CSC340: Recursion

Main Topics:

1. Recursive solutions: basics
2. Recursively solve: the n^{th} Fibonacci number, writing a string backward, the largest number, the k^{th} smallest number, binary search, and quickSort
3. Preliminary efficiency analysis: binary search, the largest number and the n^{th} fibonacci number

Readings:

1. Edition 5: Chapter 2
2. Edition 6: Chapter 5, Chapter 11
3. Class notes

Hui Yang
Computer Science Department
San Francisco State University
http://www.cs.sfsu.edu/~huiyang/

Outline

- **Recursive solutions: basics**
- **Recursively solve**
  - Binary search
  - The n^{th} Fibonacci number
  - Writing a string backward
  - The largest number
  - The k^{th} smallest (or largest) number
  - QuickSort
- **Preliminary efficiency analysis of**
  - binary search
  - the largest number
  - the n^{th} fibonacci number
  - The kth smallest (or largest) number
  - QuickSort

Recursively define and compute n!

- $n! = \begin{cases} 1 & \text{if } n=0 \\ n * (n-1)! & \text{If } n=1, 2, 3, \ldots \end{cases}$

- **The factorial function is recursively defined by itself with a smaller argument (i.e., a subproblem of the original problem)**
  - Base case(s)
  - Recursive cases

Multiplying Rabbits (The Fibonacci Sequence)

- **"Facts" about rabbits**
  - Rabbits never die
  - A rabbit reaches sexual maturity exactly two months after birth, that is, at the beginning of its third month of life
  - Rabbits are always born in male-female pairs. At the beginning of every month, each sexually mature male-female pair gives birth to exactly one male-female pair
- **Problem**
  - How many pairs of rabbits are alive in the n^{th} month?

Multiplying Rabbits (The Fibonacci Sequence)

- **Base cases**
  - *rabbit*(2), *rabbit*(1)
- **Recursive definition** (unit: pairs)

  rabbit(*n*) = 1 if *n* is 1 or 2

  rabbit(*n* – 1) + *rabbit*(*n* – 2) if *n* > 2
- **Fibonacci sequence**
  - The series of numbers *rabbit*(1), *rabbit*(2), *rabbit*(3), and so on; that is, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- **C++ implementation**?

Recursion basics

- **Recursion is a powerful problem-solving technique**
  - Breaks problem into smaller identical problems
  - An alternative to iteration, which involves loops
- **Facts about a recursive solution**
  - A recursive function calls itself
  - Each recursive call solves an identical, but smaller, problem
  - The solution to at least one smaller problem— the base case—is known
  - Eventually, one of the smaller problems must be the base case; reaching the base case enables the recursive calls to stop

Four key questions to recursion

1. How can you define the problem in terms of a smaller problem of the same type?
2. How does each recursive call reduce the size of the problem?
3. What instance(s) of the problem can serve as the base case?
4. As the problem size decreases, will you reach this base case?

Binary search: implementation

Recursive Function for Binary Search (*part 2 of 2*)

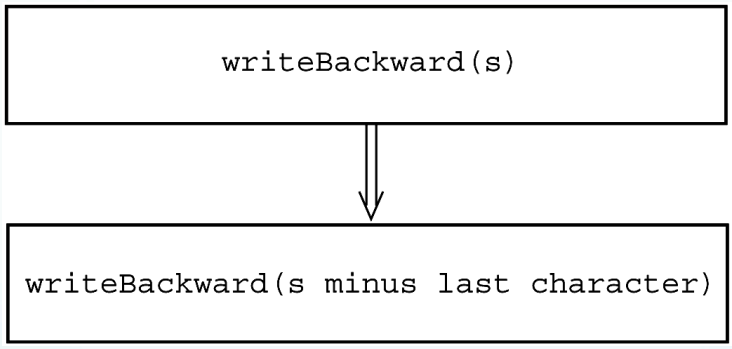
```
void search(const int a[], int first, int last,
                           int key, bool& found, int& location)
{
    int mid;
    if (first > last)
    {
        found = false;
    }
    else
    {
        mid = (first + last)/2;

        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);
        }
    }
}
```

A Recursive `void` Function: Writing a String Backward

- **Problem**
  - Given a string of characters, write it in reverse order
  - E.g.: "a cute cat" → "tac etuc a"
- **How would you solve it recursively?**
  - Key issues:
    - reduce a larger problem to smaller problem(s)
    - Base case(s)

A Recursive `void` Function: Writing a String Backward



- Each recursive step of the solution diminishes by 1 the length of the string to be written backward
- Base case: the empty string

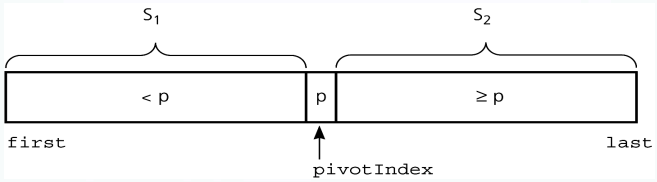
Finding the largest item in an array

- **Base case(s)?**
- **Recursive case: smaller problem(s)?**
- **A recursive solution**

```
if (anArray has only one item)
    maxArray(anArray) is the item in anArray
else if (anArray has more than one item)
        maxArray(anArray) is the maximum of
      maxArray(left half of anArray) and
      maxArray(right half of anArray)
```


Finding the k^{th} smallest item in an array

- **The recursive solution: the idea**



- Selecting a pivot item in the array
- Cleverly arranging, or partitioning, the items in the array about this pivot item
- Recursively applying the strategy to one of the partitions

Finding the k^{th} Smallest Item in an Array

- **Let**

  `kSmall(k, anArray, first, last) =`

  k^{th} smallest item in `anArray[first..last]`

- **Solution**

```
kSmall(k, anArray, first, last)
= kSmall(k,anArray,first,pivotIndex-1)
                          if k < pivotIndex - first + 1

= p                       if k = pivotIndex - first + 1

= kSmall(k-(pivotIndex-first+1), anArray,
         pivotIndex+1, last)
                          if k > pivotIndex - first + 1
```


The Partition Algorithm

```
PARTITION(A,p,r)
1  x = A[r]
2  i = p - 1
3  for j = p to r-1
4     if A[j] <= x
5        i = i + 1
6        exchange A[i] with A[j]
7  exchange A[i+1] with A[r]
8  return i+1
```

QuickSort: PseudoCode

```
QUICKSORT(A,p,r)
1  if p < r
2     q = PARTITION(A,p,r)
3     QUICKSORT(A,p,q-1)
4     QUICKSORT(A,q+1,r)
```


Preliminary efficiency analysis

- **Represent the running time as a function of the inputs**
  - T(n): the running time of an algorithm (e.g., binary search) over an input of size n
- **Binary search**
  - Complexity order: log(n)
  - Can you arrive at this solution quantitatively?
- **The largest number**
  - Complexity order?
- **The n^{th} Fibonacci number**
  - Complexity order?

Summary

- **Key issues towards constructing a recursive solution**
  - Base cases
  - Recursive cases
- **Recursively solve the following problems**
  - Binary search
  - Writing a string backward
  - The n^{th} Fibonacci number
  - The largest number
  - The k^{th} smallest number
- **Preliminary efficiency analysis**

Quiz

- **If a problem can be solved recursively, it can also be solved non-recursively.**
- **A recursive function or algorithm must have one or more base cases.**
- **Given a problem, its recursive solution is always as efficient as its non-recursive counterpart.**
- **What's the worst-case time complexity of the algorithm that selects the k^{th} smallest number for an array of size n using a recursive solution?**