D-N-C: Maximum Subarray & Fast integer multiplication

CS-6^{th}

Instructor: Dr. Ayesha Enayet

Problem definition

- Given an array of size n, maximum subarray problem deals with finding a contiguous subarray with the largest sum.
- Example:
- Given an array A

a1	a2	a3	a4	a5	a6	a7
2	-3	7	-3	4	6	-10

- the maximum subarray is {a3,a4,a5,a6}

Exercise (maximum sum subarray)

- Brute-force solution?

```
BRUTE-FORCE-FIND-MAXIMUM-SUBARRAY(A)
    n = A.length
    max-sum = -∞
    for l = 1 to n
        sum = 0
        for h = l to n
            sum = sum + A[h]
            if sum > max-sum
                max-sum = sum
                low = l
                high = h
    return (low, high, max-sum)
```

Brute-force solution

- Total# of subarrays is given by:

$$\frac{n(n+1)}{2}$$

- The brute-force takes $O(n^2)$

Algorithm 1 Divide-Conquer-Combine Algorithm

```
function FINDMAXSUBARRAY(A, low, high)
    if low = high then
        return (low, high, A[low])
    else
        mid ← ⌊(low+high)/2⌋
        L ← FINDMAXSUBARRAY(A, low, mid)
        R ← FINDMAXSUBARRAY(A, mid + 1, high)
        C ← FMCS(A, low, mid, high)
        if L.maxSum >= R.maxSum and
            L.maxSum >= C.maxSum then
            return L
        else if R.maxSum >= L.maxSum and
            R.maxSum >= C.maxSum then
            return R
        else
            return C
        end if
    end if
end function
```

Algorithm 2 Find Maximum Crossing Subarray

```
function FMCS(A, low, mid, high)
    leftSum ← −∞
    sum ← 0
    for i ← mid downto low do
        sum ← sum + A[i]
        if sum > leftSum then
            leftSum ← sum
            maxLeft ← i
        end if
    end for
    rightSum ← −∞
    sum ← 0
    for j ← mid + 1 to high do
        sum ← sum + A[j]
        if sum > rightSum then
            rightSum ← sum
            maxRight ← j
        end if
    end for
    maxSum ← leftSum + rightSum
    return (maxLeft, maxRight, maxSum)
end function
```

1	2	3	4	5	6	7
-2	2	7	-3	4	6	-10
L			M			U

i=1 to 4

i=5 to 7

1	2	3	4
-2	2	7	-3
L	M		U

5	6	7
4	6	-10
L	M	U

1	2
-2	2
L	U

3	4
7	-3
L	U

5	6
4	6
L	U

7
-10
L/U

1
-2
L/U

2
2
L/U

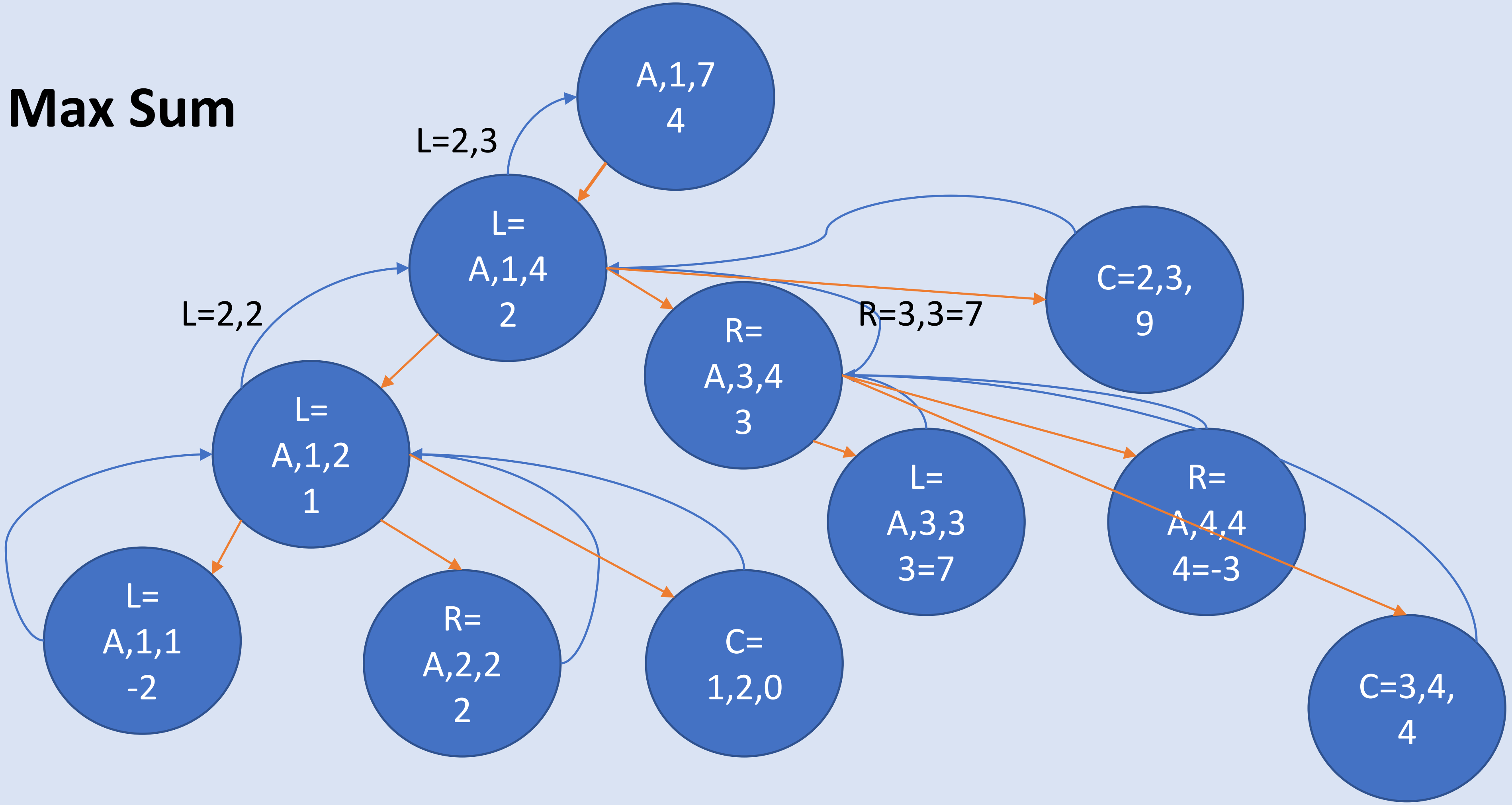
3
7
L/U

4
-3
L/U

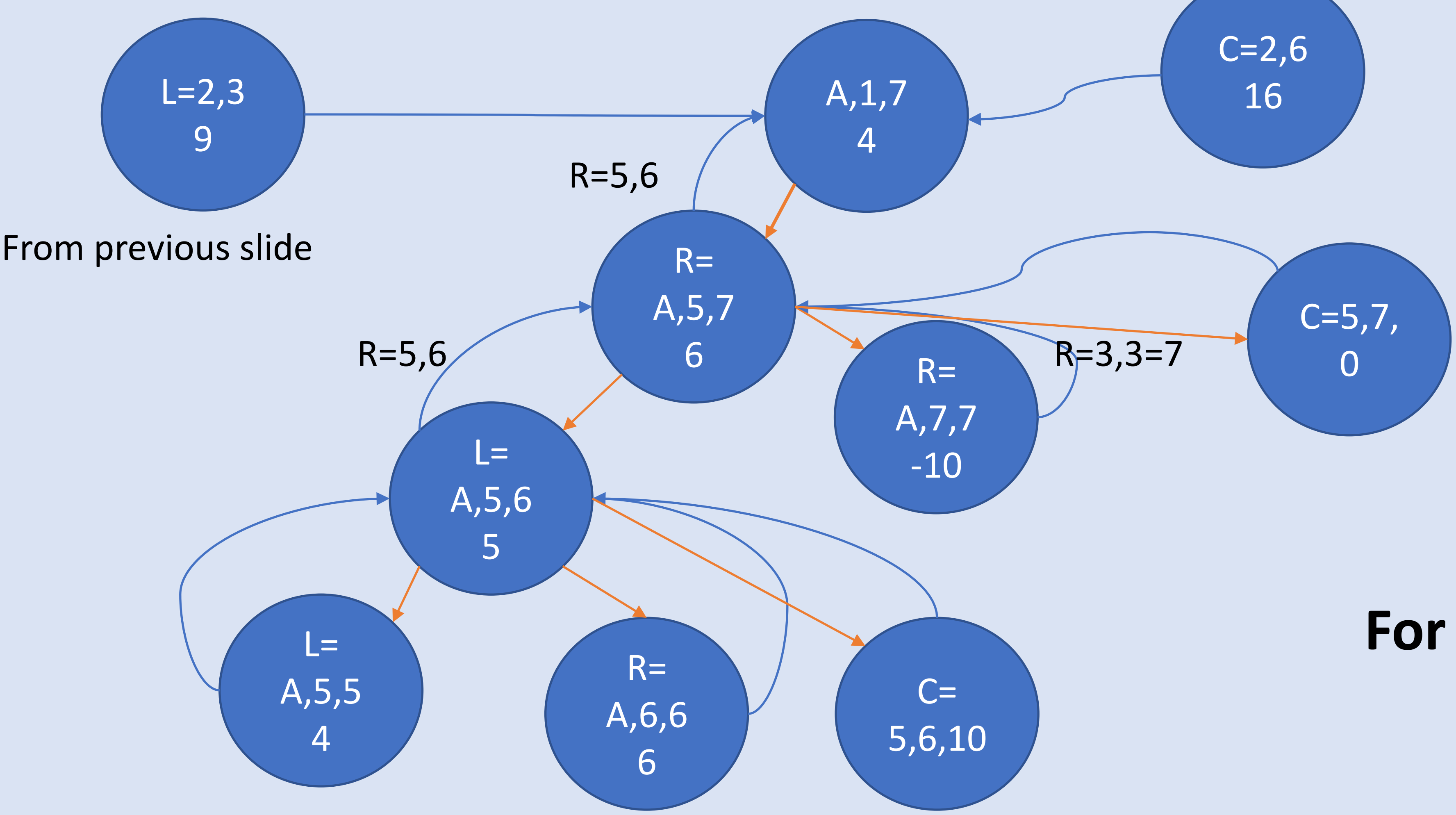
5
4
L/U

6
6
L/U

For Left Max Sum



1	2	3	4	5	6	7
-2	2	7	-3	4	6	-10
L			M			U



For Right Max Sum

1	2	3	4	5	6	7
-2	2	7	-3	4	6	-10
L			M			U

Time Complexity

- O(nlgn) [similar to merge sort]

Karatsuba algorithm

Karatsuba algorithm

- Karatsuba algorithm reduces the multiplication of two n-digit numbers to multiplication of three n/2 digit numbers, using divide-and-conquer approach, which reduces the time complexity from n^2 to $n^{lg_2 3}$.

- Input: two n digit numbers X and Y

$X=x_1B^m+x_0$ where m<n and B is the base, x1,x0,y1,y0 are n/2 digit numbers.

$Y=y_1B^m+y_0$

- Output: Z

$Z=X*Y=(x_1B^m+x_0)*(y_1B^m+y_0)$

$=x_1y_1(B^m)^2 + x_1y_0B^m + x_0y_1B^m + x_0y_0$

$= x_1y_1(B^m)^2 + (x_1y_0 + x_0y_1)B^m + x_0y_0 \rightarrow eq1$

T(n)=4T(n/2)+cn [The solution is $O(n^2)$]

Can we do better than $O(n^2)$

- $(x_0+x_1)(y_0+y_1)=x_0y_0+x_0y_1+x_1y_0+x_1y_1 \rightarrow$ eq2
- $Z=x_1y_1\,(B^m)^2$ **+(**$x_1\,y_0 + x_0\,y_1$**)** $B^m + x_0\,y_0 \rightarrow$ eq1
- $x_1\,y_0 + x_0\,y_1 = (x_0+x_1)(y_0+y_1)- x_0y_0- x_1y_1$ [we already have values of x_0y_0 and x_1y_1 from eq1]
- So we have 3 multiplication operations:
  1. x_0y_0
  2. x_1y_1
  3. $(x_0+x_1)(y_0+y_1)$
- $Z=x_1y_1\,(B^m)^2$ **+(**$(x_0+x_1)(y_0+y_1)- x_0y_0- x_1y_1$**)** $B^m + x_0\,y_0 \rightarrow$ eq1
- $T(n)=3T(n/2)+cn$

- T(n)=3T(n/2)+cn [by master theorem the solution is $n^{lg_2 3}$ which is $n^{1.59}$]
- T(n)=3^2T(n/2^2)+c(n/2)+cn
- T(n)=3^3T(n/2^3)+ c(n/2^2)+c(n/2)+cn
- T(n)=3^kT(n/2^k)+ cn(1/2^{k-1}+1/2^{k-2} …1)
- K=logn
- T(n)=3^{logn} T(1)+ cn(1/2^{k-1}+1/2^{k-2} …1)
- T(n)= $n^{lg_2 3}$ + cn(1/2^{k-1}+1/2^{k-2} …1)
- T(n)=O($n^{lg_2 3}$)