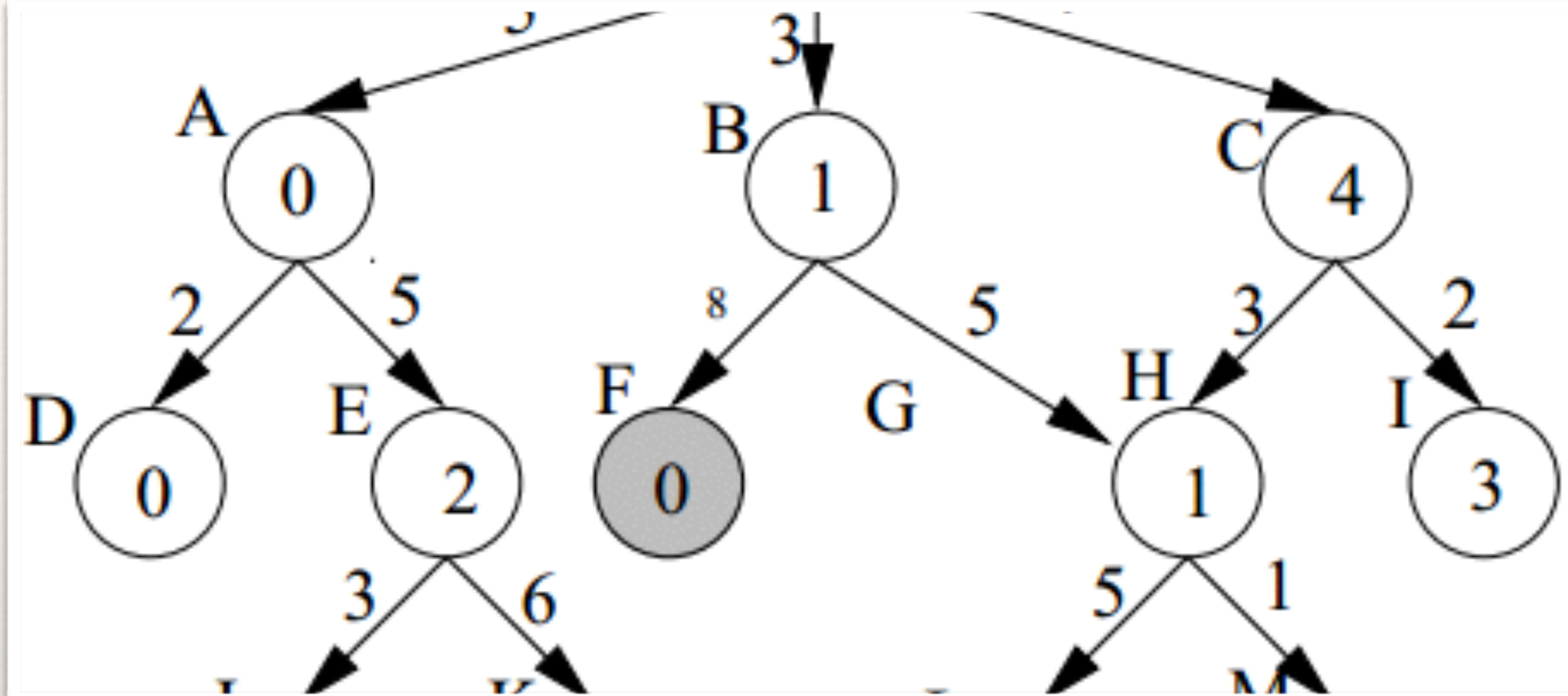
*Putting the Processing in Data Processing…*

# Algorithms

Sorting
Searching

# What are Algorithms?

- An algorithm is a sequence of actions (like a recipe).
- Algorithms can be analyzed as to their use of space and time.
- Algorithms move from an initial state and proceed through a number of actions until a terminating condition is reached.
- We can talk about the world **before** *(pre)*, **during**, and **after** *(post)* an algorithm has been executed.




*Lorem Ipsum Dolor*

# Searching

Linear Search
Binary Search

# Linear Search

- Start at the first element in an array and compare each element to the value you are looking for.
- Works with **unsorted** arrays.
- **On average,** you will find a solution in $n$ / 2 comparisons where $n$ is the total number of elements, *i.e.* you have to search half the array to find the solution on average.
- If the value is not in the array, then you must test all $n$ values. Very inefficient and very slow!

# Binary Search

- This search requires ordered data but is much faster than linear searching.
- Start with the middle value and keep dividing the array in half until the value is found.

# Binary Search

Find **7** in array **A**

| **Array A** | 1 | 4 | 7 | 8 | 9 | 11 | 17 | 21 | 22 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|
| | $L = 0$ | | | | $M = 4$ | | | | | $R = 9$ |

- M = ( L + R ) / 2 = (0 + 9) / 2 = 4 (integer truncation)
- A[M] = A[4] = 9 so the value we are looking for (7) must be in a location less than A[4] since the array is sorted.
- Set the value between L and M and move R
- Set R = M - 1 = 4 -1 = 3
- L remains the same, calculate new M = (0 + 3 ) /2 = 1

# Binary Search

Find **7** in array **A**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Array A | 1 | 4 | 7 | 8 | 9 | 11 | 17 | 21 | 22 | 30 |

$L = 0$ $M = 1$ $R = 3$

- A[M] = A[1] = 4 which is less than the value we are searching for
- Search value is in a location between M and R so move L to M + 1
- Set L = M + 1 = 1 + 1 = 2
- R stays the same (R = 3)
- M = (2+3)/2 = 2

# Binary Search

Find **7** in array **A**

| Array A | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 4 | 7 | 8 | 9 | 11 | 17 | 21 | 22 | 30 |

$L = 2$ $R = 3$

$M = 2$

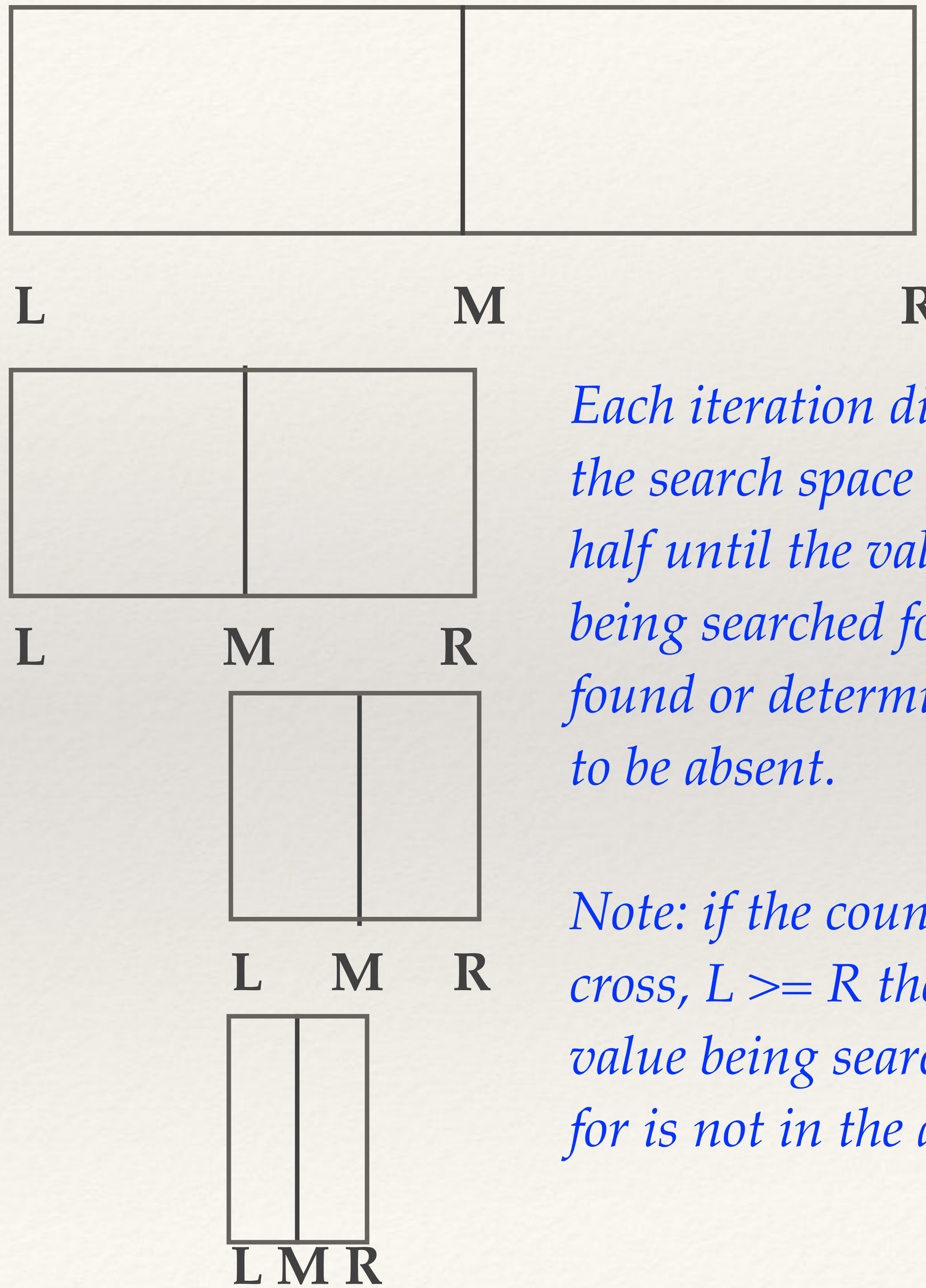
- A[M] = A[2] = 7 which is the value we are searching for!

# Binary Search Algorithm

- Search for a value $k$ in a given array $A$ with 0 to $n$-1 elements.
- Define 3 integer counters:
  - $L = 0$
  - $R = n - 1$
  - $M$

# Binary Search Algorithm

```
while ( L <= R ) {
   M = (L + R) / 2;
   if ( k == A[M] ) {
      return (M);
   } else if ( k > A[M] ) {
      L = M + 1;
   } else {
      R = M – 1;
   }
```




*Each iteration divides the search space in half until the value being searched for is found or determined to be absent.*

*Note: if the counters cross, L >= R then the value being searched for is not in the array.*

```
int binsearch ( int *array, int size, int findMe ) {

   int left  = 0;
   int right = size - 1;
   int middle;

   while ( left <= right ) {
      middle = ( left + right ) / 2;
      if ( findMe == *(array+middle) ) {
         return ( middle );
      } else if ( findMe > *(array+middle) ) {
         left = middle + 1;
      } else {
         right = middle - 1;
      }
   }

   return ( -1 );

}
```

# Some Interesting Facts…

- When Jon Bentley assigned binary search as a problem in a course for professional programmers, he found that ninety percent failed to provide a correct solution after several hours of working on it.

  *[Bentley, Jon (2000) [1986]. Programming Pearls (2nd ed.). Addison-Wesley.]*

- A study published in 1988 shows that accurate code for binary search is only found in five out of twenty textbooks.

  *[Pattis, Richard E. (1988). "Textbook errors in binary searching". SIGCSE Bulletin. 20: 190–194]*

# bsearch()

bsearch -- binary search of a sorted table

#include <stdlib.h>

The bsearch() function searches an array of nel objects, the initial member of which is pointed to by base, for a member that matches the object pointed to by key. The size (in bytes) of each member of the array is specified by width.

The contents of the array should be in ascending sorted order according to the comparison function referenced by compar.

void *bsearch ( const void *key, const void *base, size_t nel, size_t width, int (*compar) (const void *, const void *));

Using the binsearch function:

```
index = binsearch ( arr, size, i );
printf ( "binsearch index = %d\n", index );
```

Using the bsearch function in stdlib:

```
key = bsearch(&i, arr, size, sizeof(int),compare_ptr);
if ( key != NULL ) {
   printf ( "bsearch key = %d\n", *key );
} else {
   printf ( "bsearch key = NULL\n" );
}
```