CSE 230: Data Structures

Lecture 4-2: Application of Stacks

Dr. Vidhya Balasubramanian

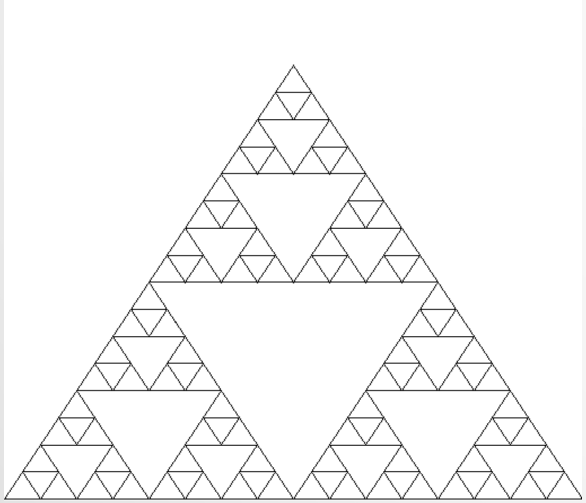
Amrita School of Engineering
Amrita Vishwa Vidyapeetham

Application of Stacks

- Recursion
- Tower of Hanoi
- Evaluation of Expressions

Recursion

- Concept of defining a function that calls itself as a sub-routine
  - Allows us to take advantage of the repeated structure in many problems
  - e.g finding the factorial of a number



http://introcs.cs.princeton.edu/java/23recursion/images/sierpinski5.png

http://www.spektyr.com/Gallery/Recursion.jpg

Linear Recursion

- Function is defined so that it makes atmost one recursive call at each time it is invoked
- Useful when an algorithmic problem
  - Can be viewed in terms of a first or last element, plus a remaining set with same structure

```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```

Src:mitpress.mit.edu

Algorithm LinearSum(A,n):
    Input: Integer array A and element n
    Output: Sum of first n elements of A
    if n=1 **then**
        return A[0]
    else
        return LinearSum(A,n-1)+A[n-1]

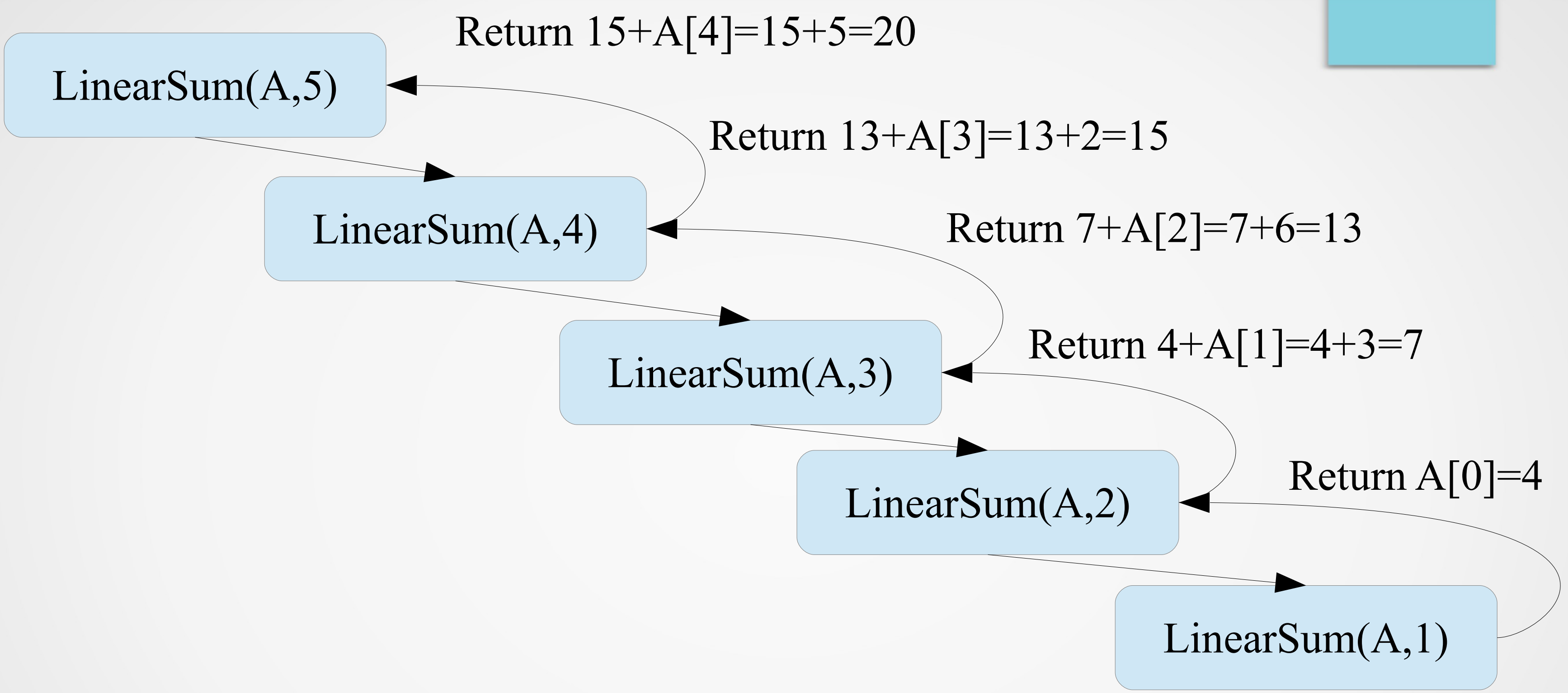
Linear Recursion

- Algorithm using linear recursion uses the following
  - Test for base cases
    - Base cases defined so that every possible chain of recursive call reaches base case
    - Helps in termination
  - Recurse
    - May decide on one of multiple recursive calls to make
    - Recursive calls must progress to base case

Analyzing Recursive Algorithms

- Use visual tool – recursion trace
  - Contains box for each recursive call
    - Contains parameters of the call
  - Links showing the return value
- Use recurrence relation
  - Mathematical formulation to capture the recursion process

Analyzing Recursive Algorithms



- Example LinearSum
  - Running time is O(n)
  - Space Complexity: O(n)

Stack Trace

- Example stack trace

LinearSum(A,1)	Return A[0]=4
LinearSum(A,1)+A[1]	Return 4+A[1]=4+3=7
LinearSum(A,2)+A[2]	Return 7+A[2]=7+6=13
LinearSum(A,3)+A[3]	
LinearSum(A,4)+A[4]	

Problem 1

- Reverse an array using linear recursion
- Solution
  - **Algorithm** ReverseArray(*A*,*i*,*n*):

    Input: Integer array *A* and integers *i*,*n*

    Output: Reversal of *n* integers in *A* starting from *i*

    if n<=1 **then**

        return

    else

        Swap *A*[i] and *A*[*i*+*n*-1]

        Call ReverseArray(*A*,*i*+1,*n*-2)

    return

Problem 2:

- Computing Powers via Linear Recursion

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

$$power(x,n)=\begin{cases} 1 \text{ if } n=0 \\ \text{x.power}(x,(n-1)/2)^2 \text{ if } n>0 \text{ is odd} \\ \text{power}(x,n/2)^2 \text{ if } n>0 \text{ is even} \end{cases}$$

Problem 3

- Describe a linear recursive algorithm for finding the minimum element in an n-element array
- Write a function using recursion to print numbers from n to 0.
- Given n write a recursive function to find the factorial of n
- Given a non-negative integer n, return the sum of its digits
- Given a string, compute recursively a new string where all the adjacent chars are now separated by a "*". e.g abc-→ a*b*c

Higher-Order Recursion

- Uses more than one recursion call
  - e.g 3-way merge sort
- Binary recursion
  - Two recursion calls
  - e.g BinarySum
    - **Algorithm** BinarySum(A,i,n):

      Input: Integer array A and integers i, n

      Output: Sum of first n elements of A starting at index i

      if n=1 then

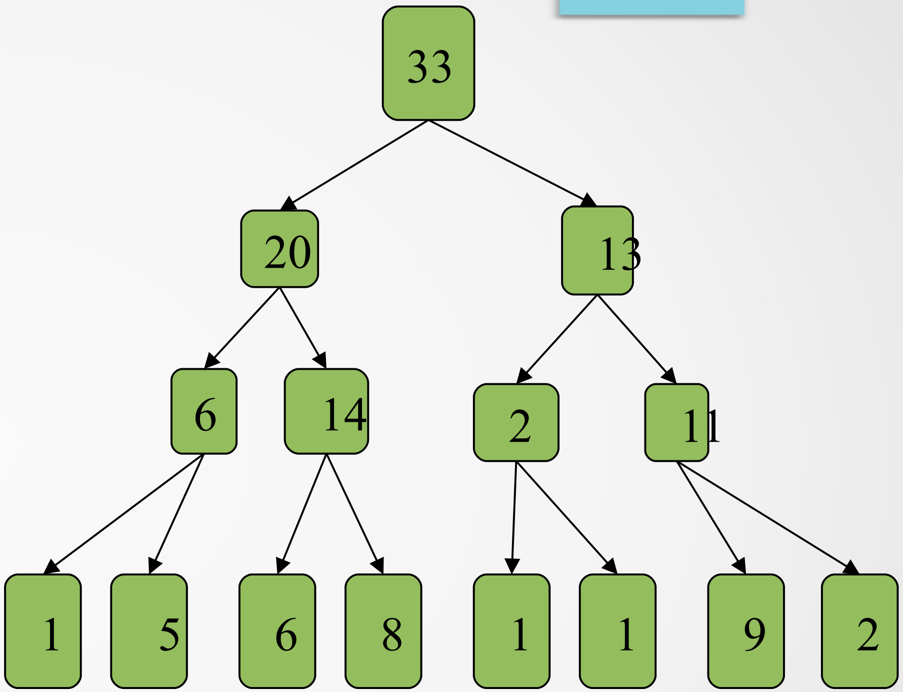
          return A[i]

      else

          return BinarySum(A,i,$\lceil n/2 \rceil$)+BinarySum(A,i+$\lceil n/2 \rceil$,$\lfloor n/2 \rfloor$)

Analysis

- Recursion trace is a tree
- Depth of recursion
  - O(logn)
  - Lesser additional space needed
- Running time
  - O(n)
  - Have to visit every element in the array



Problem

- Generate the kth Fibonacci number using Binary Recursion
- Solution (This method not recommended !! Exponential complexity)
  - **Algorithm** BinaryFib(*k*):

    Input: k

    Output: k^th Fibonacci Number

    if k<=1 **then**

        return k

    else

        return BinaryFib(k-1)+BinaryFib(k-2)

Problem

- Describe a binary recursive method for searching an element x in an n-element unsorted array A.
  - Compute the running time and space complexity of your algorithm

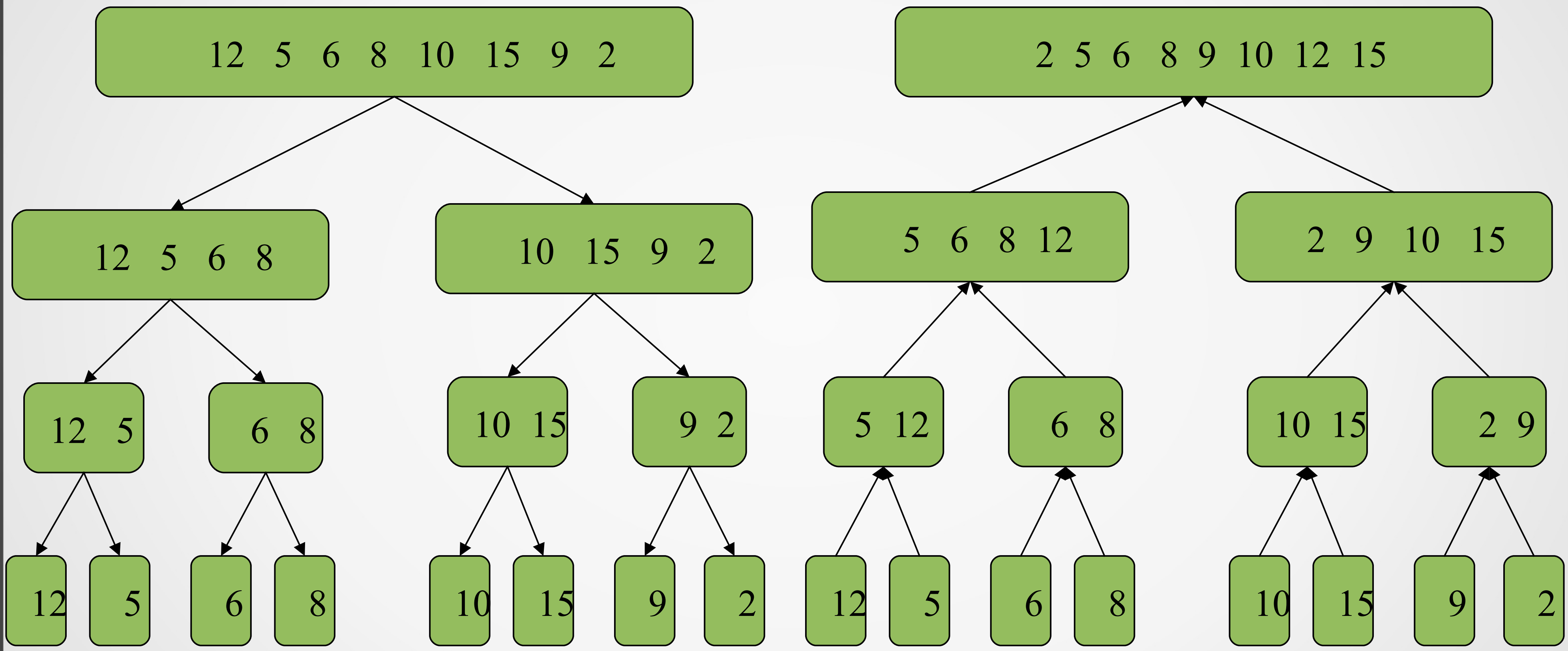
Merge Sort

- Uses divide and conquer strategy (multiple recursion) to sort a set of numbers
- Divide:
  - If S has zero or one element, return S
  - Divide S into two sequences S_1 and S_2 each containing half of the elements of S
- Recur
  - Recursively apply merge sort to S_1 and S_2
- Conquer
  - Merge S_1 and S_2 into a sorted sequence

Merging of Sorted Sequences

- Iteratively remove smallest element from one of the two sequences S_1 and S_2 and add it to end of output sequence S

Merge Sort


12 5 6 8 10 15 9 2
2 5 6 8 9 10 12 15
12 5 6 8
10 15 9 2
5 6 8 12
2 9 10 15
12 5
6 8
10 15
9 2
5 12
6 8
10 15
2 9
12
5
6
8
10
15
9
2
12
5
6
8
10
15
9
2

Merge Sort

- mergesort(S, low, high) {

```
    if (low < high) {
            middle = (low+high)/2;
            mergesort(s,low,middle);
            mergesort(s,middle+1,high);
            merge(s, low, middle, high);
        }
}
```

Src: Skiena, Algorithm Design Manual, Chapter 4