Recursion

CSC212: Data Structures

Recursion

- **Sometimes, certain statements in an algorithm are repeated on different sizes of an input instance.**
- **Repetition can be achieved in two different ways.**
  - **Iteration: uses for and while loops**
  - **Recursion: function calls itself**

Example -1:

– Factorial Function

–Factorial function of any integer n is defined as

$$n! = \begin{cases} 1 \text{ if } n = 0 & \leftarrow \text{ Base Case} \\ n(n-1)! \text{ if } n \geq 1 & \leftarrow \text{ Recusion Case} \end{cases}$$

– This is recursive definition. It consists of two parts:

i. **Base case**

ii. **Recursive case**

Recursion

Example -1 (Continue) :

– It can be written as:

$$\text{fact}(n) = \begin{cases} 1 \text{ if } n = 0 & \leftarrow \text{ Base Case} \\ n\text{fact}(n-1)! \text{ if } n \geq 1 & \leftarrow \text{ Recusion Case} \end{cases}$$

where fact(n) is the function that calculates $n!$.

Recursion

Example -1 (Continue) :

– Implementation

Recursive:

```
public static int recursiveFact(int n)
{
     if(n==0)return 1;
     else
        return n*recursiveFact(n-1);
}
```

Iterative:

```
public static int iterativeFact(int n)
{
     int fact = 1;
     for(i = 1; i <= n; i++)
        fact=fact*I;
     return fact;
}
```

Recursive Trace

- A graphical representation of recursive calls.
- It is used to analyze the algorithm.

Example -2:

- Recursive trace for recursiveFact(4)

```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- A graphical representation of recursive calls.
- It is used to analyze the algorithm.

Example -2:

- Recursive trace for recursiveFact(4)

recursiveFact(4)

4*?=?

```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- A graphical representation of recursive calls.
- It is used to analyze the algorithm.

Example -2:

- Recursive trace for recursiveFact(4)

recursiveFact(4) 4*?=?

recursiveFact(3) 3*?=?

```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- ❖ A graphical representation of recursive calls.
- ❖ It is used to analyze the algorithm.

Example -2:

– Recursive trace for recursiveFact(4)

recursiveFact(4) 4*?=?

↓

recursiveFact(3) 3*?=?

↓

recursiveFact(2) 2*?=?

```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- ❖ A graphical representation of recursive calls.
- ❖ It is used to analyze the algorithm.

Example -2:

– Recursive trace for recursiveFact(4)

recursiveFact(4) — 4*?=?

↓

recursiveFact(3) — 3*?=?

↓

recursiveFact(2) — 2*?=?

↓

recursiveFact(1) — 1*?=?

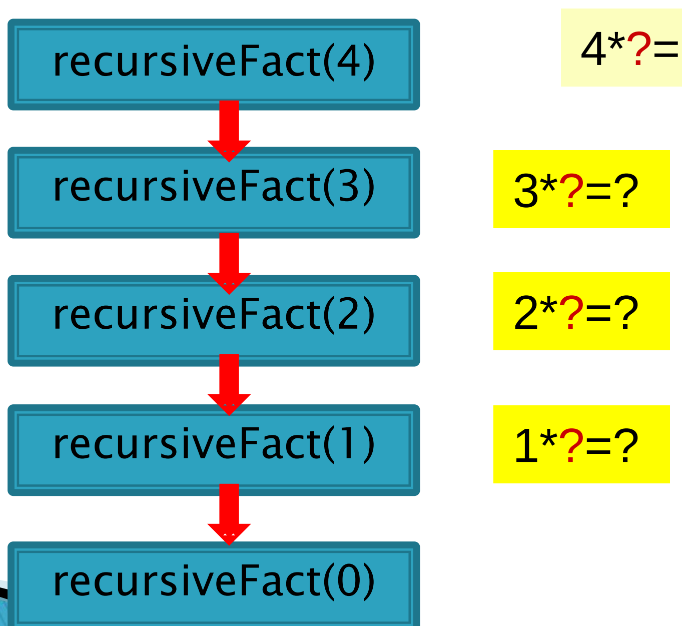
```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- A graphical representation of recursive calls.
- It is used to analyze the algorithm.

Example -2:

– Recursive trace for recursiveFact(4)



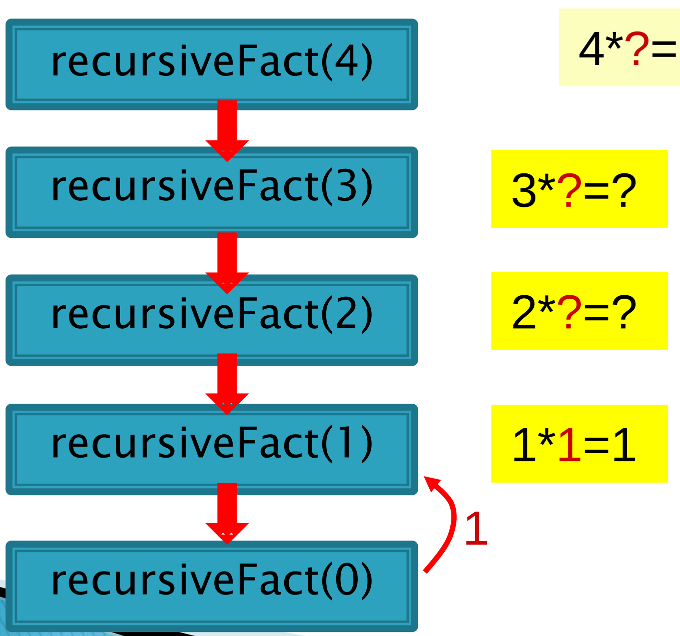

```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- A graphical representation of recursive calls.
- It is used to analyze the algorithm.

Example -2:

– Recursive trace for recursiveFact(4)



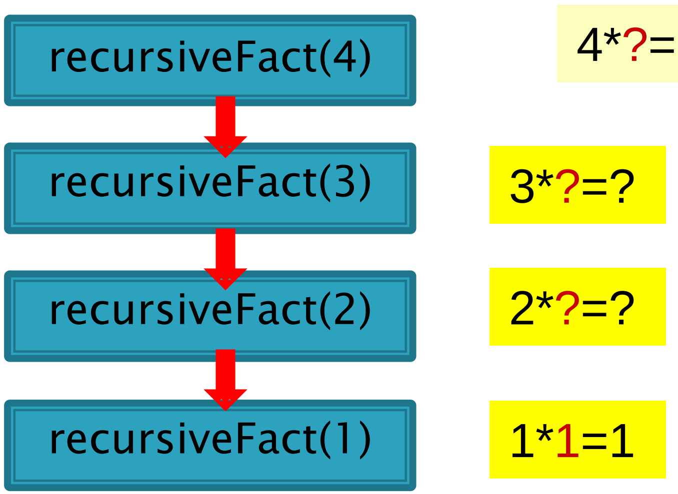

```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- A graphical representation of recursive calls.
- It is used to analyze the algorithm.

Example -2:

– Recursive trace for recursiveFact(4)



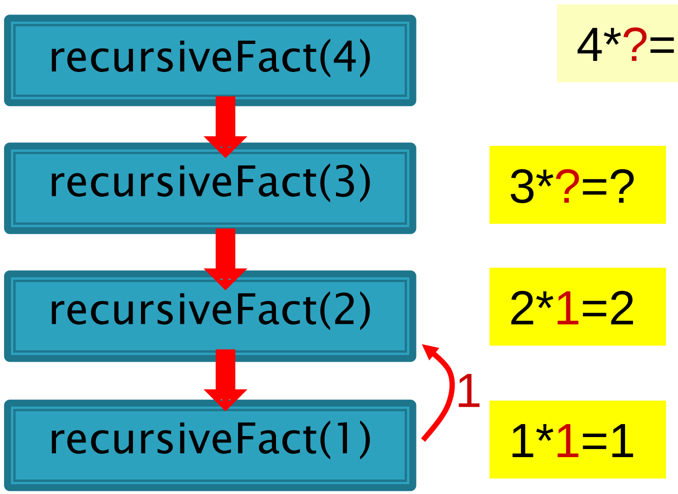

```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- A graphical representation of recursive calls.
- It is used to analyze the algorithm.

Example -2:

– Recursive trace for recursiveFact(4)



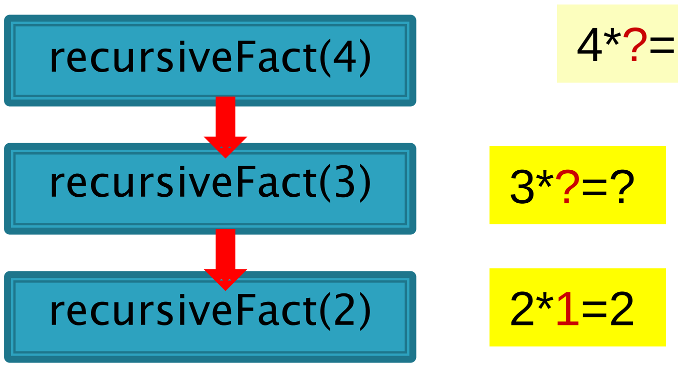

```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- ❖ A graphical representation of recursive calls.
- ❖ It is used to analyze the algorithm.

Example -2:

– Recursive trace for recursiveFact(4)



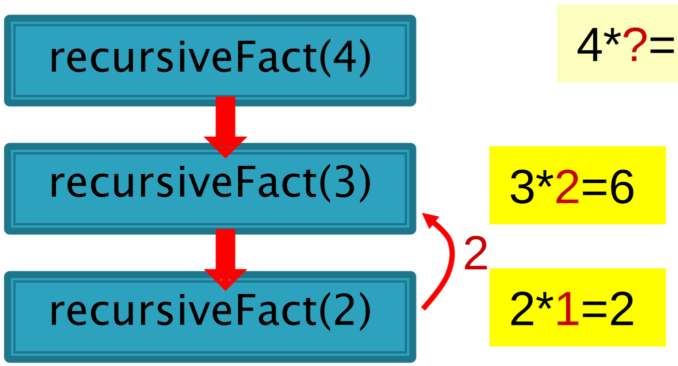

```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- ❖ A graphical representation of recursive calls.
- ❖ It is used to analyze the algorithm.

Example -2:

– Recursive trace for recursiveFact(4)




```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- A graphical representation of recursive calls.
- It is used to analyze the algorithm.

Example -2:

- Recursive trace for recursiveFact(4)

recursiveFact(4)

4*?=?

recursiveFact(3)

3*2=6

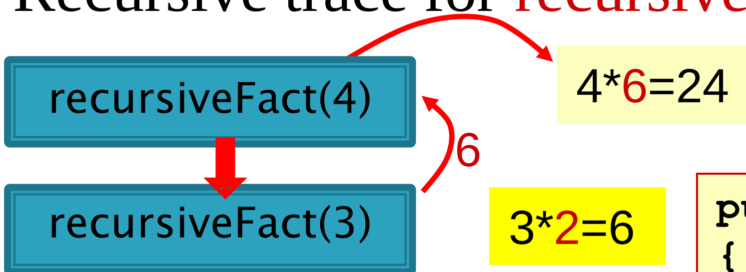
```
public static int recursiveFact(int n)
{
          if(n==0)
            return 1;
          else
            return n*recursiveFact(n-1);
}
```

Recursive Trace

- ❖ A graphical representation of recursive calls.
- ❖ It is used to analyze the algorithm.

Example -2:

– Recursive trace for recursiveFact(4)




```
public static int recursiveFact(int n)
{
        if(n==0)
          return 1;
        else
          return n*recursiveFact(n-1);
}
```

Recursive Trace

- A graphical representation of recursive calls.
- It is used to analyze the algorithm.

Example -2:

– Recursive trace for recursiveFact(4)

recursiveFact(4)

4*6=24

```
public static int recursiveFact(int n)
{
          if(n==0)
            return 1;
          else
            return n*recursiveFact(n-1);
}
```

Main Types of Recursion

- Linear Recursion
- Binary Recursion

Linear Recursion

In this case a recursive method makes at most one recursive call each time it is invoked.

Example – 3:

- **Problem:** Given an array A of n integers, find the sum of first n integers.
- **Observation:** Sum can be defined recursively as follows:

$$\mathrm{Sum}(n) = \begin{cases} \mathrm{A}[0] \text{ if } n = 0 & \leftarrow \text{ Base Case} \\ \mathrm{Sum}(n-1) + \mathrm{A}[n-1] \text{ if } n \geq 1 & \leftarrow \text{ Recusive Case} \end{cases}$$

Linear Recursion

Example – 3 (Continued):

– **Algorithm**

Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

Processing:

```
if n = 1;
    return A[0];                    → base case
else
    return Sum(A, n-1) + A[n-1];    → recursive case.
```

Note:

- Base case should be defined so that every possible chain of recursive calls eventually reach a base case.
- Algorithm must start by testing a set of base cases.
- After testing for base cases perform a single recursive call.

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5

	A
0	4
1	3
2	6
3	2
4	5

Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

Processing:

```
if n = 1;
    return A[0];              → base case
else
return Sum(A, n-1) + A[n-1];          → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5

Sum(A,5)

??+A[4]=??+5=?

A

0	4
1	3
2	6
3	2
4	5

Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

Processing:

```
if n = 1;
    return A[0];                → base case
else
return Sum(A, n-1) + A[n-1];           → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5

Sum(A,5) ↓ Sum(A,4)

??+A[4]=??+5=?

??+A[3]=??+2=?

	A
0	4
1	3
2	6
3	2
4	5

Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

Processing:

```
if n = 1;
    return A[0];            → base case
else
return Sum(A, n-1) + A[n-1];            → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5

Sum(A,5)

Sum(A,4)

Sum(A,3)

??+A[4]=??+5=?

??+A[3]=??+2=?

?+A[2]=?+6=?

A

0	4
1	3
2	6
3	2
4	5

Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

Processing:

```
if n = 1;
    return A[0];                → base case
else
return Sum(A, n-1) + A[n-1];         → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5

Sum(A,5)	??+A[4]=??+5=?
Sum(A,4)	??+A[3]=??+2=?
Sum(A,3)	?+A[2]=?+6=?
Sum(A,2)	?+A[1]=?+3=?

	A
0	4
1	3
2	6
3	2
4	5

Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

Processing:

```
if n = 1;
    return A[0];                 → base case
else
return Sum(A, n-1) + A[n-1];          → recursive case.
```

Linear Recursion

Example – 3 (Continued):

- Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5

Sum(A,5)	??+A[4]=??+5=?
Sum(A,4)	??+A[3]=??+2=?
Sum(A,3)	?+A[2]=?+6=?
Sum(A,2)	?+A[1]=?+3=?
Sum(A,1)	A[0]=4

	A
0	4
1	3
2	6
3	2
4	5

Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

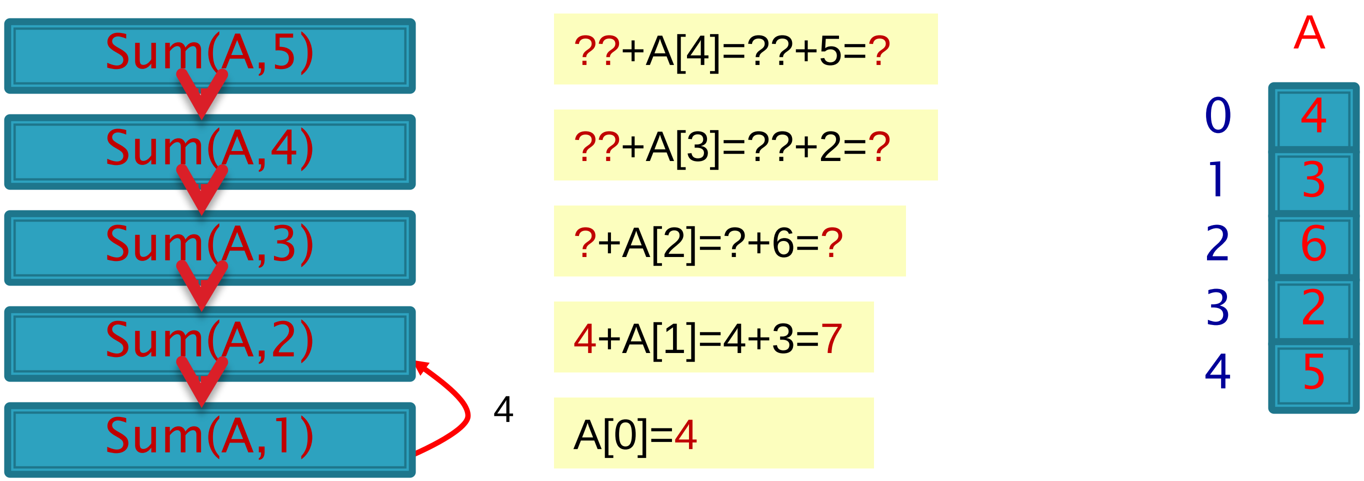
Processing:

```
if n = 1;
    return A[0];          → base case
else
return Sum(A, n-1) + A[n-1];          → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5



Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

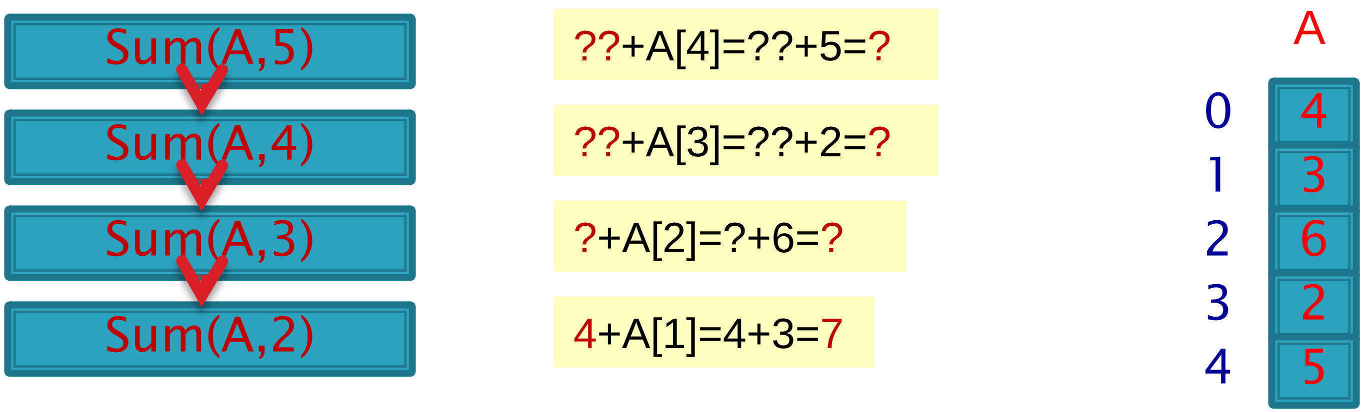
Processing:

```
if n = 1;
    return A[0];              → base case
else
return Sum(A, n-1) + A[n-1];          → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5



Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

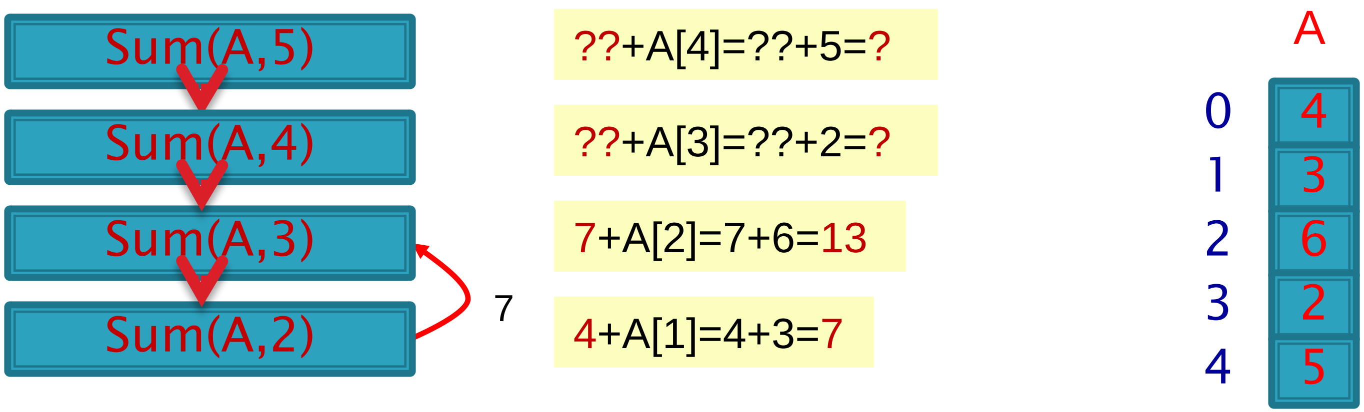
Processing:

```
if n = 1;
    return A[0];            → base case
else
return Sum(A, n-1) + A[n-1];        → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5



Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

Processing:

```
if n = 1;
    return A[0];              → base case
else
return Sum(A, n-1) + A[n-1];          → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5

Sum(A,5)	??+A[4]=??+5=?
Sum(A,4)	??+A[3]=??+2=?
Sum(A,3)	7+A[2]=7+6=13

A

0	4
1	3
2	6
3	2
4	5

Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

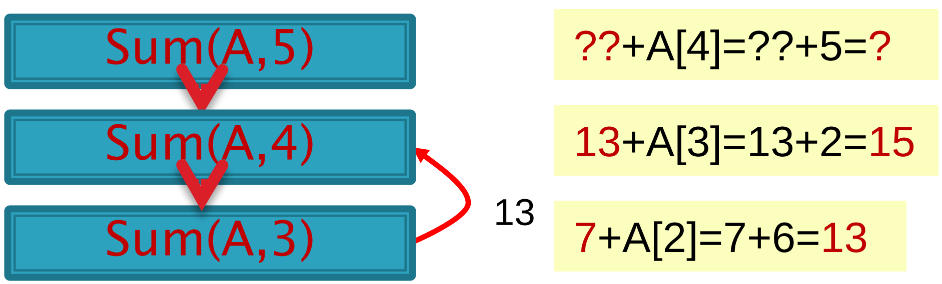
Processing:

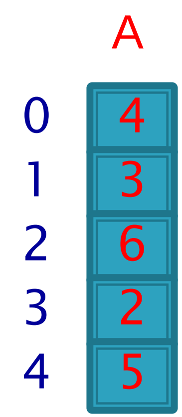
```
if n = 1;
    return A[0];                → base case
else
return Sum(A, n-1) + A[n-1];            → recursive case.
```

Linear Recursion

Example – 3 (Continued):

- Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5





Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

Processing:

```
if n = 1;
    return A[0];              → base case
else
return Sum(A, n-1) + A[n-1];          → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5

Sum(A,5) → ??+A[4]=??+5=?

Sum(A,4) → 13+A[3]=13+2=15

A	
0	4
1	3
2	6
3	2
4	5

Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

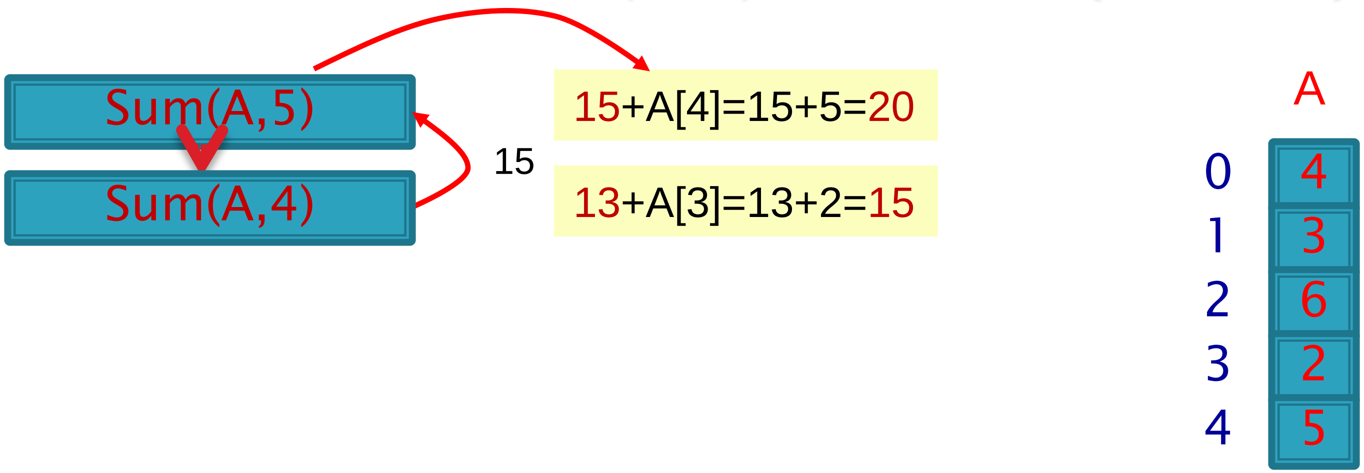
Processing:

```
if n = 1;
    return A[0];            → base case
else
return Sum(A, n-1) + A[n-1];          → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5



Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

Processing:

```
if n = 1;
      return A[0];                → base case
else
return Sum(A, n-1) + A[n-1];             → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5

Sum(A,5) → 15+A[4]=15+5=20

A

0	4
1	3
2	6
3	2
4	5

Sum(A, n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

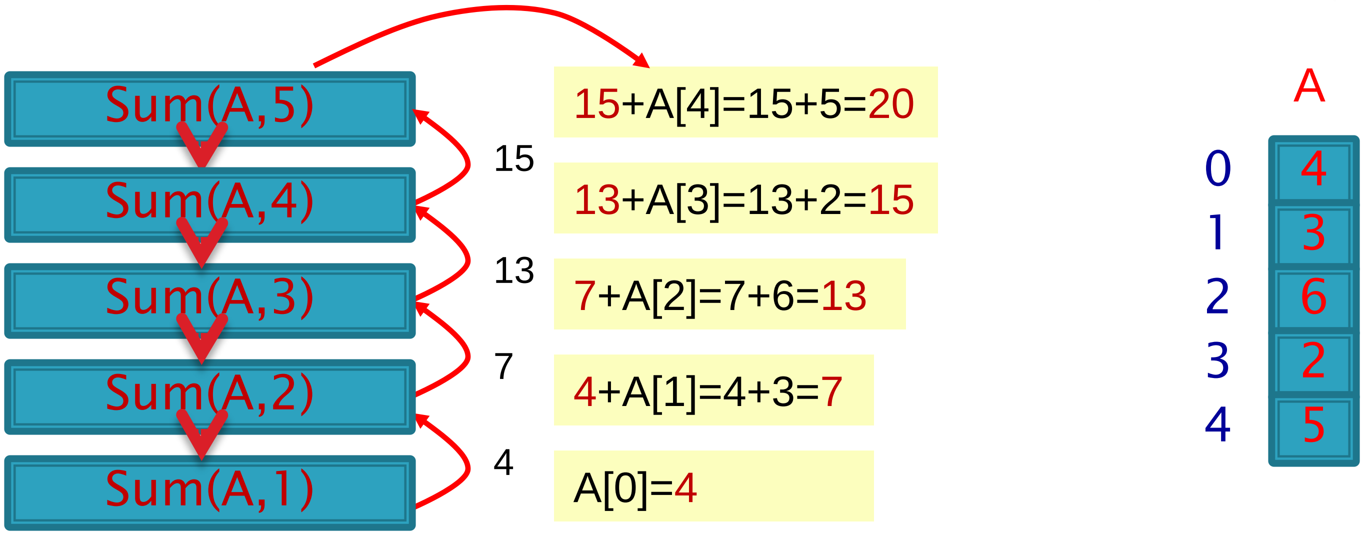
Processing:

```
if n = 1;
    return A[0];          → base case
else
return Sum(A, n-1) + A[n-1];          → recursive case.
```

Linear Recursion

Example – 3 (Continued):

– Recursive trace for sum(A ,n), where A ={4,3,6,2,5}, n=5



Note:

- For an array of size *n*, Sum(A, *n*) makes n calls.
- Each spends a constant amount of time.
- So time complexity is O(*n*).

Binary Recursion

In this case, a recursive algorithm makes two recursive calls.

Example – 4

– **Problem:** Find the sum of n elements of an integer array A.

– **Algorithm:**

- Recursively find the sum of elements in the first half of A.
- Recursively find the sum of elements in the second half of A.
- Add these two values

	A
0	6
1	5
2	3
3	2
4	8

BinarySum(A ,i ,n)

Input: An integer array A and an integer $n \geq 1$, such that A has at least n elements

Output: The sum of the first n integers in A.

Processing:

if n =1
  return A[i];
Else
return BinarySum(A ,i, $\lceil n/2 \rceil$)+BinarySum(A ,i+$\lceil n/2 \rceil$, $\lfloor n/2 \rfloor$);

Binary Recursion

Example – 4 (Continued):

– Recursive trace

A

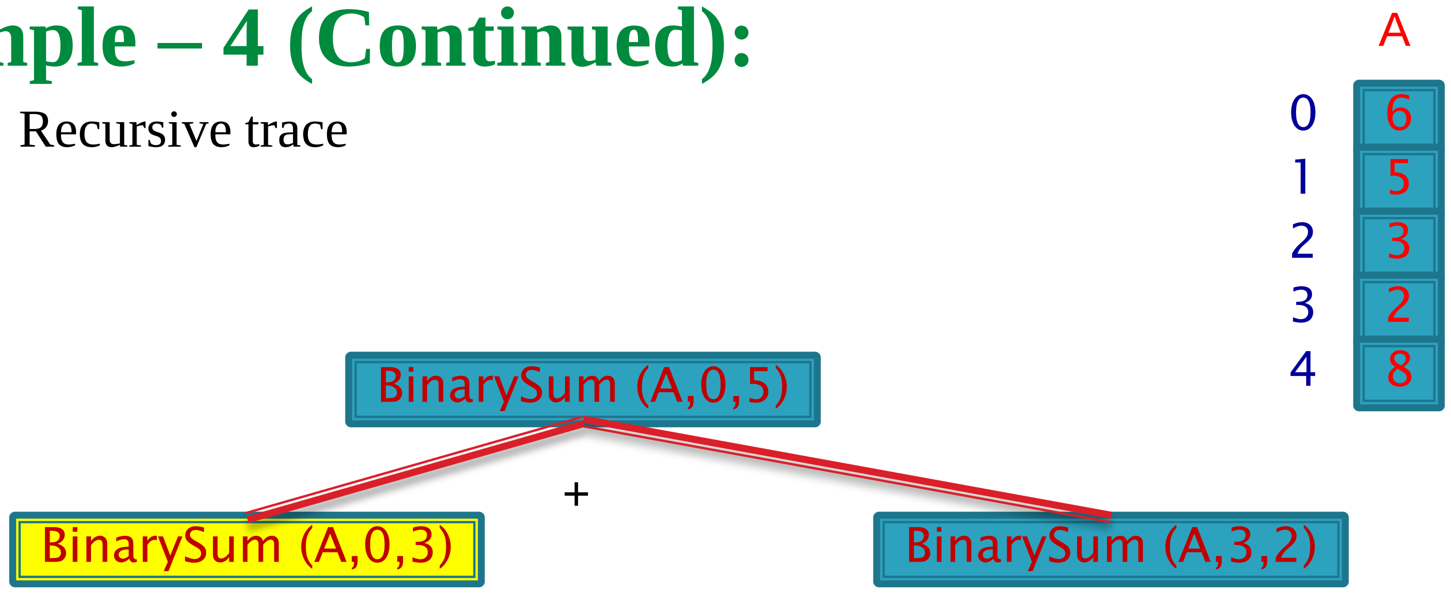
	A
0	6
1	5
2	3
3	2
4	8

BinarySum (A,0,5)

Binary Recursion

Example – 4 (Continued):

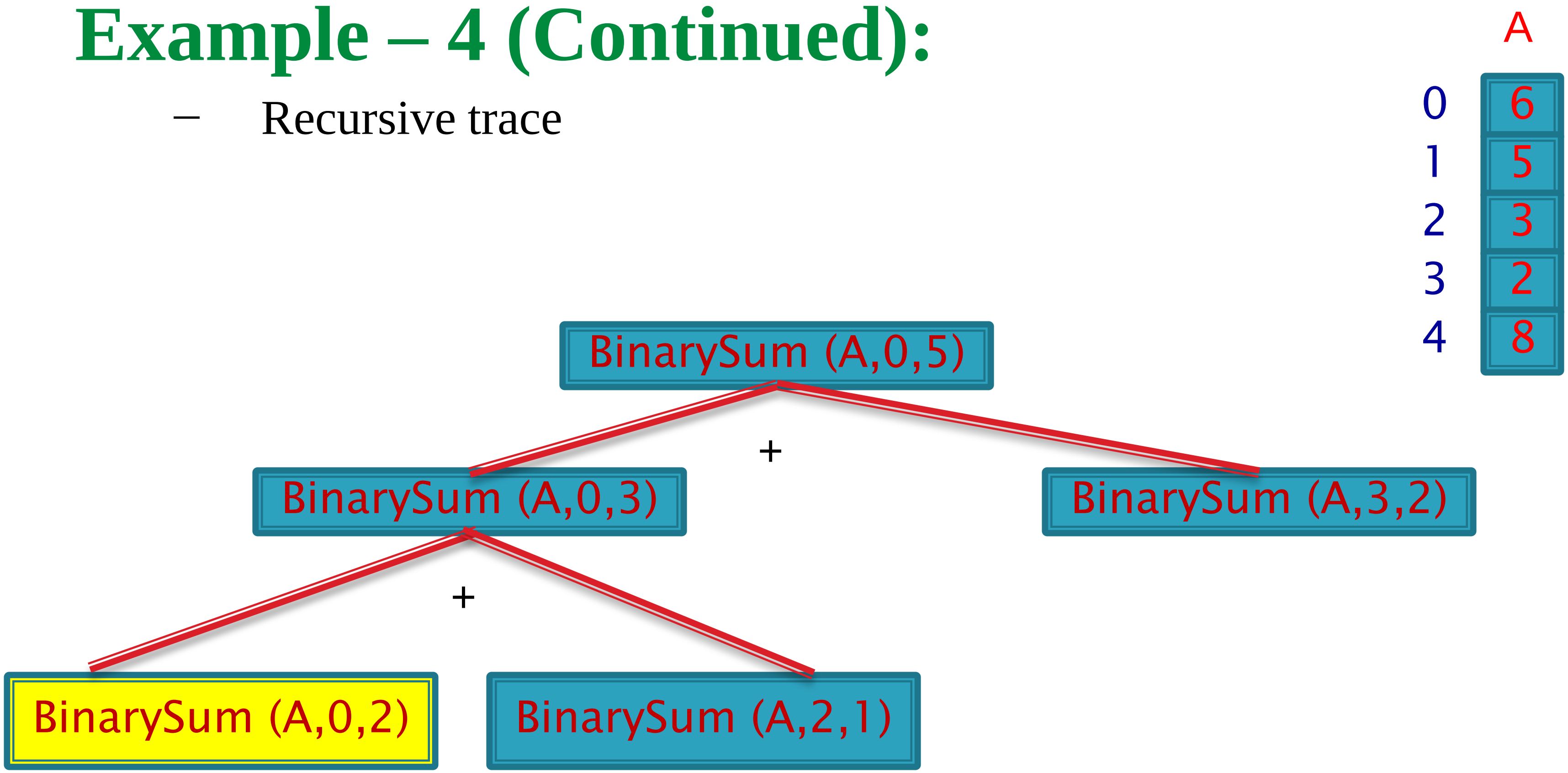
- Recursive trace



Binary Recursion

Example – 4 (Continued):

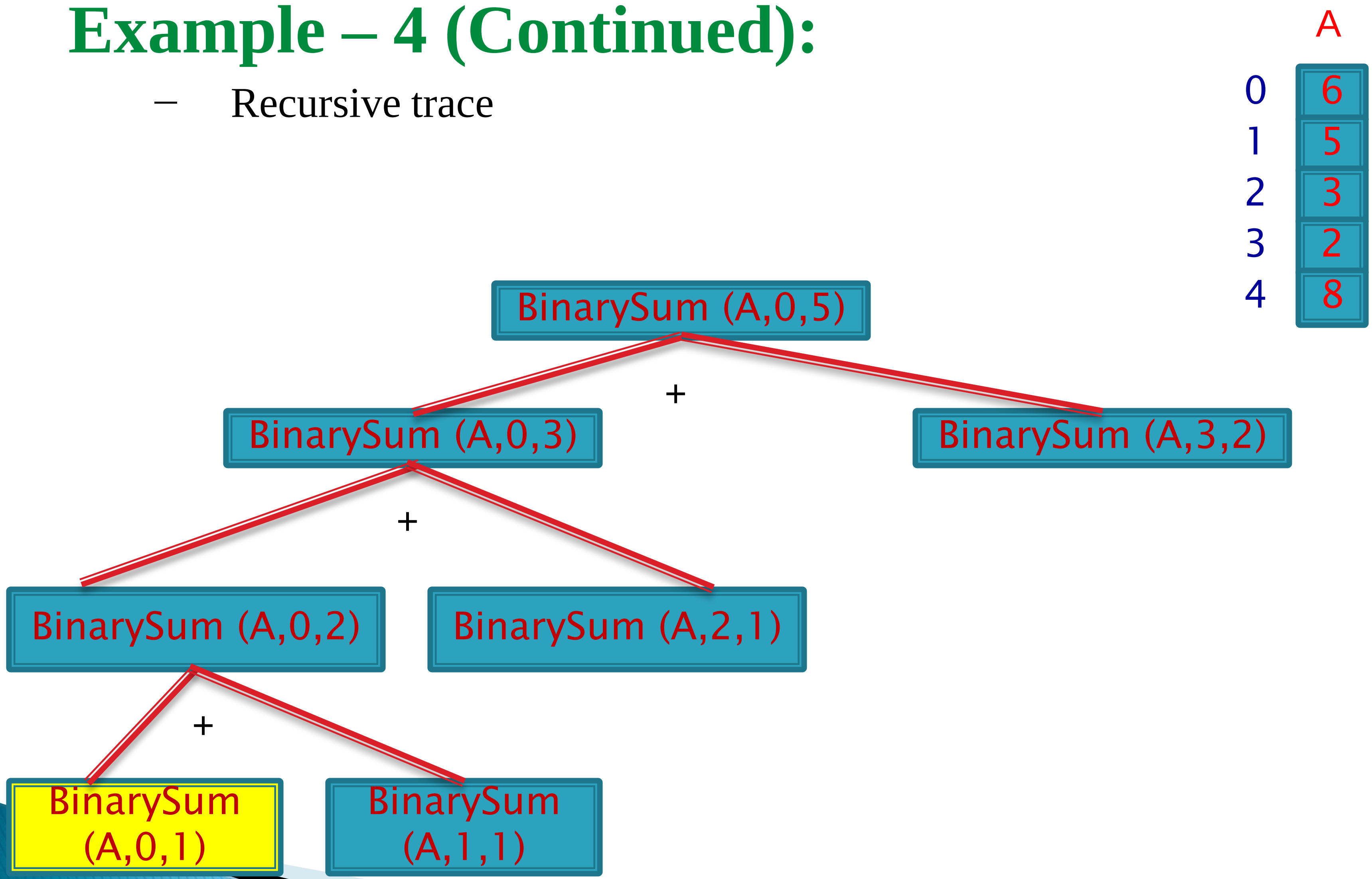
– Recursive trace



Binary Recursion

Example – 4 (Continued):

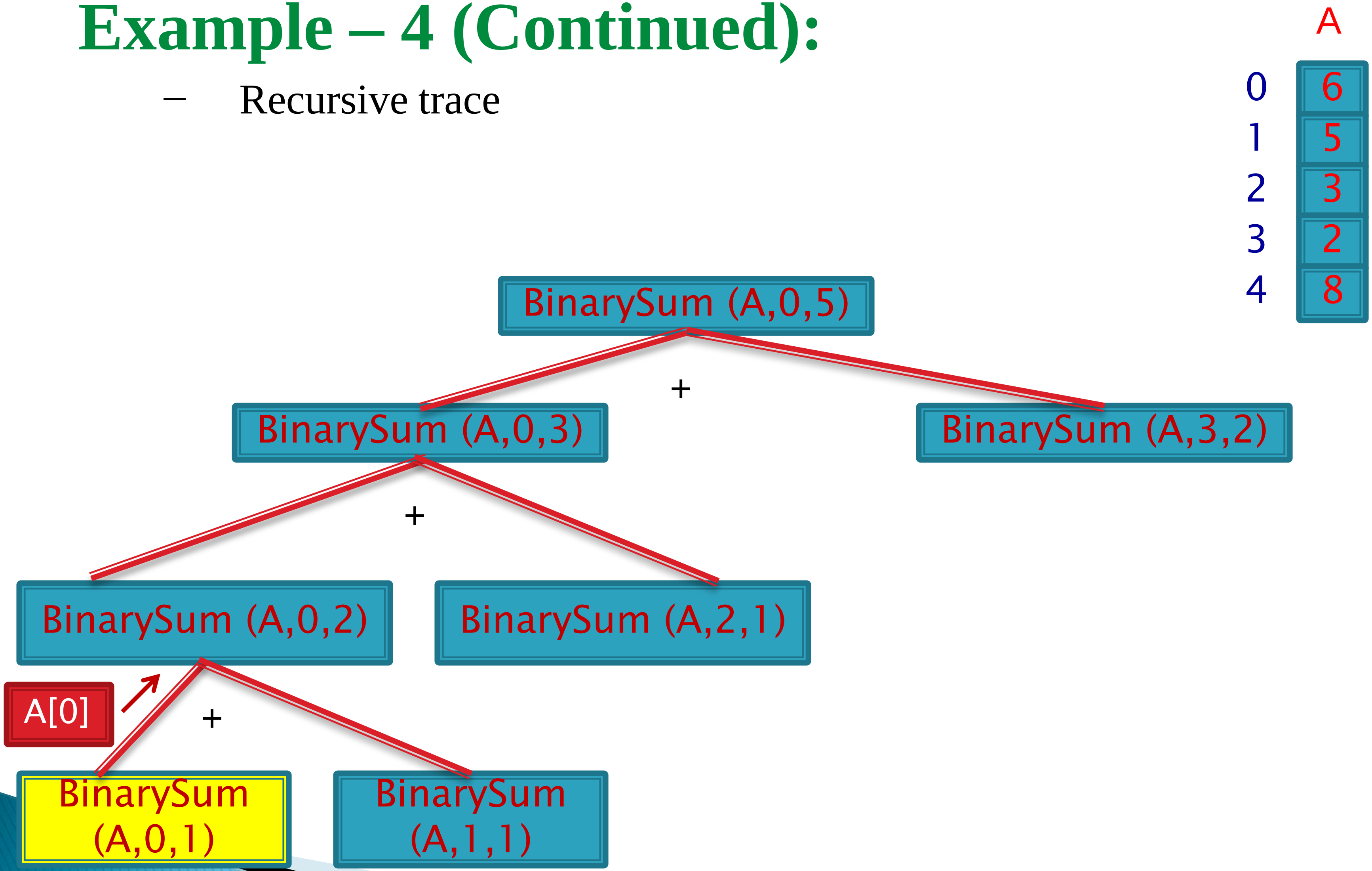
– Recursive trace



Binary Recursion

Example – 4 (Continued):

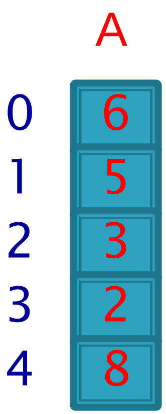
- Recursive trace

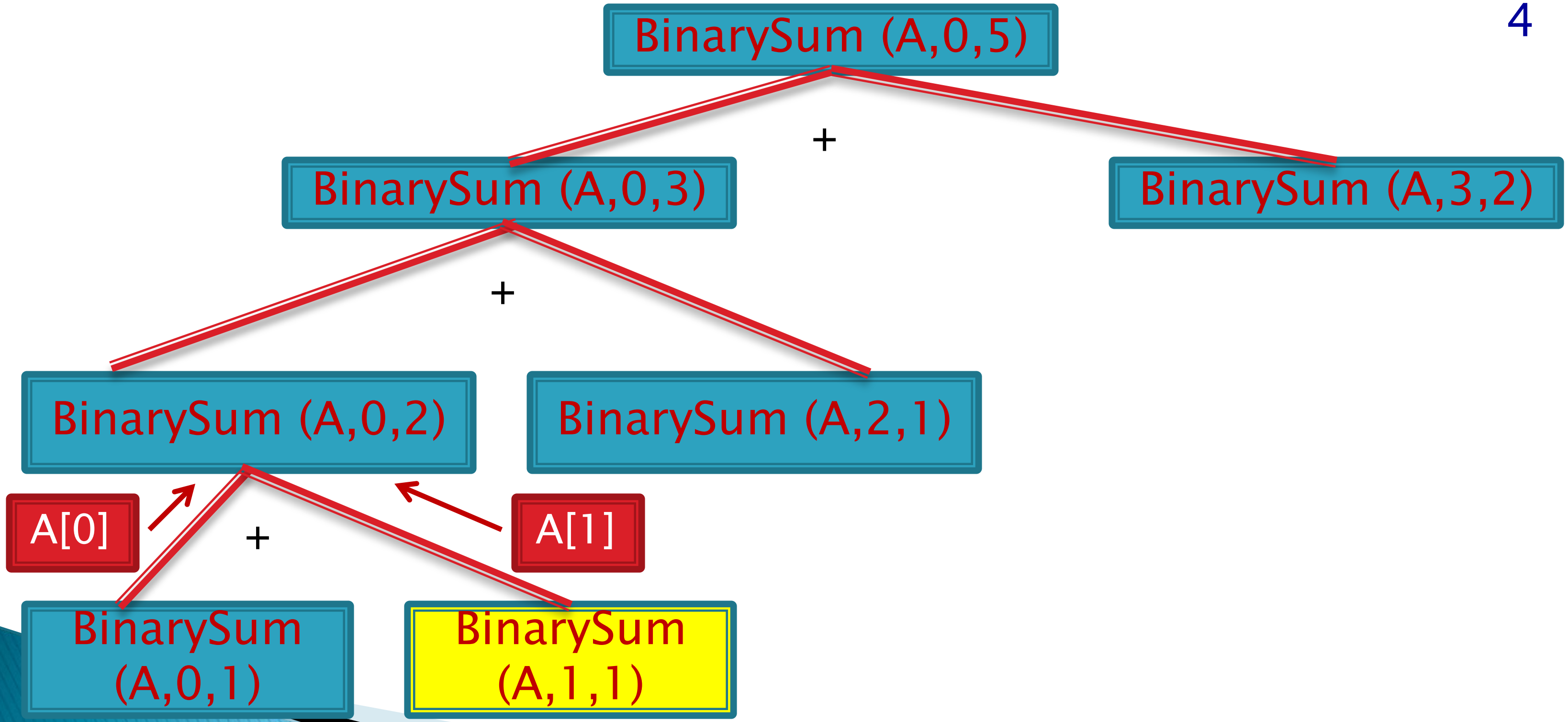


Binary Recursion

Example – 4 (Continued):

– Recursive trace

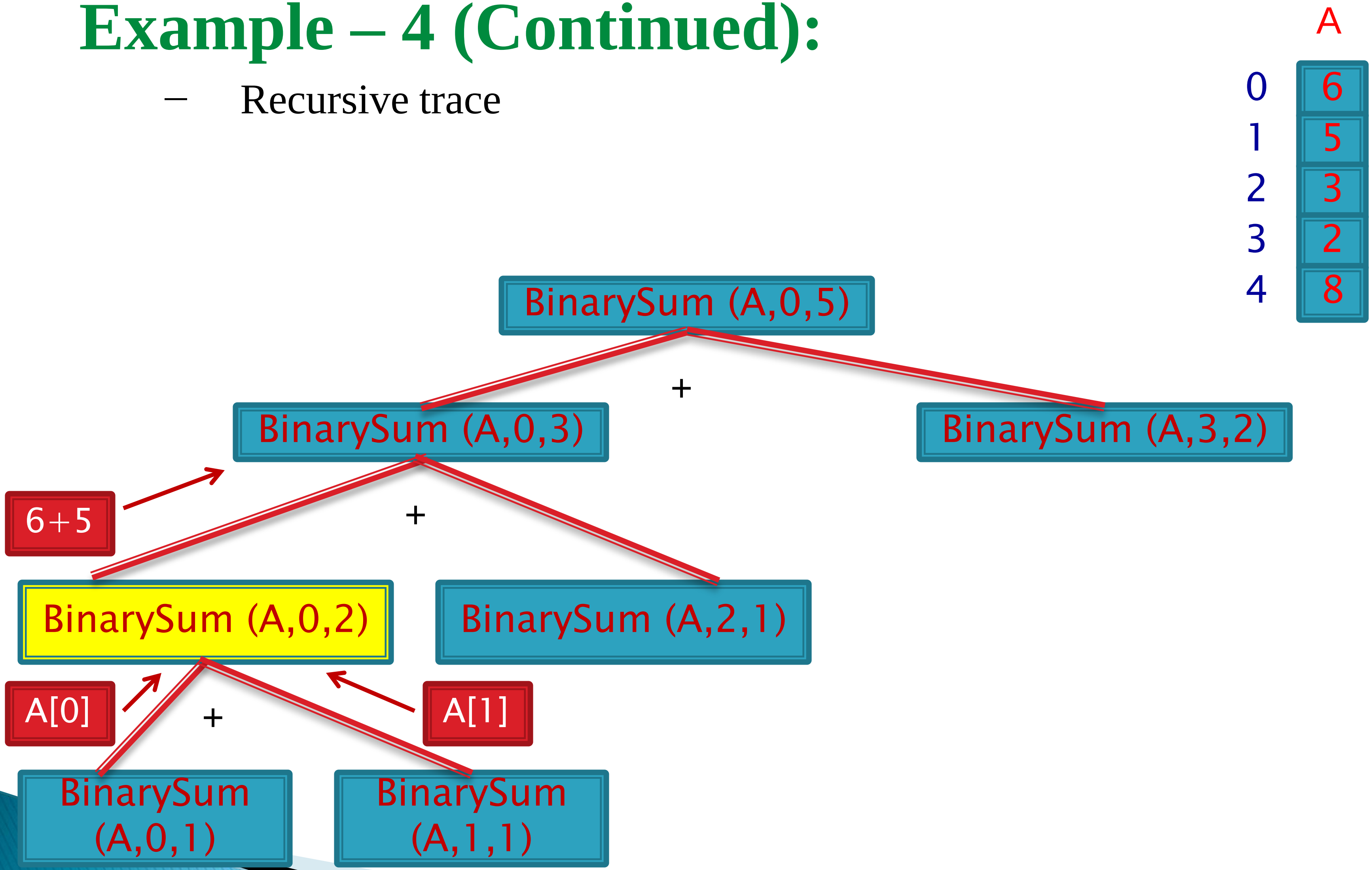




Binary Recursion

Example – 4 (Continued):

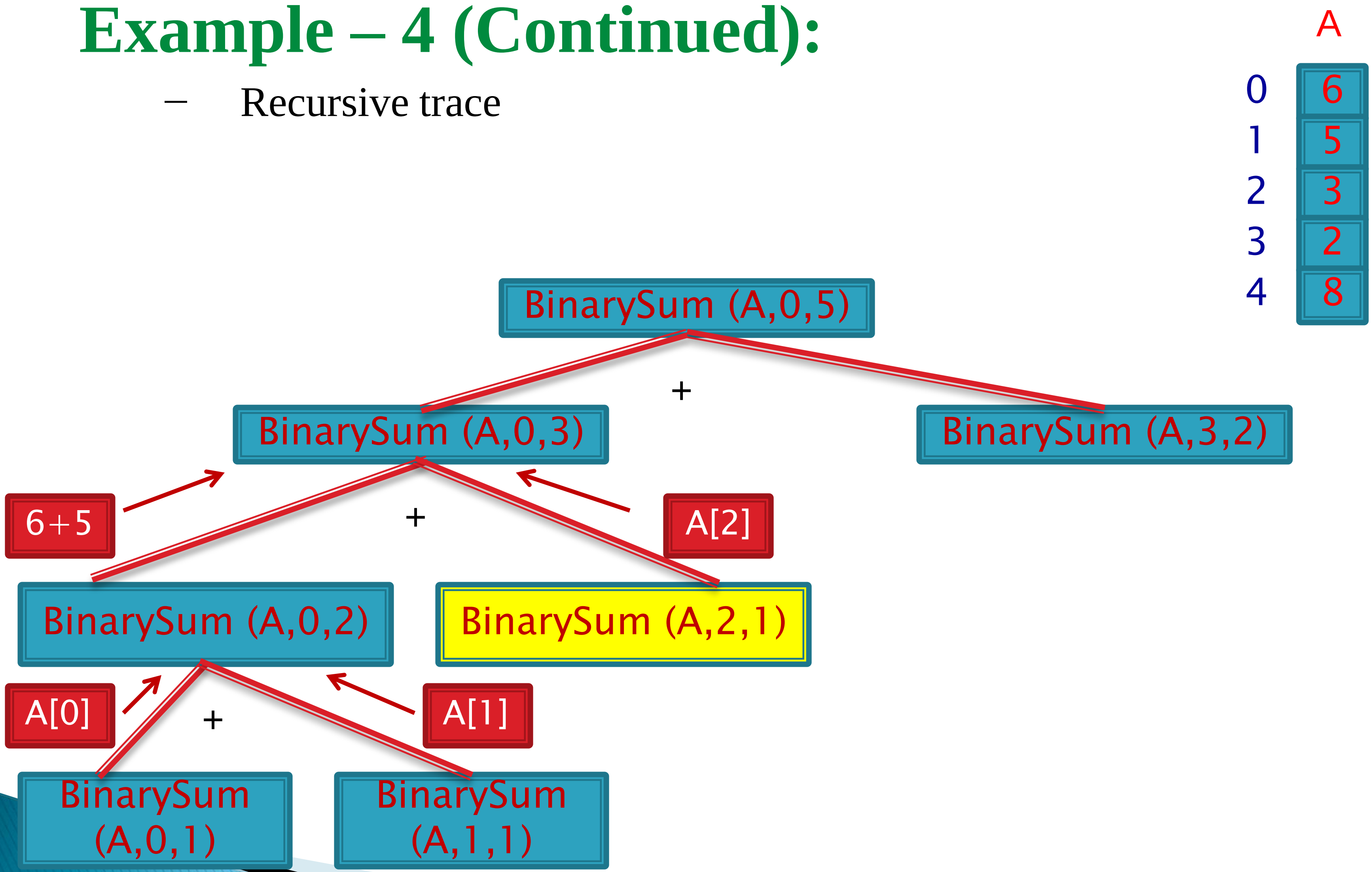
- Recursive trace



Binary Recursion

Example – 4 (Continued):

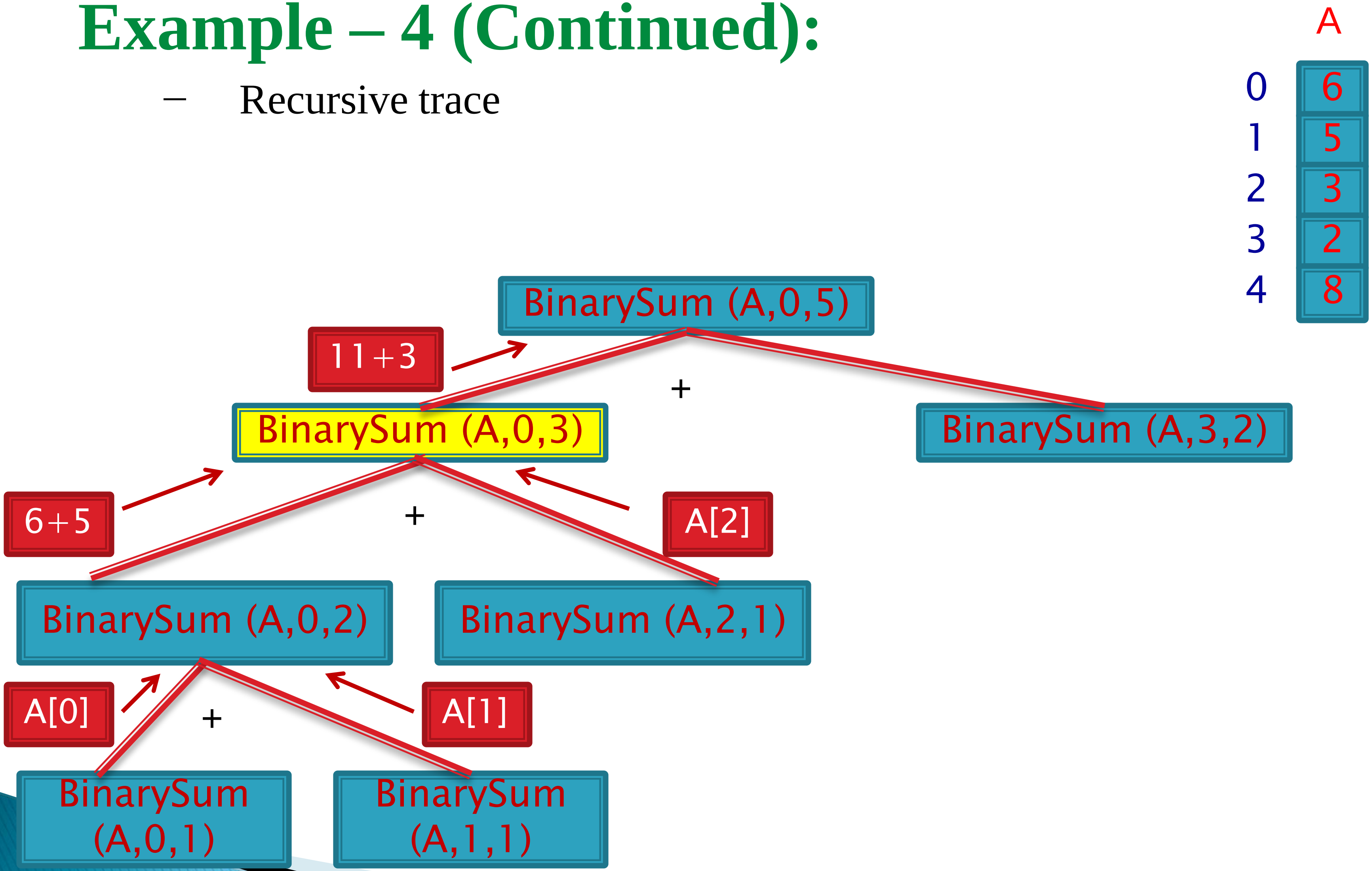
– Recursive trace



Binary Recursion

Example – 4 (Continued):

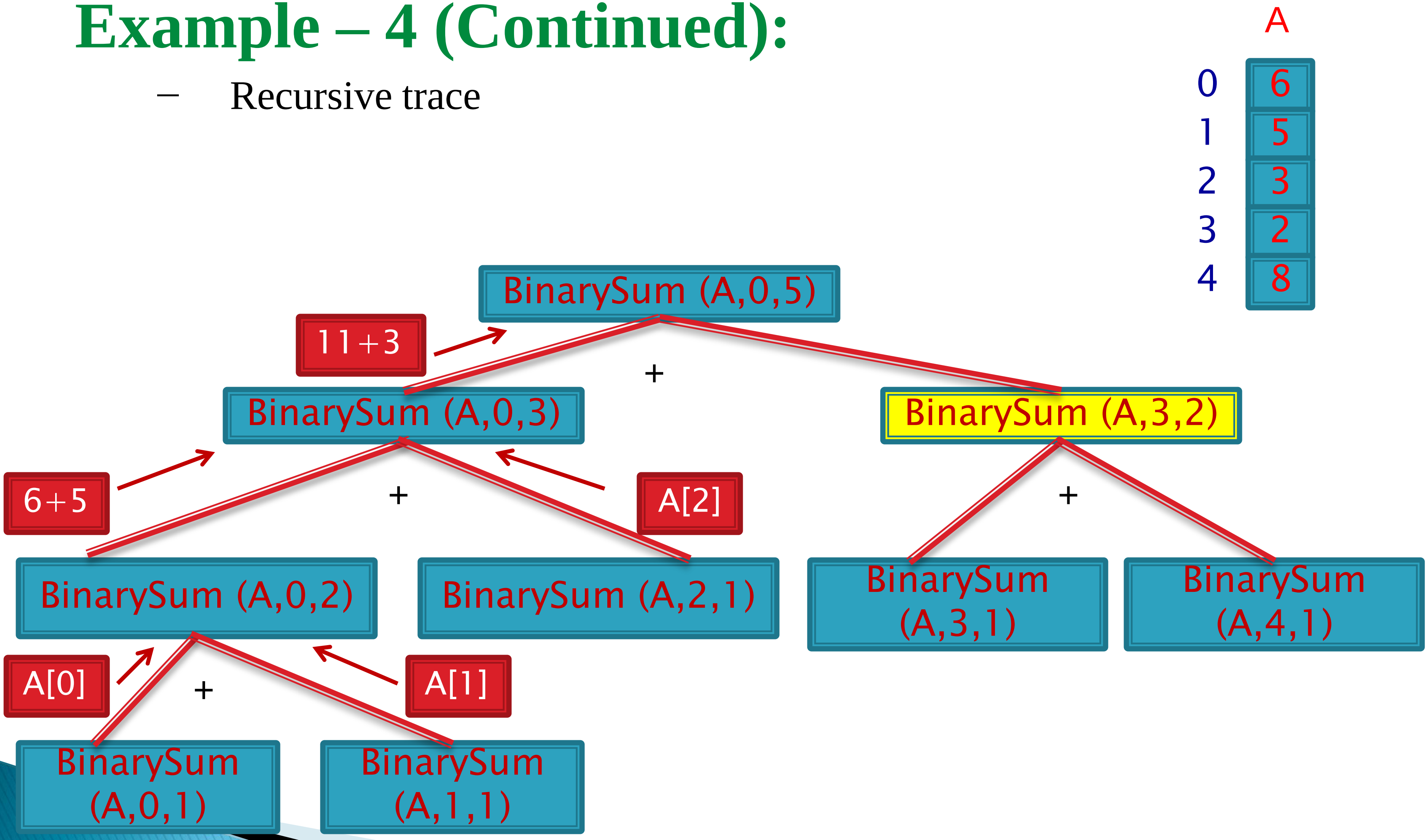
- Recursive trace



Binary Recursion

Example – 4 (Continued):

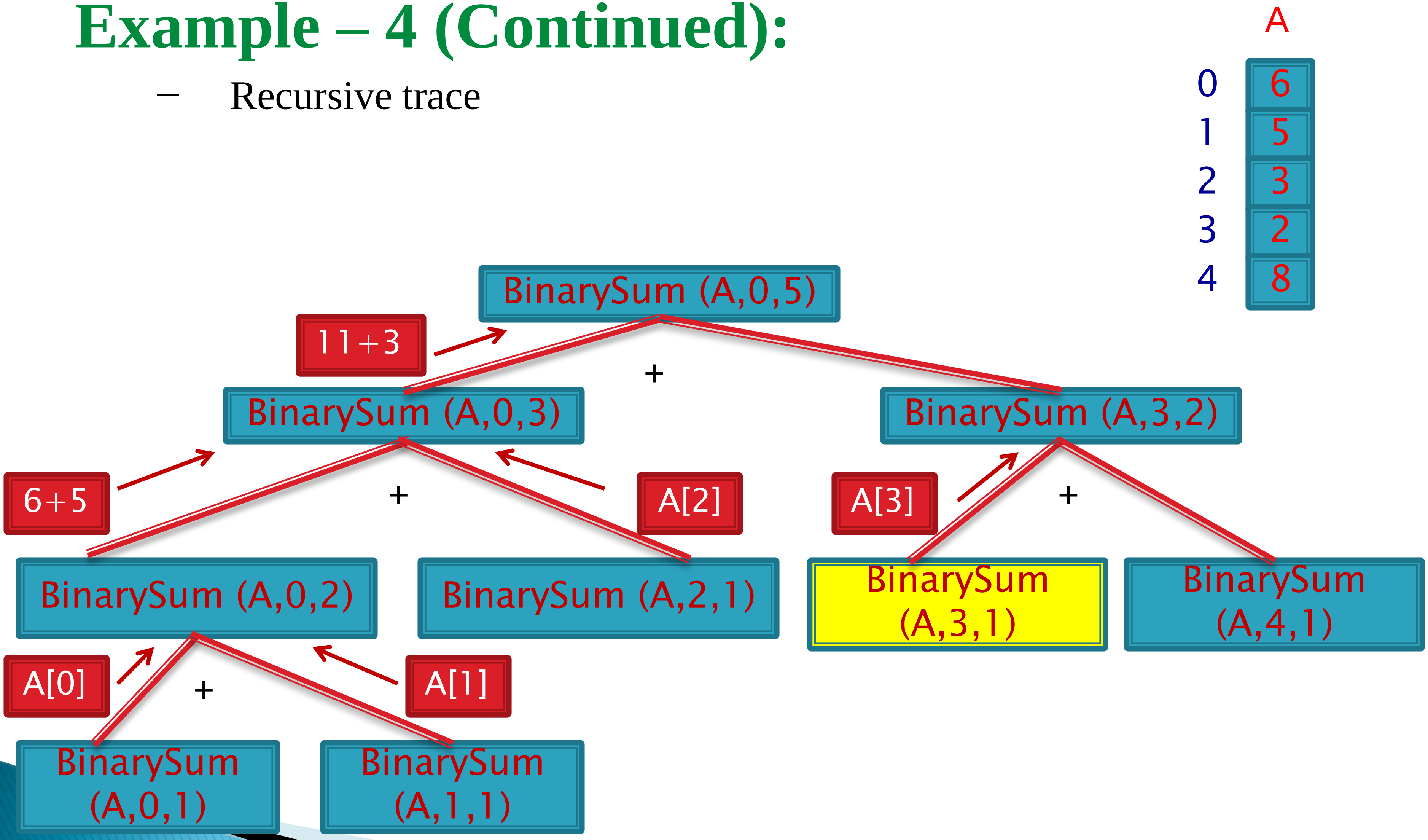
- Recursive trace



Binary Recursion

Example – 4 (Continued):

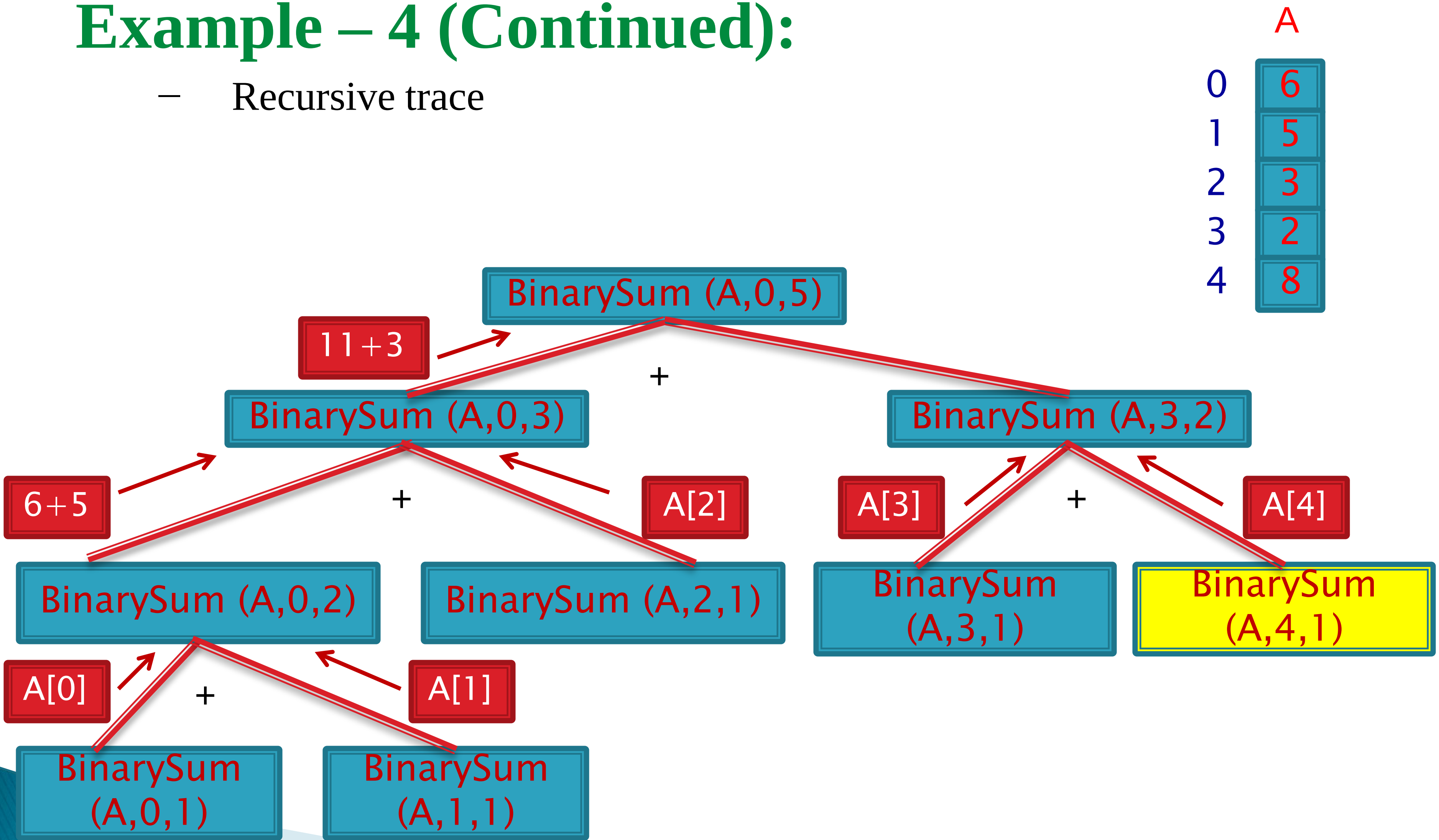
- Recursive trace



Binary Recursion

Example – 4 (Continued):

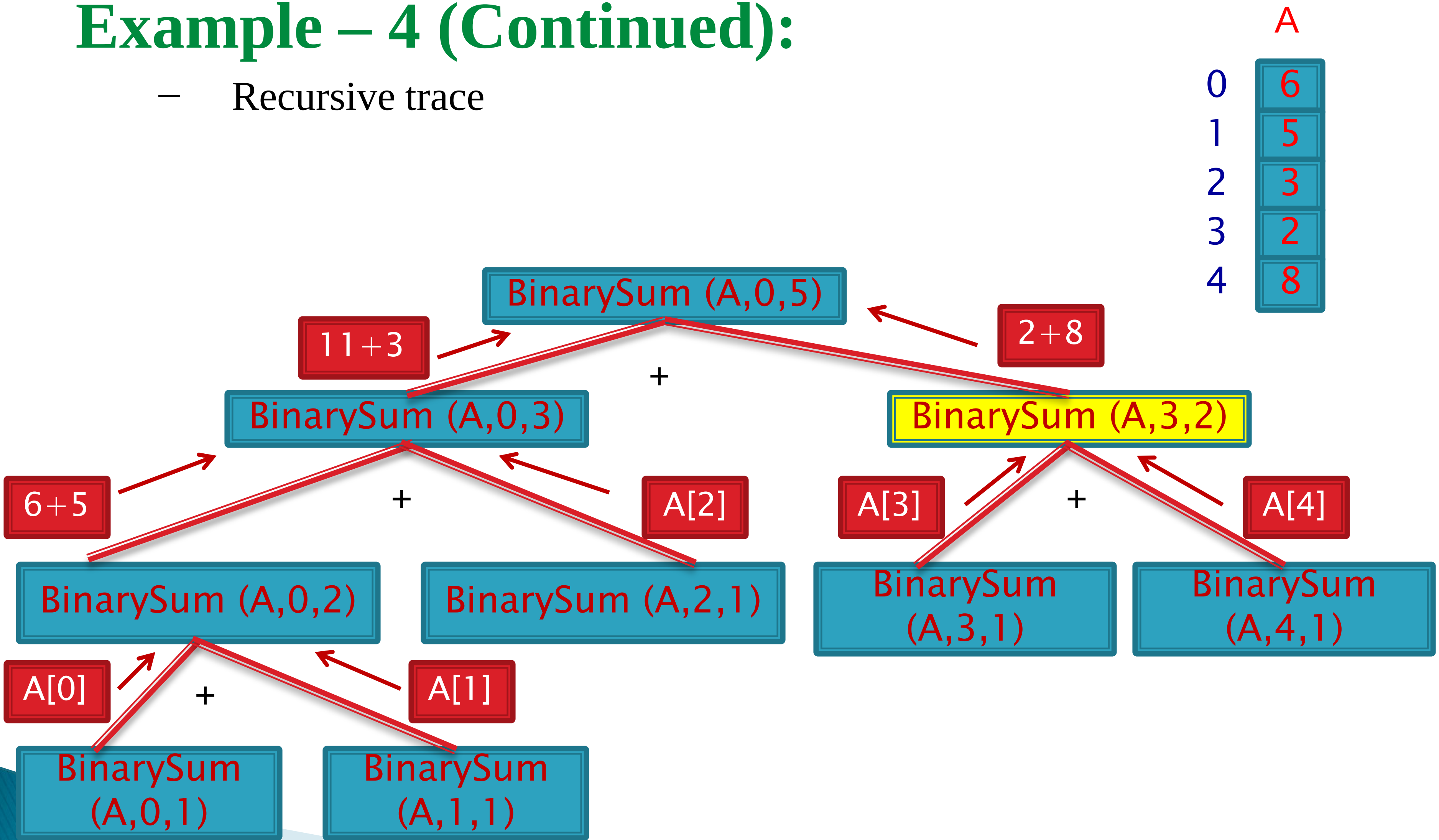
- Recursive trace



Binary Recursion

Example – 4 (Continued):

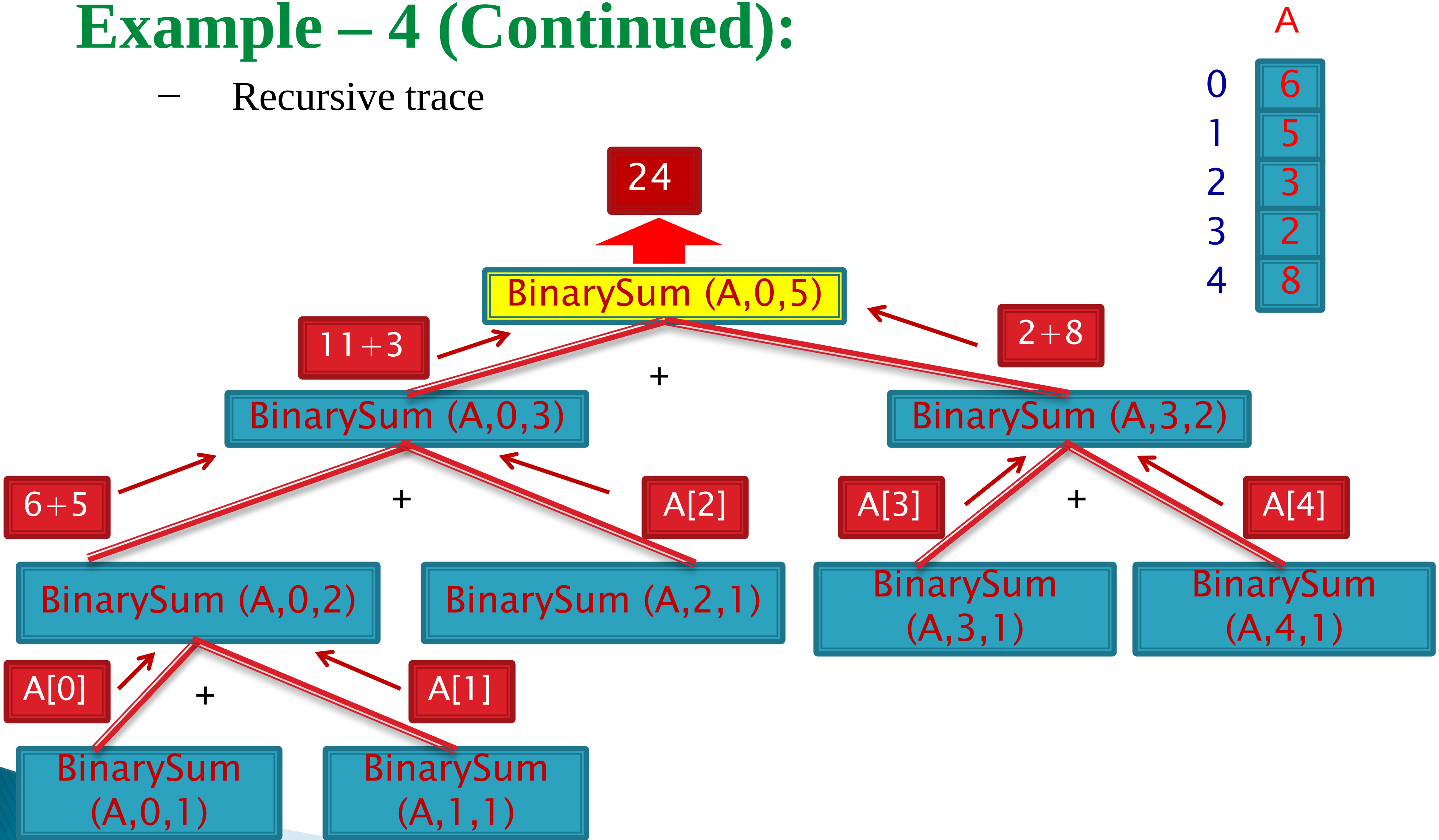
– Recursive trace



Binary Recursion

Example – 4 (Continued):

- Recursive trace



Binary Recursion

Example – 5

- **The Fibonacci Number**
  1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
- Each number after the second number is the sum of the two preceding numbers.
- These numbers can naturally be defined recursively :

$$F(n) = \begin{cases} 1 \text{ if } n = 0 & \leftarrow \text{ Base Case-1} \\ 1 \text{ if } n = 1 & \leftarrow \text{ Base Case-2} \\ F(n-1) + F(n-2) \text{ if } n > 1 & \leftarrow \text{ Recursive Case} \end{cases}$$

Binary Recursion

Example – 5 (Continued)

– Recursive Implementation of Fibonacci Function

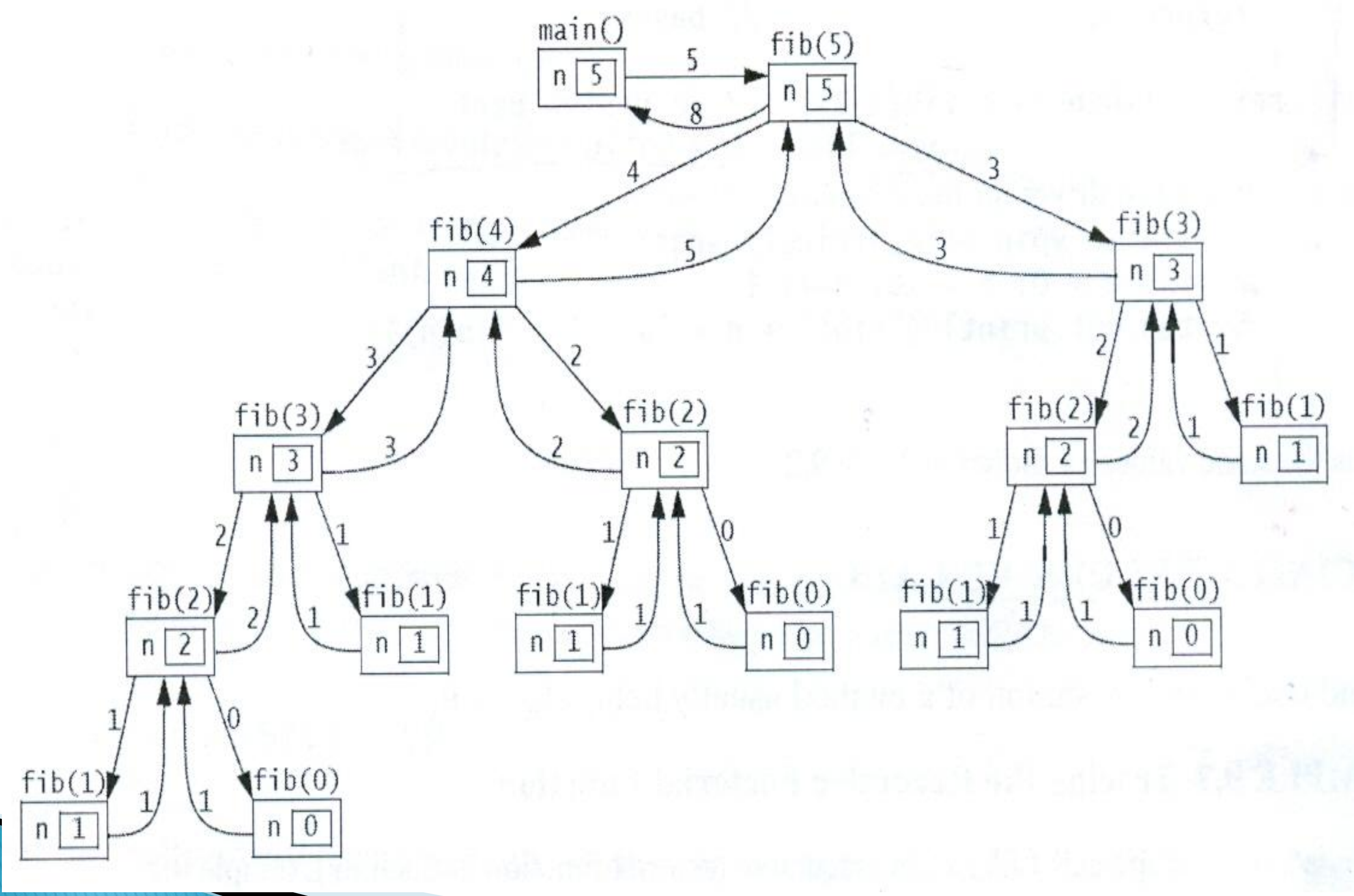
```
public static int fib(int n)
{
        if (n < 2)
                return 1;                       // base cases
        else
                return  fib(n-1)+fib(n-2);   // recursive part

}
```

Binary Recursion

Example – 5 (Continued)

– Recursive Trace of Fibonacci Function: fib(5)



Linear Recursion

Example – 6: Binary Search

- **Problem**: Given $S=\{s_0, s_1, \ldots, s_{n-1}\}$ is a sorted sequence of n integers, and an integer x. Search whether x is in S.
- **Binary Search Algorithm:**
  - If the sequence is empty, return -1.
  - Let s_i be the middle element of the sequence.
    - If $s_i = x$, return its index i.
    - If $s_i < x$, apply the algorithm on the subsequence that lies above s_i.
    - Otherwise, apply the algorithm on the subsequence of S that lies below s_i.

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                     // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

Linear Recursion

Example – 6 (continued): Binary Search

a	2	5	7	11	14	20

– Implementation:

```
public static int search(int a[], int lo, int hi, int x)
{
    if (lo > hi)  return -1; // Basis
    else{                    // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

a					
2	5	7	11	14	20

i=2

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                          // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 14)

Linear Recursion

Example – 6 (continued): Binary Search

a	2	5	7	11	14	20
					i=4	

– Implementation:

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                          // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 14)

search(a, 3, 5, 14)

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

a	2	5	7	11	14	20

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                              // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

a	2	5	7	11	14	20

i=2

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
     else{                    // Recursive part
          int i = (lo+hi/2);
          if(a[i] == x) return i;
          else if(a[i] < x)
                    return search (a, i+1, hi, x)
          else
                    return search (a, lo,i-1, x);
     }
}
```

search(a, 0, 5, 2)

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

a	2	5	7	11	14	20

i=0

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                      // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 2)

search(a, 0, 1, 2)

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

a	2	5	7	11	14	20

i=2

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                        // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 5)

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

a	2	5	7	11	14	20

i=0

```
public static int search(int a[], int lo, int hi, int x)
{
    if (lo > hi)  return -1; // Basis
    else{                      // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
            return search (a, i+1, hi, x)
        else
            return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 5)

search(a, 0, 1, 5)

Linear Recursion

Example – 6 (continued): Binary Search

a	2	5	7	11	14	20

i=0

– Implementation:

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                      // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 5)

search(a, 0, 1, 5)

search(a, 1, 1, 5)

Linear Recursion

Example – 6 (continued): Binary Search

a	2	5	7	11	14	20

– Implementation:

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                          // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

a					
2	5	7	11	14	20

i=2

```
public static int search(int a[], int lo, int hi, int x)
{
    if (lo > hi)  return -1; // Basis
    else{                     // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 21)

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

a	2	5	7	11	14	20

i=4

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                        // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 21)

search(a, 3, 5, 21)

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

a	2	5	7	11	14	20

i=5

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                          // Recursive part
         int i = (lo+hi/2);
         if(a[i] == x) return i;
         else if(a[i] < x)
                   return search (a, i+1, hi, x)
         else
                   return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 21)

search(a, 3, 5, 21)

search(a, 5, 5, 21)

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

a	2	5	7	11	14	20

```
public static int search(int a[], int lo, int hi, int x)
{
    if (lo > hi)  return -1; // Basis
    else{                          // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 21)

search(a, 3, 5, 21)

search(a, 5, 5, 21)

search(a, 6, 5, 21)

Linear Recursion

Example – 6 (continued): Binary Search

a	2	5	7	11	14	20

- Implementation:

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                    // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

Linear Recursion

Example – 6 (continued): Binary Search

a	2	5	7	11	14	20
			i=2			

– Implementation:

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                      // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 12)

Linear Recursion

Example – 6 (continued): Binary Search

a	2	5	7	11	14	20
					i=4	

– Implementation:

```
public static int search(int a[], int lo, int hi, int x)
{
    if (lo > hi)  return -1; // Basis
    else{                    // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 12)

search(a, 3, 5, 12)

Linear Recursion

Example – 6 (continued): Binary Search

– Implementation:

a	2	5	7	11	14	20

i=3

```
public static int search(int a[], int lo, int hi, int x)
{
     if (lo > hi)  return -1; // Basis
    else{                          // Recursive part
        int i = (lo+hi/2);
        if(a[i] == x) return i;
        else if(a[i] < x)
                return search (a, i+1, hi, x)
        else
                return search (a, lo,i-1, x);
    }
}
```

search(a, 0, 5, 12)

search(a, 3, 5, 12)

search(a, 3, 3, 12)

Linear Recursion

Example – 6 (continued): Binary Search

a	2	5	7	11	14	20

– Implementation:

```
public static int search(int a[], int lo, int hi, int x)
{
      if (lo > hi)  return -1; // Basis
     else{                              // Recursive part
           int i = (lo+hi/2);
           if(a[i] == x) return i;
           else if(a[i] < x)
                     return search (a, i+1, hi, x)
           else
                     return search (a, lo,i-1, x);
     }
}
```

search(a, 0, 5, 12)

search(a, 3, 5, 12)

search(a, 3, 3, 12)

search(a, 4, 3, 12)