# CSIS 3103

## Ch 5:

Recursion, Recursion, Recursion, Recursion, Recursion, Recursion, Recursion, Recursion, ...

## The Cost of Recursion

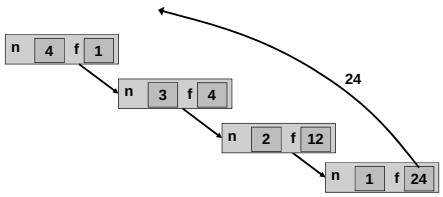
- Every method call involves allocating resources on the system stack for parameters, local variables, return address, etc.
  - Activation frame/call frame
- Exiting from the method requires removing the activation frame from the stack
- Recursive methods may have significant overhead due to allocating/deallocating activation frames

## Making Recursion More Efficient

- Recursive methods where the recursive call is the last statement executed can be as efficient as an equivalent loop
- This is called *tail recursion*
- Optimizing compilers actually replace recursive calls with a loop when tail recursion is detected

## Recursive factorial

```
factorial( 4, 1 )
```




## Tail Recursive factorial

```
private static int factorial(int n, int f) {
  if (n <= 1)
     return f;
  else
     return factorial(n – 1, n * f);
}

public static int factorial(int n) {
   return factorial(n, 1);
}
```

## Recursive Array Search

Base cases

**Empty array**: target not found, result is -1

**First element of the array = target**: result is the subscript of first element

Recursive step:

Search the rest of the array, excluding the first element

## Recursive Linear Search

```
/** Recursive linear search method (in RecursiveMethods.java).
    @param items The array being searched
    @param target The item being searched for
    @param posFirst The position of the current first element
    @return The subscript of target if found; otherwise -1
*/
private static int linearSearch(Object[] items,
                                Object target, int posFirst) {
    if (posFirst == items.length)
        return -1;
    else if (target.equals(items[posFirst]))
        return posFirst;
    else
        return linearSearch(items, target, posFirst + 1);
}
```

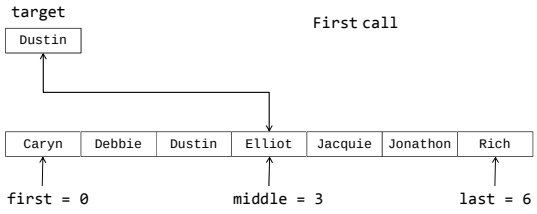
## Recursive Linear Search

```
/** Wrapper for recursive linear search method
    @param items The array being searched
    @param target The object being searched for
    @return  The subscript of target if found;
             otherwise -1
*/
public static int linearSearch(
                Object[] items, Object target) {
     return linearSearch(items, target, 0);
}
```

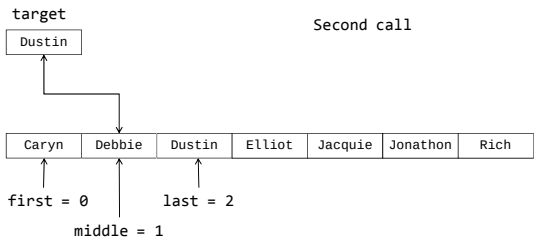
## Designing a Binary Search Algorithm

- Requires a sorted array
- Checks the middle element for a match with the target
- Base cases
  - The array is empty
  - Element being examined matches the target
- Recursive case
  - Throw away the half of the array that cannot contain the target and continue the search

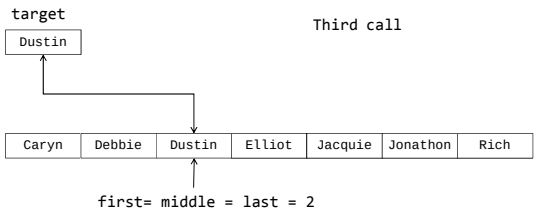
## Binary Search Algorithm




## Binary Search Algorithm




## Binary Search Algorithm



## Binary Search Algorithm

```
if the array is empty
      return –1
else if the middle element matches the target
      return the subscript of the middle
else if the target < the middle element
      recursively search array elements before the
      middle element and return the result
else
      recursively search array elements after the
      middle element and return the result
```

## Implementation of Binary Search

```
/** Recursive binary search method (in RecursiveMethods.java).
    @param items The array being searched
    @param target The object being searched for
    @param first The subscript of the first element
    @param last The subscript of the last element
    @return The subscript of target if found; otherwise –1.
*/
private static int binarySearch(Object[] items, Comparable target,
                                int first, int last) {
    if (first > last)
        return -1;     // Base case for unsuccessful search.
    else {
        int middle = (first + last) / 2;  // Next probe index.
        int compResult = target.compareTo(items[middle]);
        if (compResult == 0)
            return middle;   // Base case for successful search.
        else if (compResult < 0)
            return binarySearch(items, target, first, middle - 1);
        else
            return binarySearch(items, target, middle + 1, last);
    }
}
```

## Implementation of Binary Search

```
/** Wrapper for recursive binary search method (in RecursiveMethods.java).
    @param items The array being searched
    @param target The object being searched for
    @return The subscript of target if found; otherwise –1.
*/
public static int binarySearch(Object[] items, Comparable target) {
    return binarySearch(items, target, 0, items.length - 1);
}
```

```
binarySearch(items, target)
```

## Testing Binary Search

Use arrays with
- an even number of elements
- an odd number of elements
- duplicate elements

Test each array for the following cases:
- the target is the element at each position of the array, starting with the first position and ending with the last position
- the target is less than the smallest array element
- the target is greater than the largest array element
- the target is a value between each pair of items in the array

## Efficiency of Binary Search

At each recursive call half the array elements are eliminated

$O(\log_2 n)$

- An array of 16 needs 5 probes in the worst case

  $16 = 2^4$

  $5 = \log_2 16 + 1$
- An array with 32,768 elements requires only 16 probes! ($\log_2 32768 = 15$)

## `Arrays.binarySearch` Method

Java API class `Arrays` contains a `binarySearch` method
- Can be called with sorted arrays of primitive types or of objects
- If the array is not sorted, the results are undefined
- If there are multiple copies of the target value, there is no guarantee which one will be found
- Throws `ClassCastException` if the target is not comparable to the array elements

## Removing Recursion

- Tail recursive algorithms can easily be replaced by loops
- Other recursive algorithms may have to use stacks to replace recursive calls
  - Defeats the purpose, since recursion automatically incorporates use of the system stack

## Infinite Recursion

- Calling `factorial` with a negative argument will not terminate because `n` will never equal `0`
- Eventually a `StackOverflowError` exception occurs
- Make sure recursive methods always will reach a stopping case
- In the `factorial` method, throw an `IllegalArgumentException` if `n` is negative

## Recursive Definition of Linked List

A linked list is

- Empty, or
- It contains a node that has a reference to a linked list (the rest of the list)

Class `LinkedListRec<E>` implements several list operations using recursive methods