CSC180: Lecture 15

Wael Aboulsaadat

wael@cs.toronto.edu
http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

Recursion

how to approach recursion?

1. **Strategy:**
   - Rewrite the problem definition in a recursive way..
2. **Header:**
   - What info needed as input and output?
   - Write the function header.
   - Use a noun phrase for the function name
3. **Spec:**
   - Write a method specification in terms of the parameters and return value.
   - Include preconditions
4. **Base cases:**
   1. When is the answer so simple that we know it without recursing?
   2. What is the answer in these base cases(s)?
   3. Write code for the base case(s)
5. **Recursive Cases:**
   1. Describe the answer in the other case(s) in terms of the answer on smaller inputs
   2. Simplify if possible
   3. Write code for the recursive case(s)

Factorial using Recursion

$$N! = 1 * 2 * \ldots * N$$

```
int Factorial(int n) {
  int Product = 1,
      Scan    = 2;

  while ( Scan <=  n ) {
    Product  =  Product  *  Scan ;
    Scan = Scan + 1 ;
  }
  return (Product) ;
}
```

Factorial using Recursion

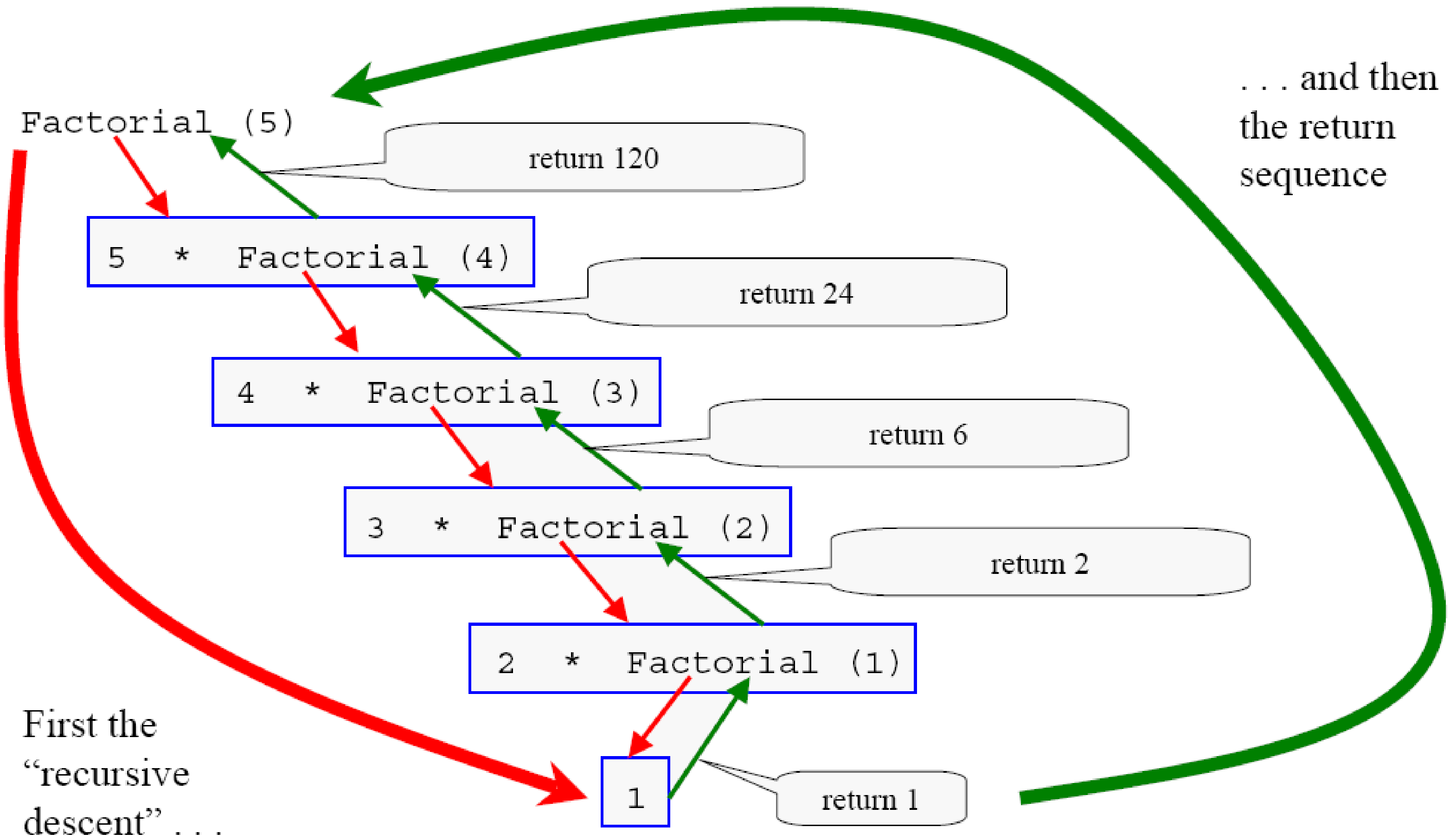
$$N! = 1 * 2 * \ldots * N$$

```
int Factorial(int n) {
  int Product = 1,
      Scan    = 2;

  while ( Scan <=  n ) {
    Product  =  Product  *  Scan ;
    Scan = Scan + 1 ;
  }
  return (Product) ;
}
```

```
int Factorial(int n ) {
  if ( n > 1 )
    return( n * Factorial (n-1) );
  else
    return(1);
}
```

Factorial using Recursion

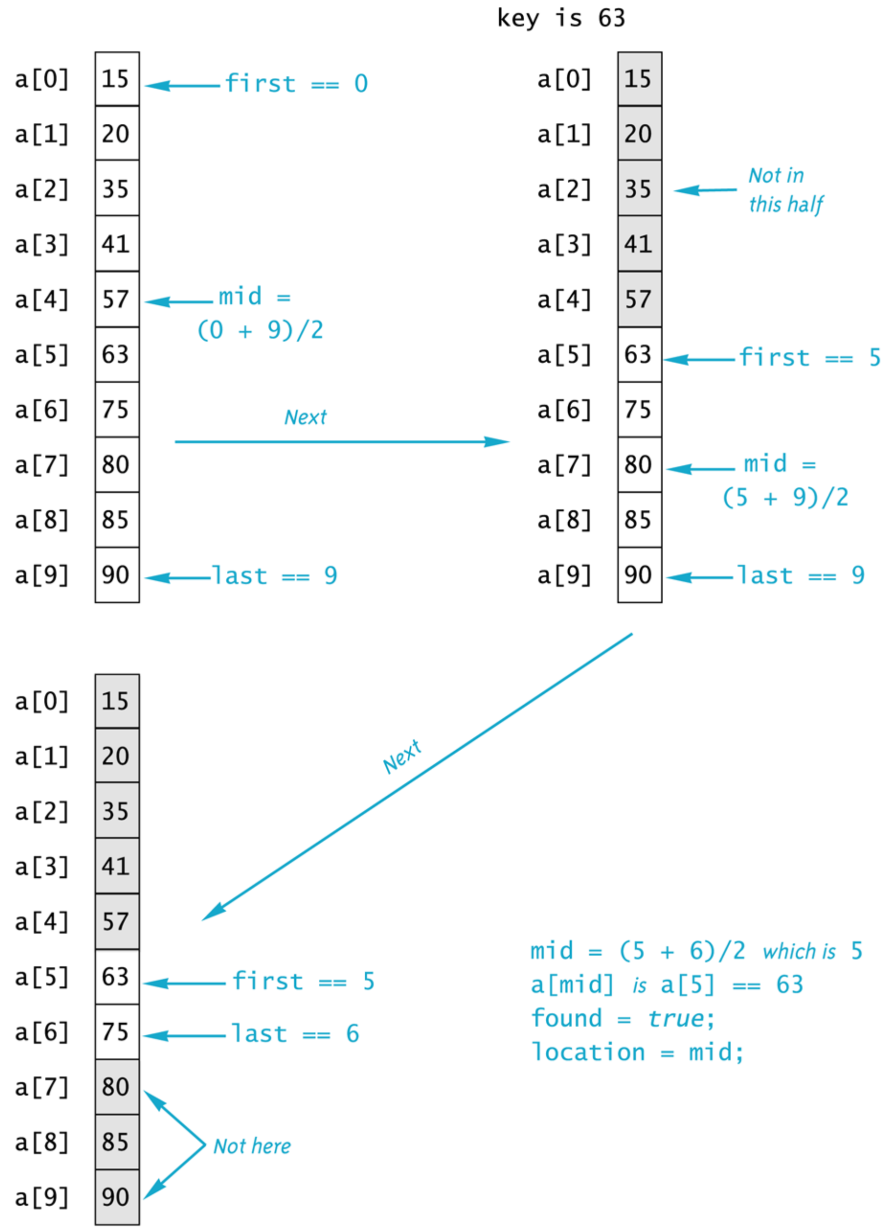
$$N! = 1 * 2 * \ldots * N$$



Binary Search

- Our algorithm is basically:
  - Look at the item in the middle
    - If it is the number we are looking for, we are done
    - If it is greater than the number we are looking for, look in the first half of the list
    - If it is less than the number we are looking for, look in the second half of the list

Execution of the Function search



Binary Search

An Iterative Version

Iterative Version of Binary Search

Function Declaration

```
void search(const int a[], int low_end, int high_end,
                           int key, bool& found, int& location);
//Precondition: a[low_end] through a[high_end] are sorted in increasing
//order.
//Postcondition: If key is not one of the values a[low_end] through
//a[high_end], then found == false; otherwise, a[location] == key and
//found == true.
```

Function Definition

```
void search(const int a[], int low_end, int high_end,
                           int key, bool& found, int& location)
{
    int first = low_end;
    int last = high_end;
    int mid;

    found = false;//so far
    while ( (first <= last) && !(found) )
    {
        mid = (first + last)/2;
        if (key == a[mid])
        {
            found = true;
            location = mid;
        }
        else if (key < a[mid])
        {
            last = mid - 1;
        }
        else if (key > a[mid])
        {
            first = mid + 1;
        }
    }
}
```

Binary Search
Recursive Version

- Since searching each of the shorter lists is a smaller version of the task we are working on, a recursive approach is natural

Binary Search
Recursive Version – pseudo code

- Here is our first refinement:

```
found = false;
mid = approx. midpoint between first and last;
if (key == a[mid])
{
    found = true;
    location = mid;
}
else if (key < a[mid])
   search a[first] through a[mid -1]
else if (key > a[mid])
   search a[mid +1] through a[last];
```

Binary Search
Recursive Version – pseudocode

- We must ensure that our algorithm ends
  - If key is found in the array, there is no recursive call and the process terminates
  - What if key is not found in the array?
    - At each recursive call, either the value of first is increased or the value of last is decreased
    - If first ever becomes larger than last, we know that there are no more indices to check and key is not in the array

Pseudocode for Binary Search

```
   int a[Some_Size_Value];
Algorithm to search a[first] through a[last]

    //Precondition:

    //a[first]<= a[first + 1] <= a[first + 2] <= ... <= a[last]
To locate the value key:
    if (first > last) //A stopping case
        found = false;
    else
    {
        mid = approximate midpoint between first and last;
        if (key == a[mid]) //A stopping case
        {
            found = true;
            location = mid;
        }
        else if key < a[mid] //A case with recursion
            search a[first] through a[mid - 1];
        else if key > a[mid] //A case with recursion
            search a[mid + 1] through a[last];
     }
```

Binary Search
Writing the Code

- Function search implements the algorithm:
  - Function search interface:

```
void search(const int a[ ], int first, int last,
                         int key, bool& found, int& location);
 //precondition:  a[0] through a[final_index] are
 //                          sorted in increasing order

 //postcondition: if key is not in a[0] - a[final_index]
 //                          found = = false; otherwise
 //                          found = = true
```

Binary Search
Checking the Recursion

1) There is no infinite recursion

- On each recursive call, the value of first is increased or the value of last is decreased. Eventually, if nothing else stops the recursion, the stopping case of first > last will be called

Binary Search
Checking the Recursion (cont.)

2) Each stopping case performs the correct action

- If first > last, there are no elements between a[first] and a[last] so key is not in this segment and it is correct to set found to false

- If k = = a[mid], the algorithm correctly sets found to true and location equal to mid

- Therefore both stopping cases are correct

Binary Search
Checking the Recursion (cont.)

- For each case that involves recursion, if all recursive calls perform their actions correctly, then the entire case performs correctly
  Since the array is sorted…
  - If key < a[mid], key is in one of elements a[first] through a[mid-1] if it is in the array. No other elements must be searched…the recursive call is correct
  - If key > a[mid], key is in one of elements a[mid+1] through a[last] if it is in the array. No other elements must be searched… the recursive call is correct

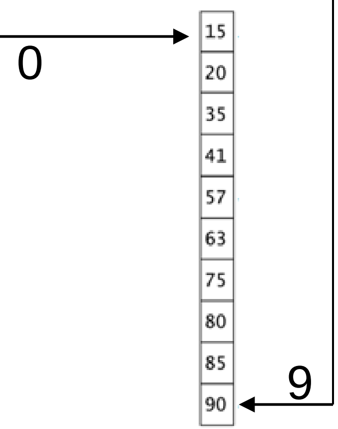
Recursive Function for Binary Search

```
void search(const int a[], int first, int last,
                            int key, bool& found, int& location)
{
    int mid;
    if (first > last)
    {
        found = false;
    }
    else
    {
        mid = (first + last)/2;

        if (key == a[mid])
        {
            found = true;
            location = mid;
        }
        else if (key < a[mid])
        {
            search(a, first, mid - 1, key, found, location);
        }
        else if (key > a[mid])
        {
            search(a, mid + 1, last, key, found, location);
        }
    }
}
```

Binary Search
Recursive Version

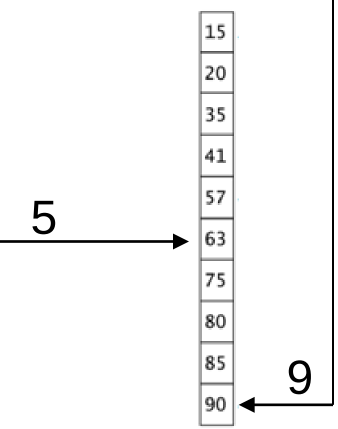
Key = 63

```
void search(const int a[], int first, int last,
                               int key, bool& found, int& location)
{
    int mid;
    if (first > last)
    {
        found = false;
    }
    else
    {
        mid = (first + last)/2;

        if (key == a[mid])
        {
            found = true;
            location = mid;
        }
        else if (key < a[mid])
        {
            search(a, first, mid - 1, key, found, location);
        }
        else if (key > a[mid])
        {
            search(a, mid + 1, last, key, found, location);
        }
    }
}
```



Key = 63

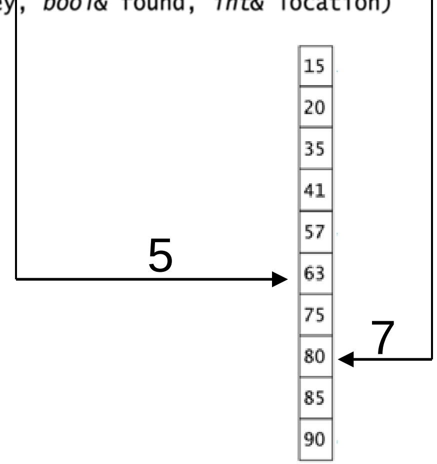
```
void search(const int a[], int first, int last,
                               int key, bool& found, int& location)
{
    int mid;
    if (first > last)
    {
        found = false;
    }
    else
    {
        mid = (first + last)/2;

        if (key == a[mid])
        {
            found = true;
            location = mid;
        }
        else if (key < a[mid])
        {
            search(a, first, mid - 1, key, found, location);
        }
        else if (key > a[mid])
        {
            search(a, mid + 1, last, key, found, location);
        }
    }
}
```



Binary Search
Recursive Version

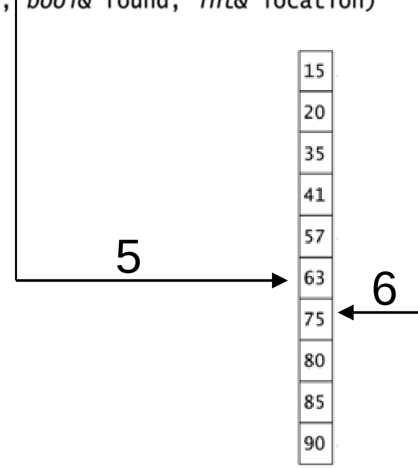
Key = 63

```
void search(const int a[], int first, int last,
                                  int key, bool& found, int& location)
{
    int mid;
    if (first > last)
    {
        found = false;
    }
    else
    {
        mid = (first + last)/2;

        if (key == a[mid])
        {
            found = true;
            location = mid;
        }
        else if (key < a[mid])
        {
            search(a, first, mid - 1, key, found, location);
        }
        else if (key > a[mid])
        {
            search(a, mid + 1, last, key, found, location);
        }
    }
}
```



Key = 63

```
void search(const int a[], int first, int last,
                                  int key, bool& found, int& location)
{
    int mid;
    if (first > last)
    {
        found = false;
    }
    else
    {
        mid = (first + last)/2;

        if (key == a[mid])
        {
            found = true;
            location = mid;
        }
        else if (key < a[mid])
        {
            search(a, first, mid - 1, key, found, location);
        }
        else if (key > a[mid])
        {
            search(a, mid + 1, last, key, found, location);
        }
    }
}
```



Pitfall: Stack Overflow

- Because each recursive call causes an activation frame to be placed on the stack
  - infinite recursion can force the stack to grow beyond its limits to accommodate all the activation frames required
  - The result is a stack overflow
  - A stack overflow causes abnormal termination of the program

Recursion Types

- Recursion for Tasks
  - E.g. binary search, sorting (later…)
- Recursion for Values
  - E.g. power, factorial, etc…

Recursion versus Iteration

- Any task that can be accomplished using recursion can also be done without recursion
  - A nonrecursive version of a function typically contains a loop or loops
  - A non-recursive version of a function is usually called an iterative-version
  - A recursive version of a function
    - Usually runs slower
    - Uses more storage
    - May use code that is easier to write and understand