries still to be searched is given by the value

$$\mathsf{high} - \mathsf{low} + 1.$$

Moreover, the number of remaining candidates is reduced by at least one half with each recursive call. Specifically, from the definition of mid, the number of remaining candidates is either

$$(\mathsf{mid} - 1) - \mathsf{low} + 1 = \left\lfloor \frac{\mathsf{low} + \mathsf{high}}{2} \right\rfloor - \mathsf{low} \leq \frac{\mathsf{high} - \mathsf{low} + 1}{2}$$

or

$$\mathsf{high} - (\mathsf{mid} + 1) + 1 = \mathsf{high} - \left\lfloor \frac{\mathsf{low} + \mathsf{high}}{2} \right\rfloor \leq \frac{\mathsf{high} - \mathsf{low} + 1}{2}.$$

Initially, the number of candidates is $n$; after the first call in a binary search, it is at most $n/2$; after the second call, it is at most $n/4$; and so on. In general, after the $j^{th}$ call in a binary search, the number of candidate entries remaining is at most $n/2^j$. In the worst case (an unsuccessful search), the recursive calls stop when there are no more candidate entries. Hence, the maximum number of recursive calls performed, is the smallest integer $r$ such that

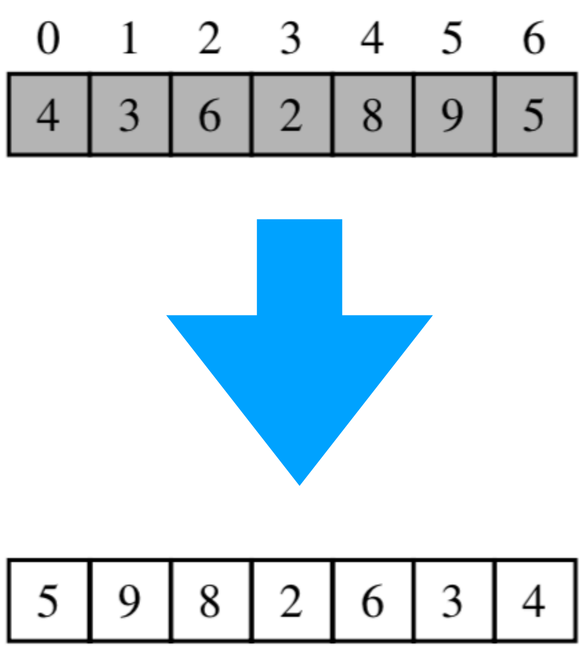
$$\frac{n}{2^r} < 1.$$

In other words (recalling that we omit a logarithm's base when it is 2), $r > \log n$. Thus, we have

$$r = \lfloor \log n \rfloor + 1,$$

which implies that binary search runs in $O(\log n)$ time. ■

# Prob 3. Reversing Sequence




```
def reverse(S, start, stop):
  """Reverse elements in implicit slice S[start:stop]."""
  if start < stop − 1:                          # if at least 2 elements:
    S[start], S[stop−1] = S[stop−1], S[start]   # swap first and last
    reverse(S, start+1, stop−1)                 # recur on rest
```

# Prob 5. Power Comp.

$$x^n = x \cdot x^{n-1} \text{ for } n > 0.$$

$$power(x,n) = \begin{cases} 1 & \text{if } n = 0 \\ x \cdot power(x, n-1) & \text{otherwise.} \end{cases}$$

```
def power(x, n):
  """Compute the value x**n for integer n."""
  if n == 0:
    return 1
  else:
    return x * power(x, n-1)
```

# Program Assignments

- Build a non-recursive program for factorial function

# In next class

- Array