Recursion: Simple Examples

Paolo Camurati
Dip. Automatica e Informatica
Politecnico di Torino

Maximum of an array

- Specifications
  - Given an array of $n=2^k$ integers
  - Find its maximum and print it on standard output

divide and conquer
$a = 2$ and $n/n' = 2$

Solution

Maximum in an array of integers

- If n=1, find maximum explicitly
- If n>1
  - Divide array in 2 subarrays, each being half the original array
  - Recursively search for maximum in each subarray
  - Compare results and return bigger one

Solution

In the main program (initial call #0):

```
result = max(a, 0, 3);
```

l = 0 r = 3

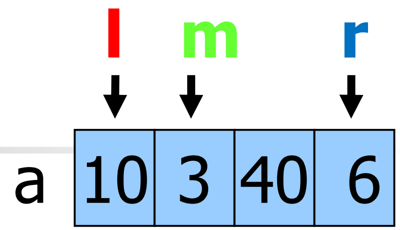
	l			r
a	10	3	40	6

$n = 2^2$

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

```
max(a, 0, 3);
```

Solution



l = 0 r = 3 m = 1

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 0, 1);`

Solution

a | 10 | 3 | 40 | 6

l = 0 **r** = 3 **m** = 1

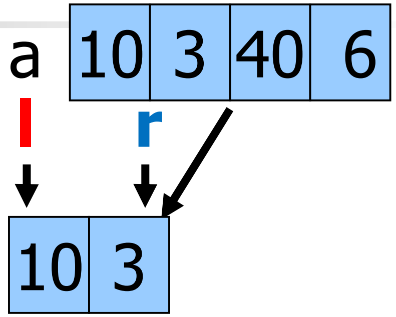
Recursive call #1

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

```
max(a, 0, 1);
```

Solution

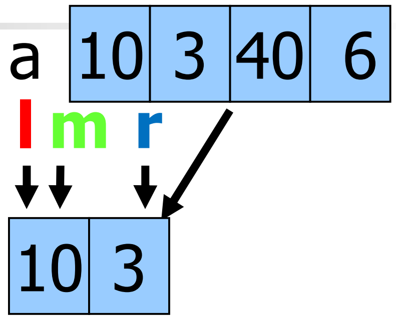


l = 0 r = 1

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

```
max(a, 0, 1);
```

Solution



l = 0 r = 1 m = 0

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 0, 1);`

Solution

a	10	3	40	6

10	3

l = 0 r = 1 m = 0

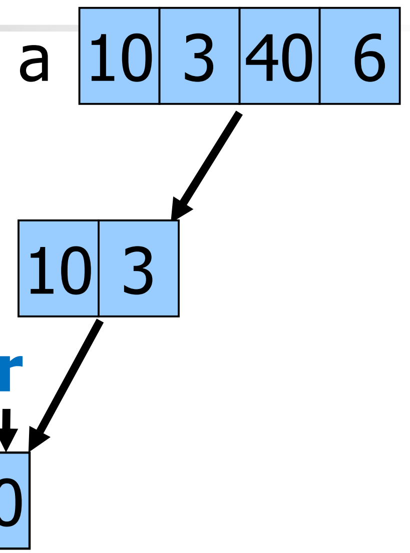
```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

Recursive call #2

`max(a, 0, 0);`

`max(a, 0, 0);`

Solution

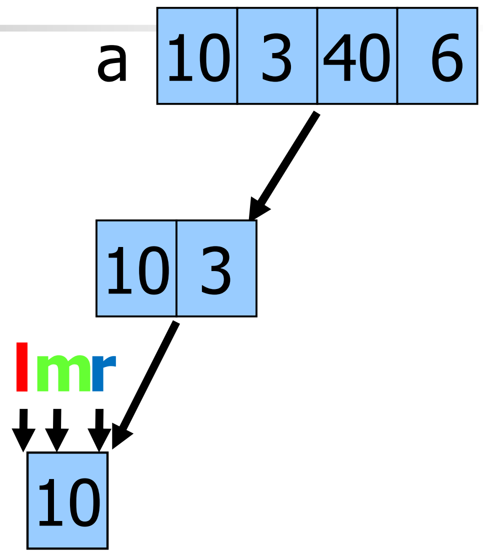


l = 0 r = 0

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

max(a, 0, 0);

Solution



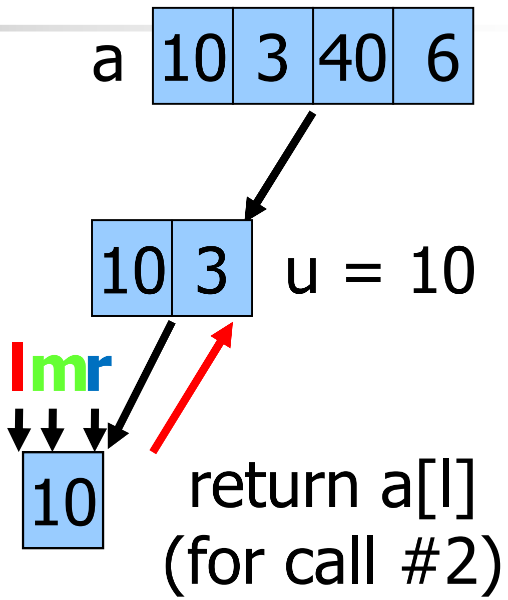
l = 0 r = 0 m = 0

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 0, 0);`

Solution

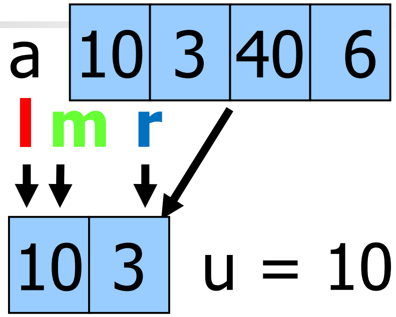
l = 0 r = 0 m = 0




```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

```
max(a, 0, 1);
```

Solution



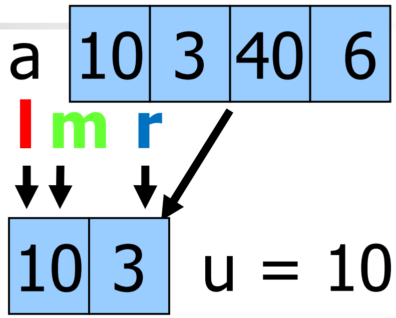
u = 10

l = 0 r = 1 m = 0

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

```
max(a, 0, 1);
```

Solution



l = 0 r = 1 m = 0

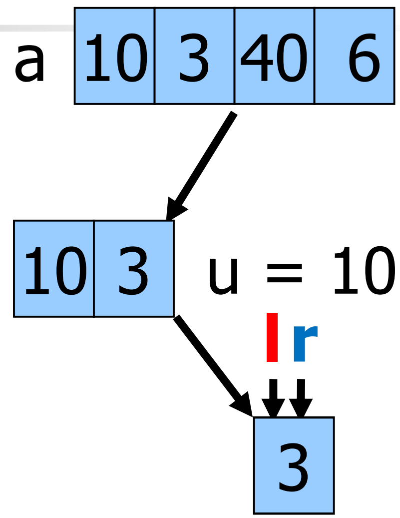
```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

Recursive call #3

```
max(a, 1, 1);
```

`max(a, 1, 1);`

Solution

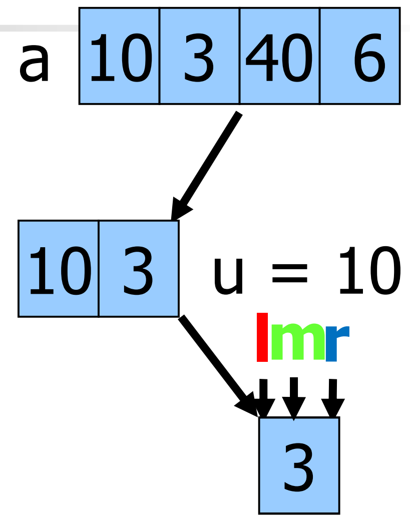


l = 1 r = 1

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 1, 1);`

Solution

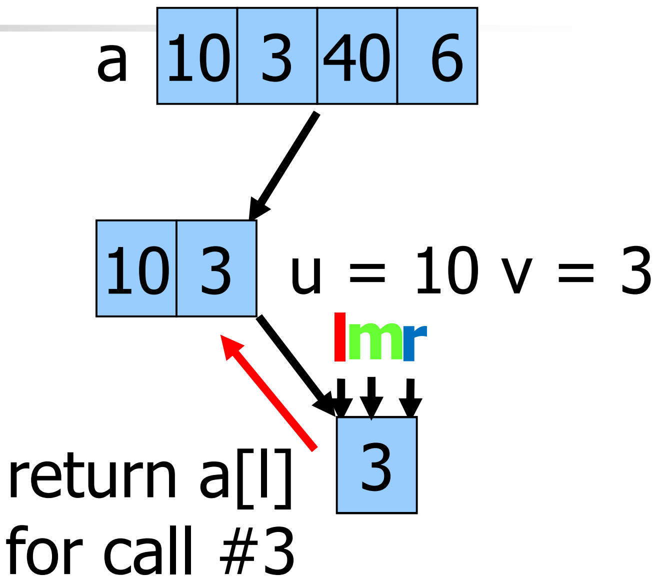


l = 1 r = 1 m = 1

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 1, 1);`

Solution

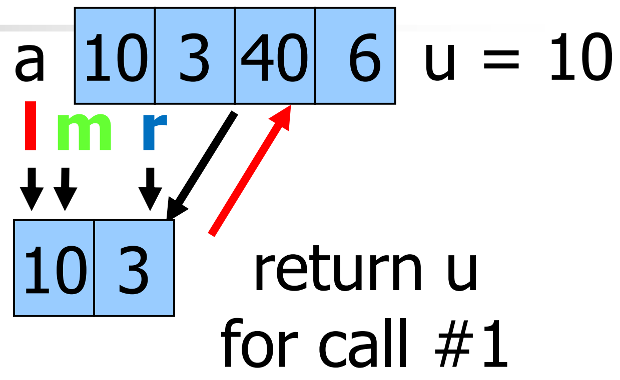


l = 1 r = 1 m = 1

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 0, 1);`

Solution



l = 0 r = 1 m = 0

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

Solution

`max(a, 0, 3);`

	l	m		r	
a	10	3	40	6	u = 10

l = 0 **r** = 3 **m** = 1

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

```
max(a, 0, 3);
```

Solution

a	10	3	40	6
	l	m		r

u = 10

l = 0 r = 3 m = 1

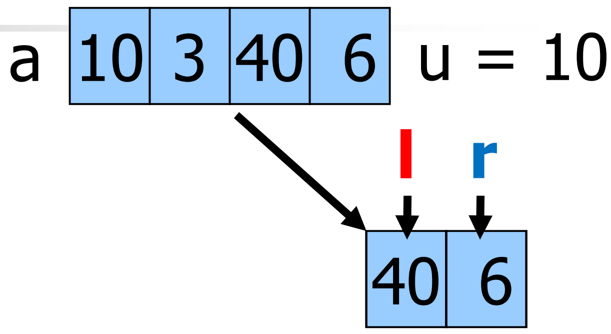
```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

Recursive call #4

```
max(a, 2, 3);
```

`max(a, 2, 3);`

Solution

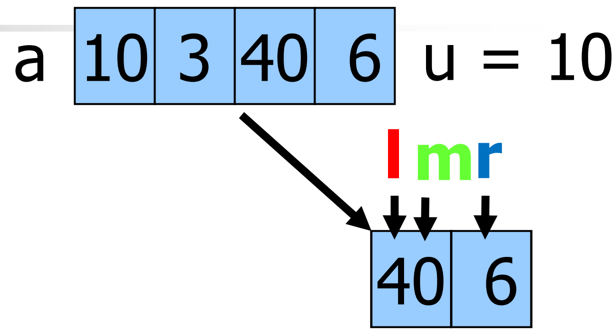


l = 2 r = 3

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 2, 3);`

Solution

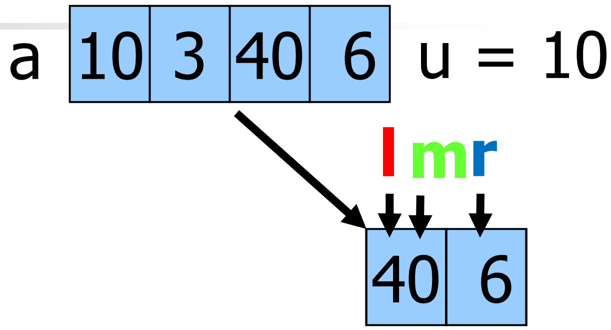


l = 2 **r** = 3 **m** = 2

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 2, 3);`

Solution



l = 2 r = 3 m = 2

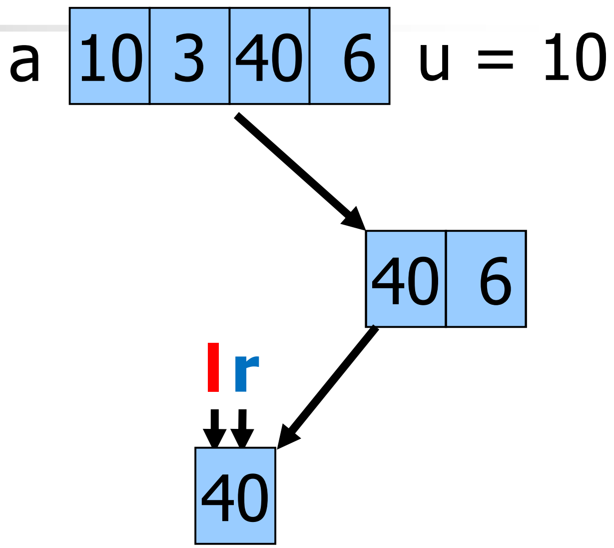
```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

Recursive call #5

`max(a, 2, 2);`

```
max(a, 2, 2);
```

Solution

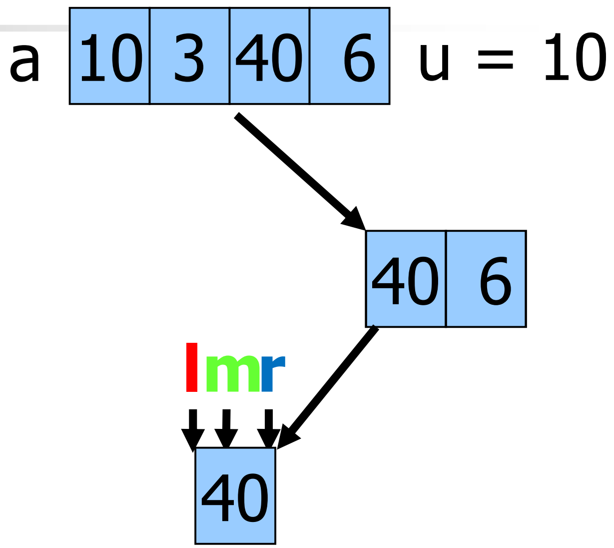


l = 2 r = 2

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

```
max(a, 2, 2);
```

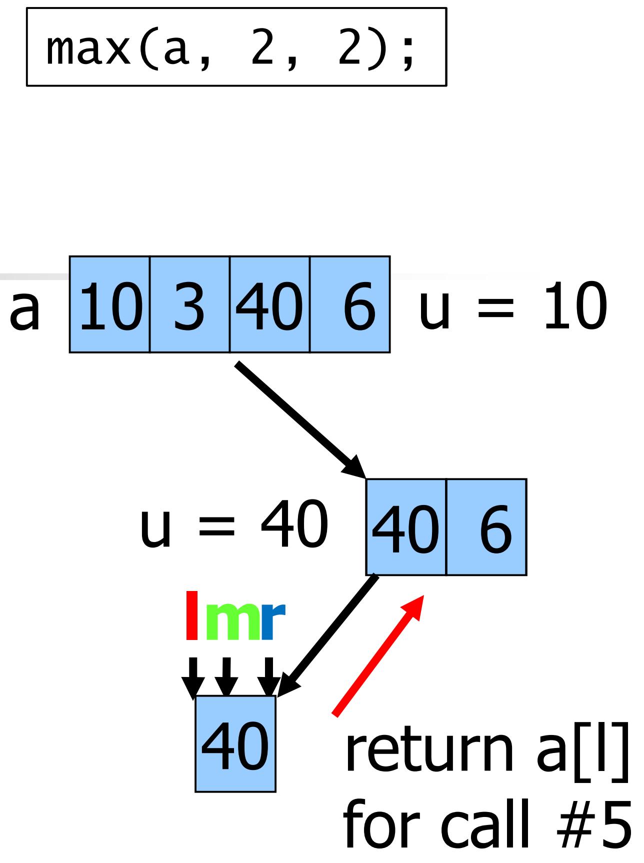
Solution



l = 2 r = 2 m = 2

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

Solution

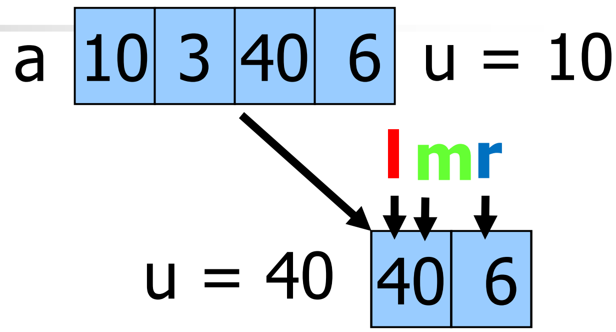


l = 2 r = 2 m = 2

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 2, 3);`

Solution

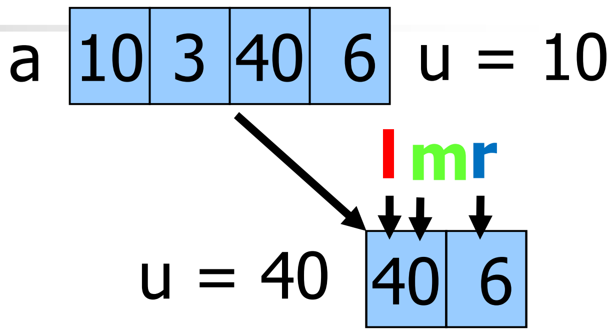


l = 2 r = 3 m = 2

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 2, 3);`

Solution



l = 2 r = 3 m = 2

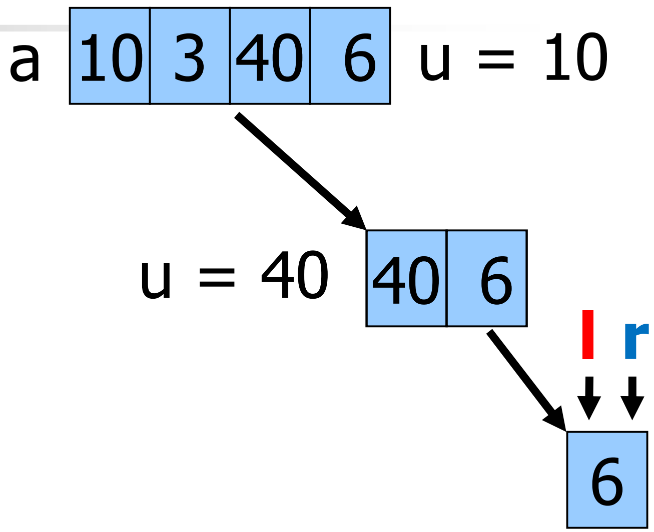
```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

Recursive call #6

`max(a, 3, 3);`

```
max(a, 3, 3);
```

Solution

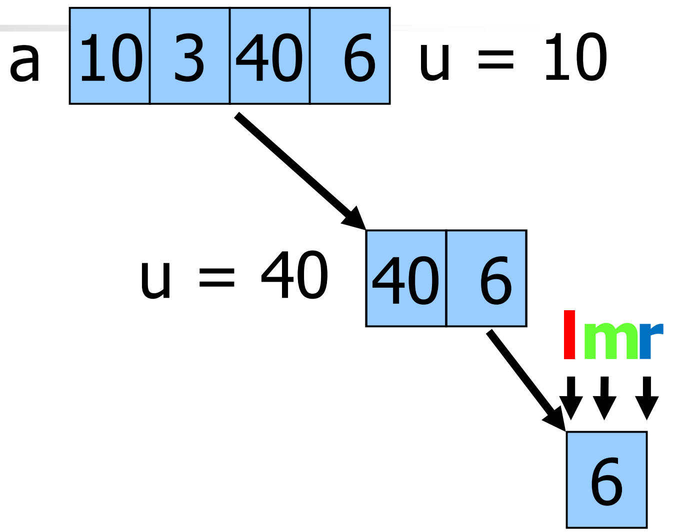


l = 3 **r** = 3

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 3, 3);`

Solution



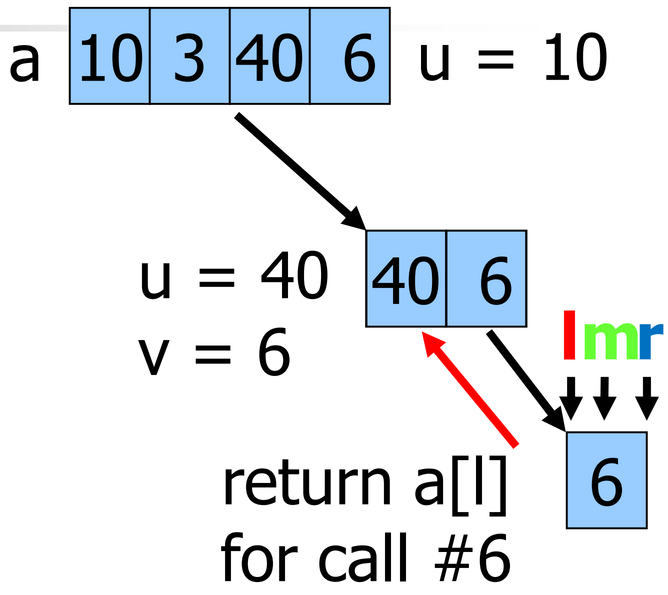
l = 3 r = 3 m = 3

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`max(a, 3, 3);`

Solution

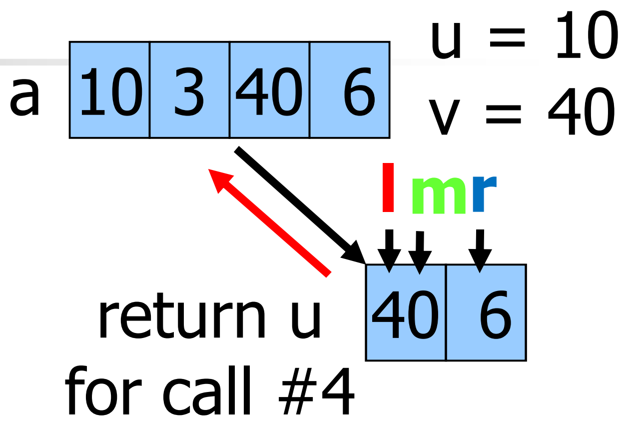
l = 3 r = 3 m = 3




```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

```
max(a, 2, 3);
```

Solution



l = 2 **r** = 3 **m** = 2

u = 40

v = 6

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

`result = max (A, 0, 3);` ⇨ `result = 40`

Solution

	l	m		r
a	10	3	40	6

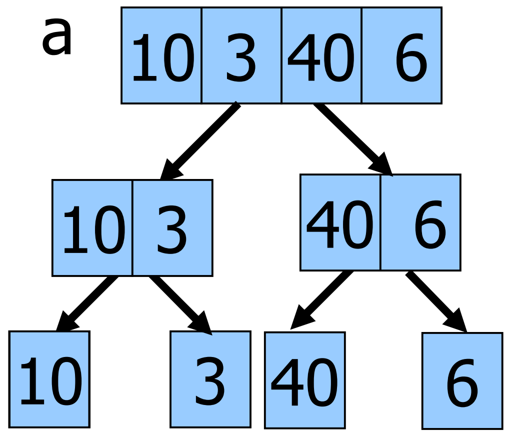
u = 10
v = 40

l = 0 r = 3 m = 1

return v

```
int max(int a[],int l,int r){
  int u, v;
  int m = (l + r)/2;
  if (l == r)
    return a[l];
  u = max (a, l, m);
  v = max (a, m+1, r);
  if (u > v)
    return u;
  else
    return v;
}
```

Solution



Complexity Analysis

divide and conquer
$a = 2$ $b = 2$

- Complexity analysis
  - $D(n) = \Theta(1)$
  - $C(n) = \Theta(1)$
  - $a = 2, b = 2$
- Recurrence equation
  - $T(n) = D(n) + a\,T(n/b) + C(n)$

Divide – Recur - Combine

- That is
  - $T(n) = 2T(n/2) + 1$ $\quad n > 1$
  - $T(1) = 1$ $\quad n=1$

Complexity Analysis

- Resolution by unfolding
  - $T(n) = 1 + 2T(n/2)$
  - $T(n/2) = 1 + 2T(n/4)$
  - $T(n/4) = 1 + 2T(n/8)$

Termination condition
$n/2^i = 1$
$i = \log_2 n$

$\sum_{i=0}^{k} x^i = (x^{k+1} - 1)/(x-1)$

- Replacing in T(n)
  - $T(n) = 1 + 2 + 4 + 2^3 T(n/8)$

$$= \sum_{i=0}^{\log_2 n} 2^i = (2^{\log_2 n + 1} - 1)/(2-1)$$

$$= 2 \cdot 2^{\log_2 n} - 1 = 2n - 1$$

- Thus
  - $T(n) = O(n)$

Factorials

- Factorial (iterative definition)
  - n! ≡ $\prod_{i=0}^{n-1}(n-i)$ = n · (n-1) · …. · 2 · 1
- Factorial (recursive definition)
  - n! ≡ n · (n-1)!   n≥1
  - 0! ≡ 1

Decease and conquer
$a = 1$ $k_i = 1$

Example: 5!

$$5! = 5 \cdot 4! = 120$$

$$4! = 4 \cdot 3! = 24$$

$$3! = 3 \cdot 2! = 6$$

$$2! = 2 \cdot 1! = 2$$

$$1! = 1 \cdot 0! = 1$$

$$0! = 1$$

Solution

```
#include <stdio.h>

long fact(int n);

main() {
  long n;
  printf("Input n:  ");
  scanf("%d", &n);
  printf("%d ! = %d\n", n, fact(n));
}

long fact(long n) {
  if(n == 0)
    return(1);
  return(n * fact(n-1));
}
```

Complexity Analysis

- Complexity analysis
  - $D(n) = \Theta(1)$
  - $C(n) = \Theta(1)$
  - $a = 1$
  - $k_i = 1$
- Recurrence equation
  - $T(n) = D(n) + \sum_{i=0}^{a-1} T(n-ki) + C(n)$
- That is
  - $T(n) = 1 + T(n-1)$ $\quad n > 1$
  - $T(1) = 1$

Complexity Analysis

- Resolution by unfolding
  - T(n) = 1 + T(n-1)
  - T(n-1) = 1 + T(n-2)
  - T(n-2) = 1 + T(n-3)
  - ...
- Replacing in T(n)
  - T(n) = 1+1+1+T(n-3) = $\sum_{i=0}^{n-1} 1 = 1 + (n-1) = n$
- Thus
  - T(n) = O(n)

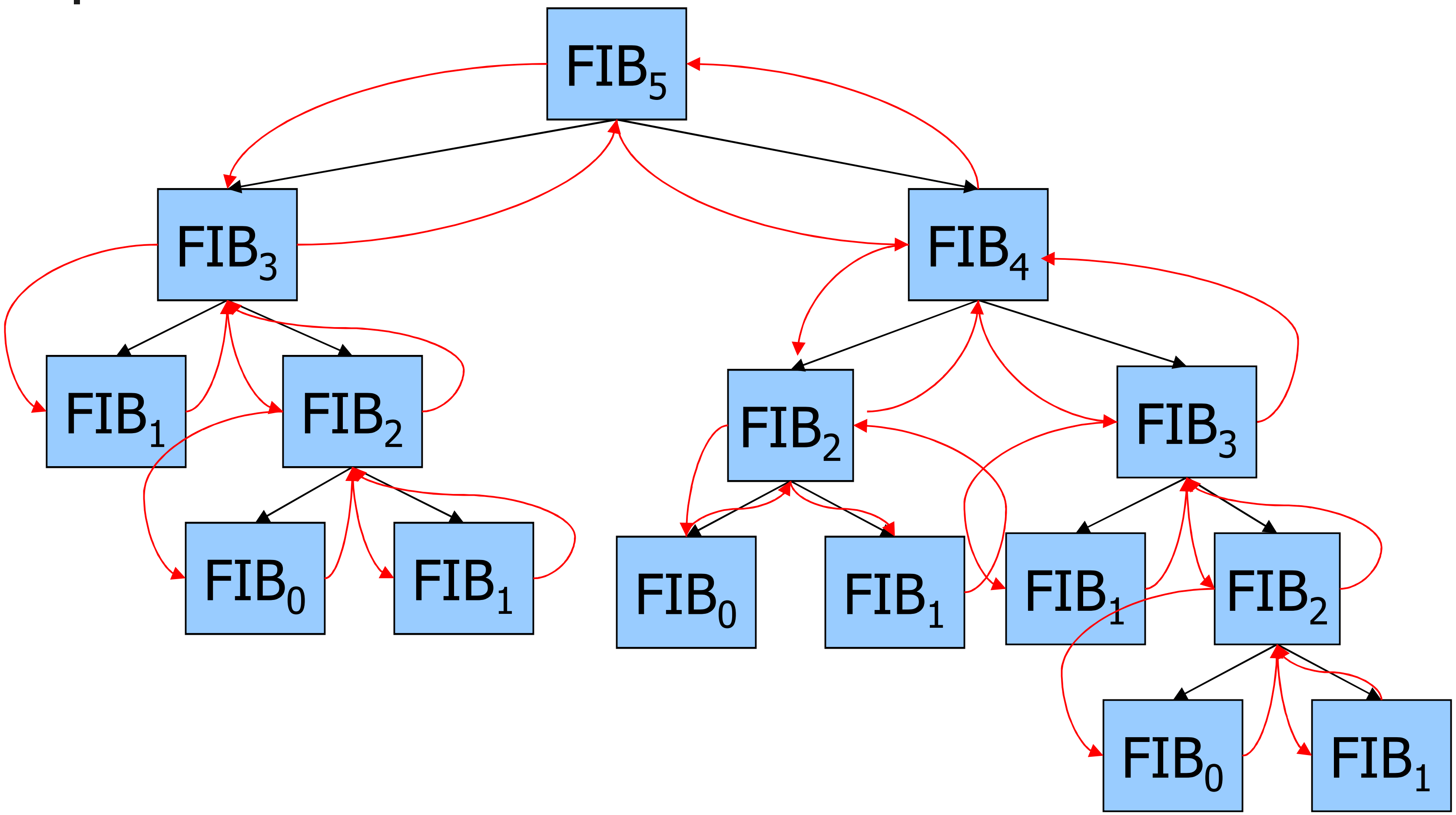
Termination
n-i = 1
i = n -1

Fibonacci Numbers

- Fibonacci numbers
  - $FIB_n = FIB_{n-2} + FIB_{n-1} \qquad n>1$
  - $FIB_0 = 0$
  - $FIB_1 = 1$
- Example
  - 0 1 1 2 3 5 8 13 21 34 …

Decrease and conquer
$a = 2 \ k_i = 1 \ k_{i-1} = 2$

Example: Computing FIB_5



Solution

```
#include <stdio.h>

long fib(long n);

main() {
  long n;
  printf("Input n:  ");
  scanf("%d", &n);
  printf("Fibonacci of %d is: %d \n", n, fib(n));
}

long fib(long n) {
  if(n == 0 || n == 1)
    return (n);
  return (fib(n-2) + fib(n-1));
}
```

Complexity Analysis

- Complexity Analysis
  - $D(n) = \Theta(1)$
  - $C(n) = \Theta(1)$
  - $a = 2$
  - $k_i = 1$
  - $k_{i-1} = 2$
- Recurrence equation
  - $T(n) = D(n) + \sum_{i=0}^{a-1} T(n-ki) + C(n)$

Complexity Analysis

- That is
  - $T(n) = 1 + T(n-1) + T(n-2) \qquad n > 1$
  - $T(0) = 1$
  - $T(1) = 1$
  - …
- Conservative approximation
  - $T(n-2) \leq T(n-1)$
    - Replace T(n-2) with T(n-1)
  - $T(n) = 1 + 2T(n-1) \qquad n > 1$
  - $T(n) = 1$

Complexity Analysis

- Resolution by unfolding
  - $T(n) = 1 + 2T(n-1)$
  - $T(n-1) = 1 + 2T(n-2)$
  - $T(n-2) = 1 + 2T(n-3)$
  - …
- Replacing in T(n)
  - $T(n) = 1 + 2 + 4 + 2^3T(n-3) = \sum_{i=0}^{n-1} 2^i = 2^n - 1$
- Thus
  - $T(n) = O(2^n)$

Termination
$n-i = 1$
$i = n - 1$

$\sum_{i=0}^{k} x^i = (x^{k+1} - 1)/(x-1)$

Binary search

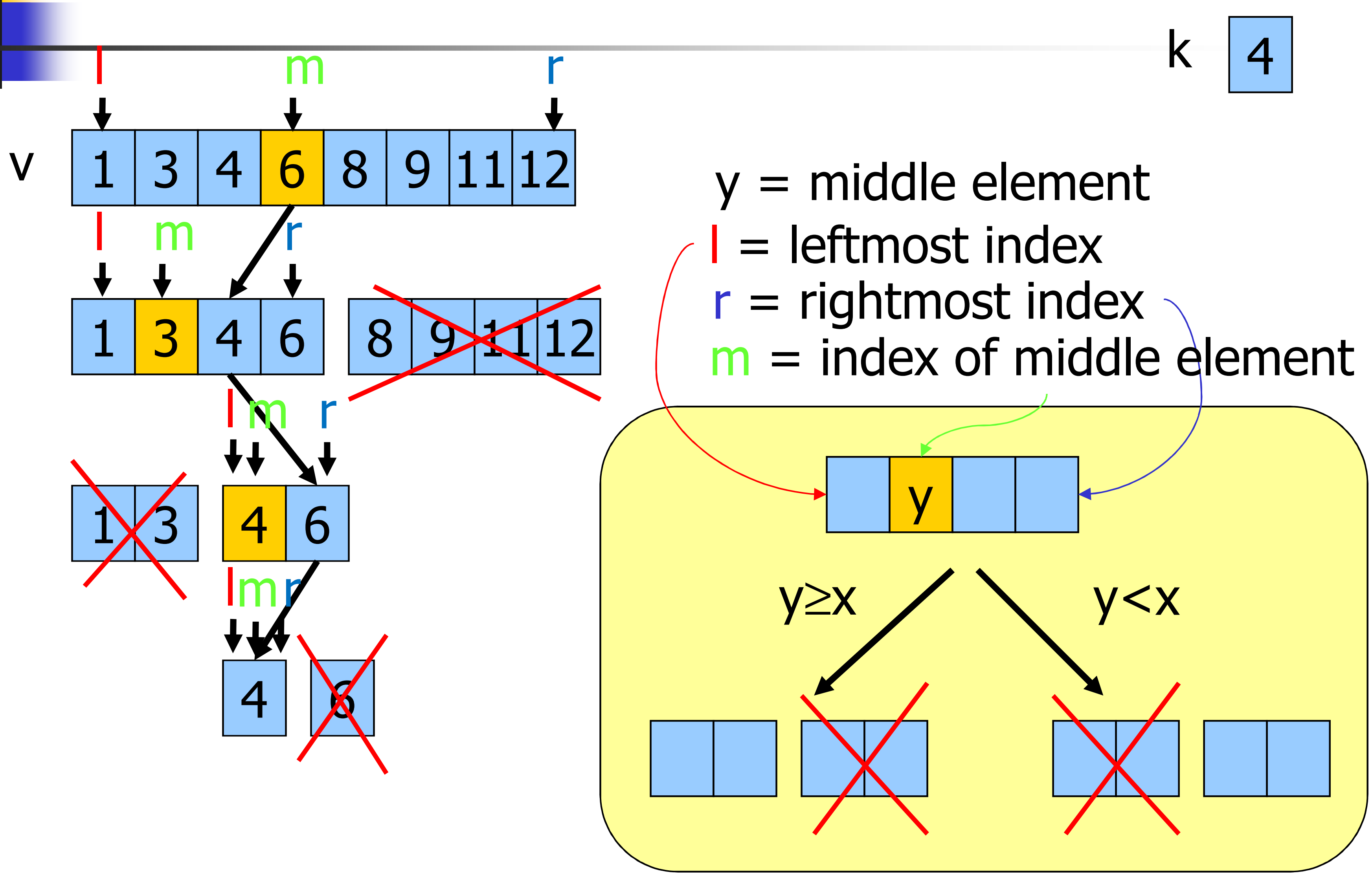
- Binary search
  - Does key k belong to the sorted array v[n]? Yes/No

Assumption: $n = 2^p$

- Approach
  - At each step: compare k with middle element in the array:
    - =: Termination with success
    - <: Search continues in left sub-array
    - >: Search continues in right sub-array

Divide and conquer
$a = 1$ $b = 2$

Example


k
4
l
m
r
v
1
3
4
6
8
9
11
12
y = middle element
l = leftmost index
r = rightmost index
m = index of middle element
y
y≥x
y<x

Solution

```
int BinSearch(int v[], int l, int r, int k){
  int m;

  if (l > r)
    return(-1);

  m = (l+r) / 2;

  if (k < v[m])
    return(BinSearch(v, l, m-1, k));
  if (k > v[m])
    return(BinSearch(v, m+1, r, k));

  return m;
}
```

Complexity Analysis

- Complexity analysis
  - $D(n) = \Theta(1)$
  - $C(n) = \Theta(1)$
  - $a = 1, b = 2$
- Recurrence equation
  - $T(n) = D(n) + a\,T(n/b) + C(n)$
- That is
  - $T(n) = T(n/2) + 1$ $\quad n > 1$
  - $T(1) = 1$ $\quad n=1$

Complexity Analysis

- Resolution by unfolding
  - $T(n/2) = T(n/4) + 1$
  - $T(n/4) = T(n/8) + 1$
  - $T(n/8) = \ldots$
- Replacing in T(n)
  - $T(n) = 1 + 1 + 1 + T(n/8)$

    $= \sum_{i=0}^{log_2 n} 1$

    $= 1 + \log_2 n$
  - $T(n) = O(\log n)$

Termination:
$n/2^i = 1$
$i = \log_2 n$

The ruler

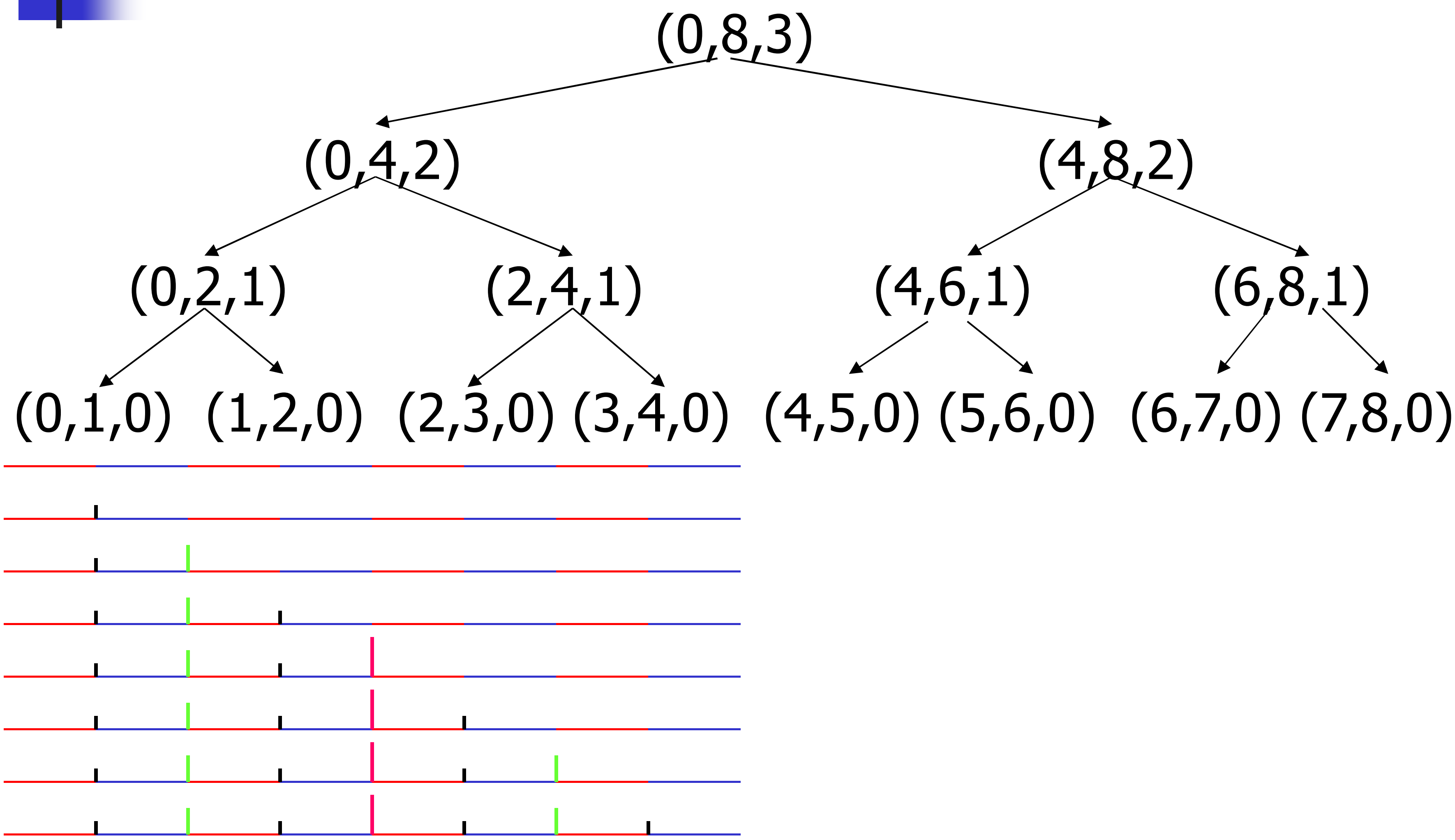
- Draw one mark at each point between 0 and 2^n (excepted boundaries) where
  - The middle mark is n units high
  - The 2 marks in the middle of the 2 left and right halves are n-1 units high
  - etc.
  - Function mark(x, h) draws a mark of height h in position x

Divide and conquer
a = 2 b = 2

Solution

- Algorithm
  - Divide interval in 2
  - Recursively draw (shorter) marks in left half
  - Draw (higher) mark in the middle
  - Recursively draw (shorter) marks in right half
  - Termination condition: marks of height 0

Solution



Solution

```
void ruler(int l, int r, int h) {
  int m;
  m = (l + r)/2;
  if (h > 0) {
    ruler(l, m, h-1);
    mark(m, h);
    ruler(m, r, h-1);
  }
}

void mark(int m, int h) {
  int i;
  printf("%d \t", m);
  for (i = 0; i < h; i++)
    printf("*");
  printf("\n");
}
```

Recursive descent/termination

Division

recursive call

elementary solution

recursive call

Complexity Analysis

- Complexity analysis
  - $D(n) = \Theta(1)$, $C(n) = \Theta(1)$
  - $a = 2$, $b = 2$
- Recurrence equation
  - $T(n) = D(n) + a\,T(n/b) + C(n)$
- That is
  - $T(n) = 2T(n/2) + 1 \qquad n > 1$
  - $T(1) = 1 \qquad n=1$
- $T(n) = O(n)$

Reverse printing

- Read a string from input
- Print it in reverse order (starting from last character and moving back to first one)

Decrease and conquer
$a = 1 \; k_i = 1$

Solution

```
main() {
  char str[max+1];
  printf("Input string: ");
  scanf("%s", str);
  printf("Reverse string is: ");
  reverse_print(str);
}

void reverse_print(char *s) {
  if(*s != '\0') {
    reverse_print(s+1);
    putchar(*s);
  }
  return;
}
```

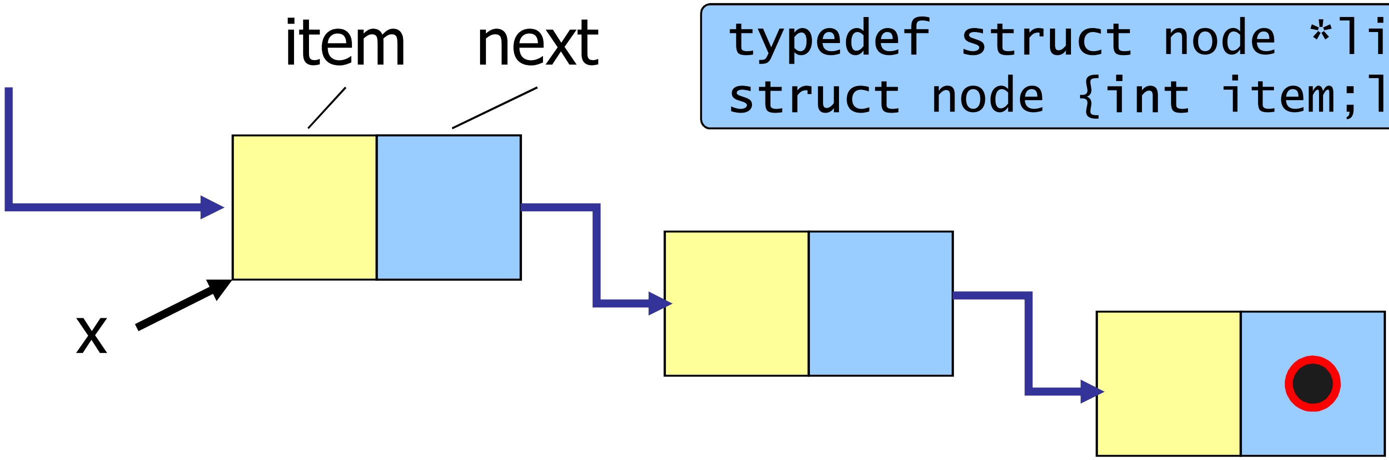
Complexity Analysis

- Complexity analysis
  - D(n) = $\Theta(1)$, C(n) = $\Theta(1)$
  - a = 1, k_i = 1
- Recurrence equation
  - $T(n) = D(n) + \sum_{i=0}^{a-1} T(n-ki) + C(n)$
- That is
  - T(n) = 1 + T(n-1)   n > 1
  - T(1) = 1
- Then
  - T(n) = O(n)

List processing

decrease and conquer
$a = 1\ k_i = 1$

Linked list




```
typedef struct node *link;
struct node {int item;link next;};
```

- Recursive list processing
  - Count the number of elements in a list
  - Traverse a list in order
  - Traverse a list in reverse order
  - Delete an element from the list

Solution

```
int count (link x) {
  if (x == NULL)
    return 0;
  return 1 + count(x->next);
}
void traverse (link h) {
  if (h == NULL)
    return;
  printf("%d", h->item);
  traverse(h->next);
}
void traverseR (link h) {
  if (h == NULL)
    return;
  traverseR(h->next);
  printf("%d", h->item);
}
```

Solution

```
link delete(link x, Item v) {
  if (x == NULL)
    return NULL;
  if ( x->item == v) {
    link t = x->next;
    free(x);
    return t;
  }
  x->next = delete(x->next, v);
  return x;
}
```

Complexity Analysis

- Complexity analysis
  - $D(n) = \Theta(1)$, $C(n) = \Theta(1)$
  - $a = 1$, $k_i = 1$
- Recurrence equation
  - $T(n) = D(n) + \sum_{i=0}^{a-1} T(n-ki) + C(n)$
- That is
  - $T(n) = 1 + T(n-1)$ $\qquad n > 1$
  - $T(1) = 1$
- Then
  - $T(n) = O(n)$

Greatest Common Divisor

- The greatest common divisor *gcd* of 2 non 0 integers *x* and *y* is the greatest among the common divisors of *x* and *y*
- Inefficient algorithm based on decomposition in prime factors
  - x = $p_1^{e_1} \cdot p_2^{e_2} \cdot \cdot \cdot p_r^{e_r}$
  - y = $p_1^{f_1} \cdot p_2^{f_2} \cdot \cdot \cdot p_r^{f_r}$
  - gcd(x,y) = $p_1^{\min(e_1, f_1)} \cdot p_2^{min(e_2, f_2)} \cdot \cdot \cdot p_r^{\min(e_r, fr)}$

Decrease and conquer
a = 1 variable k_i

Common factors with the minimum exponent

Euclid’s Algorithm: Version 1

- Version 1 is based on subtraction

```
if x > y
    gcd(x, y) = gcd(x-y, y)
else
    gcd(x, y) = gcd(x, y-x)
```

- Termination

```
if x=y
    return x
```

Examples

- gcd (20, 8) =
  = gcd (20-8, 8) = gcd (12, 8)
  = gcd (12-8, 8) = gcd (4, 8)
  = gcd (4, 8-4) = gcd (4, 4)
  = 4 → return 4
- gcd (600, 54) =
  = gcd (600-54, 54) = gcd (546, 54)
  = gcd (546-54, 54) = gcd (492, 54) …
  = gcd (6,54) = gcd (6, 54-6) …
  = gcd (6, 12) = gcd (6,6)
  = 6→ return 6

Solution 1

```
#include <stdio.h>
int gcd(int x, int y);

main() {
  int x, y;
  printf("Input x and y:  ");
  scanf("%d%d", &x, &y);
  printf("gcd of %d and %d: %d \n", x, y, gcd(x, y));
}

int gcd(int x, int y) {
  if (x == y)
    return (x);
  if (x > y)
    return gcd (x-y, y);
  else
    return gcd (x, y-x);
}
```

Euclid’s Algorithm: Version 2

- Version 2 is based on the remainder of integer divisions

```
if y > x
  swap (x, y)    // that is; tmp=x; x=y; y=tmp;
gcd(x, y) = gcd(y, x%y)
```

- Termination

```
if y = 0
    return x
```

Examples

- gcd (20, 8) =
  = gcd (8, 20%8) = gcd (8, 4)
  = gcd (4, 8%4) = gcd (4, 0)
  = 4 → return 4

- gcd (600, 54) =
  = gcd (54, 600%54) = gcd (54, 6)
  = gcd (6, 54%6) = gcd (6, 0)
  = 6 → return 6

Examples

- gcd (314159, 271828)=
  = gcd (271828, 314159%271828) = gcd (271828,42331)
  = gcd (42331, 271828%42331)= gcd(42331,17842)
  = gcd (17842, 42331%17842) = gcd (17842, 6647)
  = gcd (6647, 17842%6647) = gcd (6647, 4548)
  = gcd (4548, 6647%4548) = gcd (4548, 2099)
  = gcd (2099, 4548%2099) = gcd (2099, 350)
  = gcd (350, 2099%350) = gcd (350, 349)
  = gcd (349, 350%349), gcd (349, 1)
  = gcd (1,349%1) = gcd (1, 0)
  = 1 → return 1
- 314159 are 271828 mutually prime

Solution 2

```
#include <stdio.h>
int gcd(int m, int n);
main() {
   int m, n, r;
   printf("Input m and n:  ");
   scanf("%d%d", &m, &n);
   if (m>n)
     r = gcd(m, n);
   else
     r = gcd(n, m);
   printf("gcd of (%d, %d) = %d\n", m, n, r);
}
int gcd(int m, int n) {
   if(n == 0)
     return(m);
   return gcd(n, m % n);
}
```

Complexity Analysis

- Complexity analysis
  - $D(x,y) = \Theta(1)$
  - $C(x,y) = \Theta(1)$
  - $a = 1$
  - Variable reduction
- Worst case
  - x and y are 2 consecutive Fibonacci numbers
  - $x = FIB(n+1)$
  - $y = FIB(n)$

Termination:
n steps

Complexity Analysis

- Recurrence equation
  - T(x,y) = T(FIB(n+1), FIB(n))
    = 1 + T(FIB(n),FIB(n+1)%FIB(n))
  - T(x,0) = 1
- But by construction
  - FIB(n+1)%FIB(n) = FIB(n-1)

Termination:
n steps

Complexity Analysis

- $T(x,y) = T(FIB(n+1), FIB(n))$
  $= 1 + T(FIB(n), FIB(n+1)\%FIB(n))$
  $= \sum_{i=0}^{n-1} 1 = n$

- $T(x, y) = O(n)$
  - But, as $y = FIB(n) = (\varphi^n - \varphi'^n) / \sqrt{5} = \Theta(\varphi^n)$
  - Then n is a function of $\log_\varphi(y)$
- Thus
  - $T(n) = O(\log(y))$

Determinant of a (n·n) matrix

- Laplace Algorithm with unfolding on row I
  - Square matrix M (n·n) with indices from 1 to n
- Computation
  - $\det(M) = \Sigma_{1 \leq j \leq n} (-1)^{i+j} M[i][j] \cdot \det(M_{minor\ i,\ j})$
  - Where $M_{minor\ i,\ j}$ is obtained from M eliminating row i and column j

Divide and conquer
$a = n \quad k_i = 2n\text{-}1$

Example

$$M = \begin{bmatrix} -2 & 2 & -3 \\ -1 & 1 & 3 \\ 2 & 0 & -1 \end{bmatrix}$$

$$\begin{aligned} \det(M) = \quad & (-1)^{1+1} \cdot (-2) \cdot \det(M_{\text{minor } 1,\, 1}) + \\ & (-1)^{1+2} \cdot (2) \cdot \det(M_{\text{minor } 1,\, 2}) + \\ & (-1)^{1+3} \cdot (-3) \cdot \det(M_{\text{minor } 1,\, 3}) \end{aligned}$$

Example

$$M_{\text{minor } 1,\, 1} = \begin{bmatrix} -2 & 2 & -3 \\ -1 & 1 & 3 \\ 2 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 0 & -1 \end{bmatrix}$$

$$M_{\text{minor } 1,\, 2} = \begin{bmatrix} -2 & 2 & -3 \\ -1 & 1 & 3 \\ 2 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 3 \\ 2 & -1 \end{bmatrix}$$

$$M_{\text{minor } 1,\, 3} = \begin{bmatrix} -2 & 2 & -3 \\ -1 & 1 & 3 \\ 2 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 2 & 0 \end{bmatrix}$$

Example

- Elementary case
  - Square matrix M 2x2
  - det(M) = M[1][1] · M[2][2] - M[1][2] · M[2][1]

$$\det\left(\begin{bmatrix} 1 & 3 \\ 0 & -1 \end{bmatrix}\right) = -1 - 0 = -1$$

$$\det\left(\begin{bmatrix} -1 & 3 \\ 2 & -1 \end{bmatrix}\right) = 1 - 6 = -5$$

$$\det\left(\begin{bmatrix} -1 & 1 \\ 2 & 0 \end{bmatrix}\right) = 0 - 2 = -2$$

Example

- Then

$$M = \begin{bmatrix} -2 & 2 & -3 \\ -1 & 1 & 3 \\ 2 & 0 & -1 \end{bmatrix}$$

  - $\det(M) = (-1)^{1+1} \cdot (-2) \cdot \det(M_{\text{minor } 1,\, 1}) + (-1)^{1+2} \cdot (2) \cdot \det(M_{\text{minor } 1,\, 2}) + (-1)^{1+3} \cdot (-3) \cdot \det(M_{\text{minor } 1,\, 3})$

    $= (1)\cdot(-2)\cdot(-1) + (-1)\cdot(2)\cdot(-5) + (1)\cdot(-3)\cdot(-2)$

    $= 18$

Solution

- Recursive algorithm (indices ranging between 0 and n-1)
- If n = 2, compute
  - M[0][0] · M[1][1] - M[0][1] · M[1][0]
- If n>2
  - With row=0 and column ranging from 0 and n-1
  - Store in tmp the value of $M_{minor\ 0,\ j}$
  - Recursively compute det($M_{minor\ i,\ j}$)
  - Store result results in
    - sum = sum + M[0][k] * pow (-1,k)*det (tmp, n-1)

Solution

```
int det2x2(int m[][MAX]) {
  return(m[0][0]*m[1][1] - m[0][1]*m[1][0]);
}

void minor(int m[][MAX],int i,int j,int n,int m2[][MAX]){
  int r, c, rr, cc;

  for (rr = 0, r = 0; r < n; r++)
    if (r != i) {
      for (cc = 0, c = 0; c < n; c++) {
        if (c != j) {
           m2[rr][cc] = m[r][c];
           cc++;
        }
        rr++;
      }
  }
}
```

Solution

```
int det (int a[][MAX], int n) {
  int sum, k, i, j, r, c;
  int tmp[MAX][MAX];
  sum = 0;

  if (n == 2)
    return (det2x2(a));

  for (k = 0; k < n; k++) {
    minor (a, 0, k, n, tmp);
    sum = sum + a[0][k] * pow(-1,k) * det (tmp,n-1);
  }

  return (sum);
}
```

Complexity Analysis

- Demonstration beyond the scope of this course
- $T(n) = O(N!)$

Hanoi Towers (E. Lucas 1883)

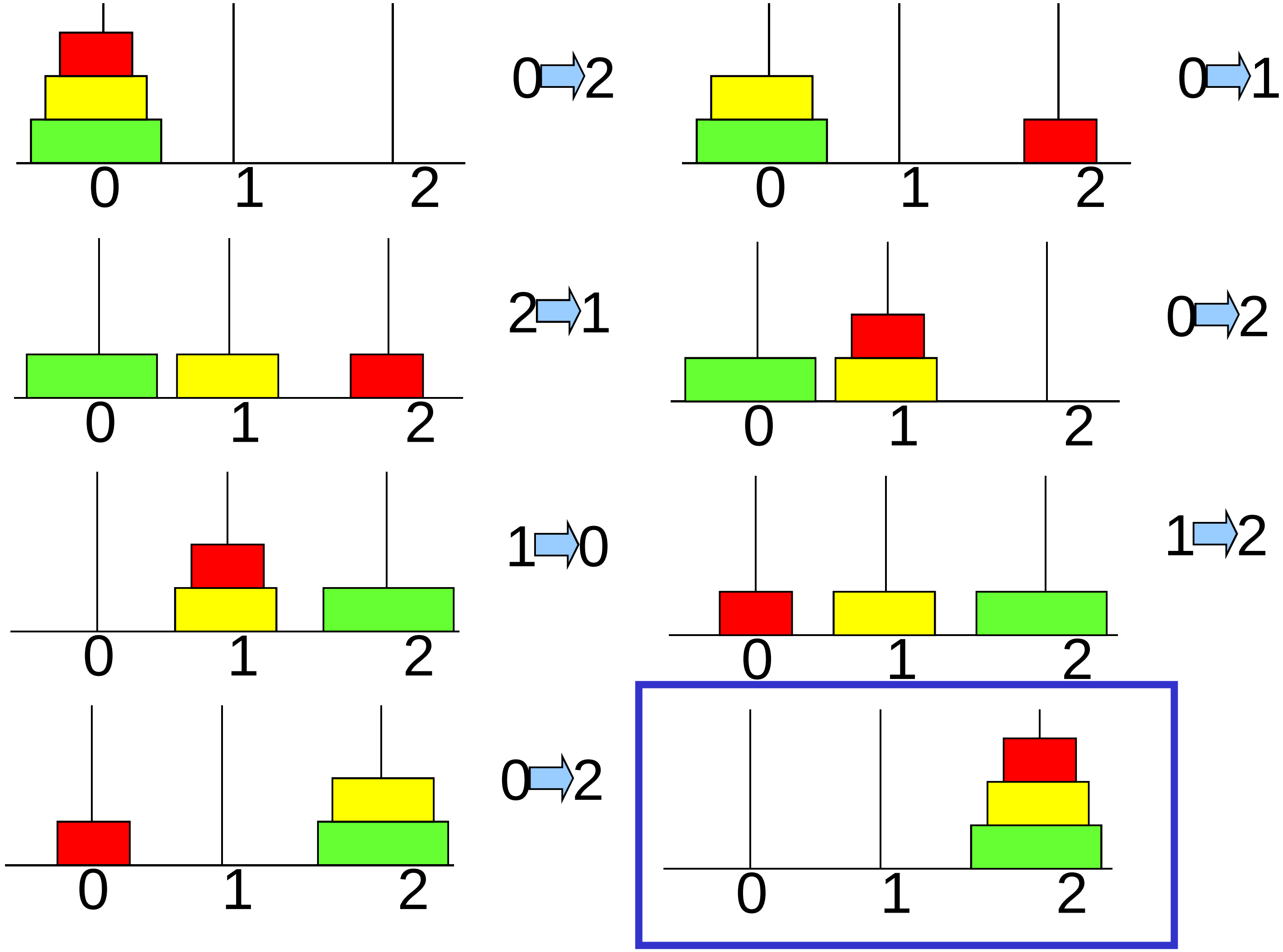
- Initial configuration
  - 3 pegs, 3 disks
  - Disks of decreasing size on first peg
- Final configuration
  - 3 disks on third peg

Decrease and conquer
$a = 2$ $k_i = 1$

Hanoi Towers (E. Lucas 1883)

- Rules
  - Access only to the top disk
  - On each disk overalp only smaller disks
- Generalization
  - Work with n disks and k pegs

Example of solution



Divide and conquer strategy

- Initial problem
  - Move n disks from 0 to 2
- Reduction to subproblems
  - Move n-1 disks from 0 to 1, 2 temporary storage
  - Move last disk from 0 to 2
  - Move n-1 disks from 1 to 2, 0 temporary storage
- Termination condition
  - Move just 1 disk

Solution

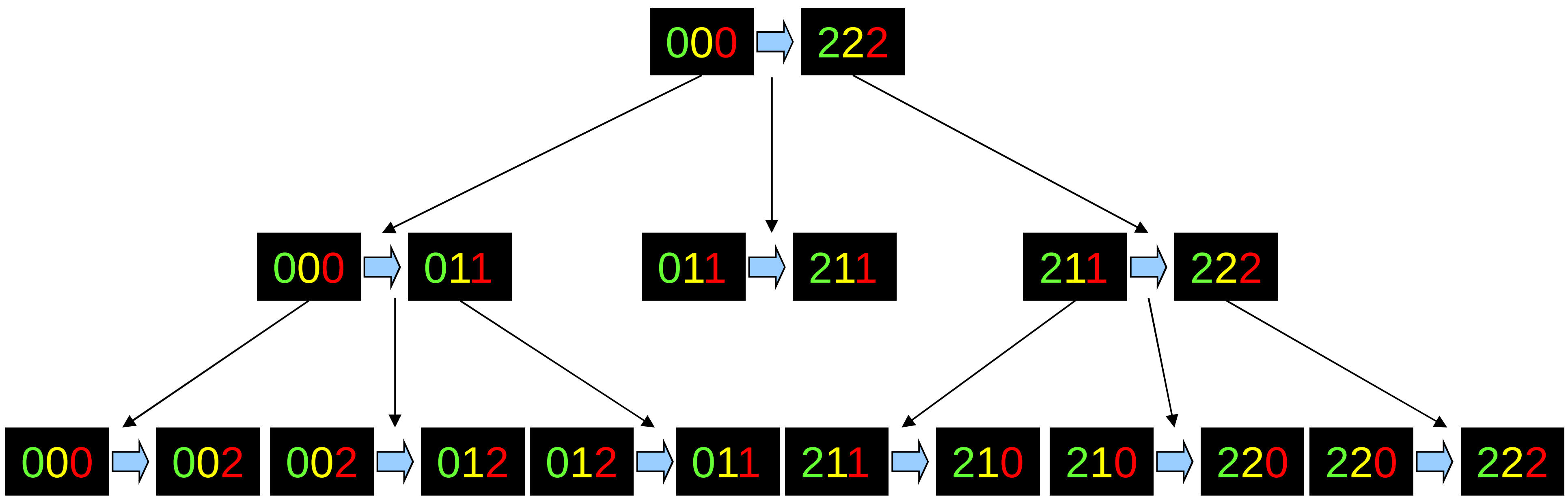
- 0, 1, 2: pegs 0, 1, 2
- large disk
- medium disk
- small disk
- 0 means small disk on peg 0, 2 means large disk on peg 2, etc.
- state 011
- state transition

Solution

Problem 000 ⇨ 222 decomposed into 3 subproblems

1. medium and small disks from 0 to 1 000 ⇨ 011
2. large disk from 0 to 2 011 ⇨ 211
3. medium and small disks from 1 to 2 211 ⇨ 222

Recursion tree



Solution

```
void Hanoi(int n, int src, int dest) {
  int aux;

  aux = 3 - (src + dest);

  if (n == 1) {
    printf("src %d -> dest %d \n", src, dest);
    return;
  }

  Hanoi(n-1, src, aux);
  printf("src %d -> dest %d \n", src, dest);
  Hanoi(n-1, aux, dest);
}
```

termination

division

recursive call

elementary solution

division

recursive call

Complexity Analysis

- Divide: consider n-1 disks
  - $D(n)=\Theta(1)$
- Solve: solve 2 subproblems whose size is n-1 each
  - $2T(n-1)$
- Termination: move 1 disk
  - $\Theta(1)$
- Combine: no action
  - $C(n) = \Theta(1)$

Complexity Analysis

- Recurrence equation
  - $T(n) = D(n) + \sum_{i=0}^{a-1} T(n-ki) + C(n)$
- That is
  - $T(n) = 2T(n\text{-}1) + 1$ $\qquad$ $n>1$
  - $T(1) = 1$ $\qquad$ $n=1$
- Solution
  - $T(n) = O(2^n)$

Product of 2 integers

- Multiplication of 2 integers x and y on n digits
  - n = 2k
- If size is 1, compute x * y (elementary problem)
- If N>1
  - Divide x in 2: x = 10n/2 *xl + xr
  - Divide y in 2: y = 10n/2 *yl + yr
  - Recursively compute xl*yl, xl*yr, xr*yl, xr*yr,
  - Compute
    - x * y = 10n *xl *yl + 10n/2 *(xl*yr + xr*yl) + xr*yr

Divide and conquer
a = 4 b = 2

Solution

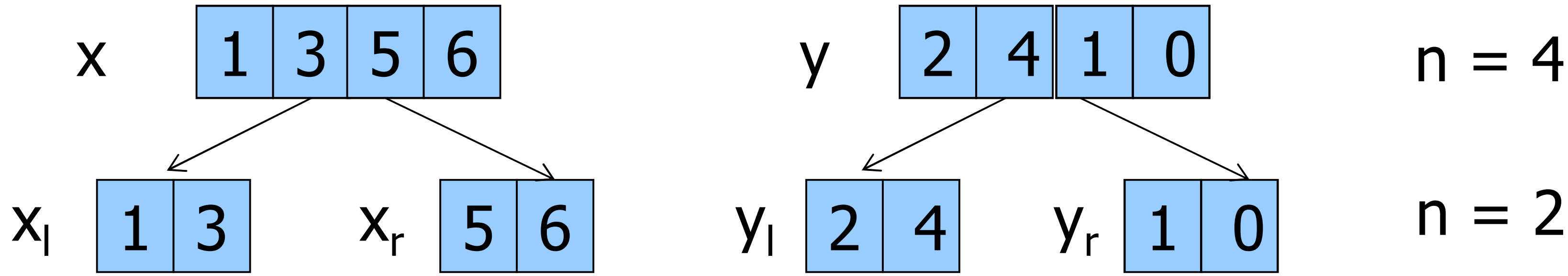
1356 * 2410 = 3.267.960

x | 1 | 3 | 5 | 6 | * y | 2 | 4 | 1 | 0 | n = 4

x	1	3	5	6	*	y	2	4	1	0	n = 4

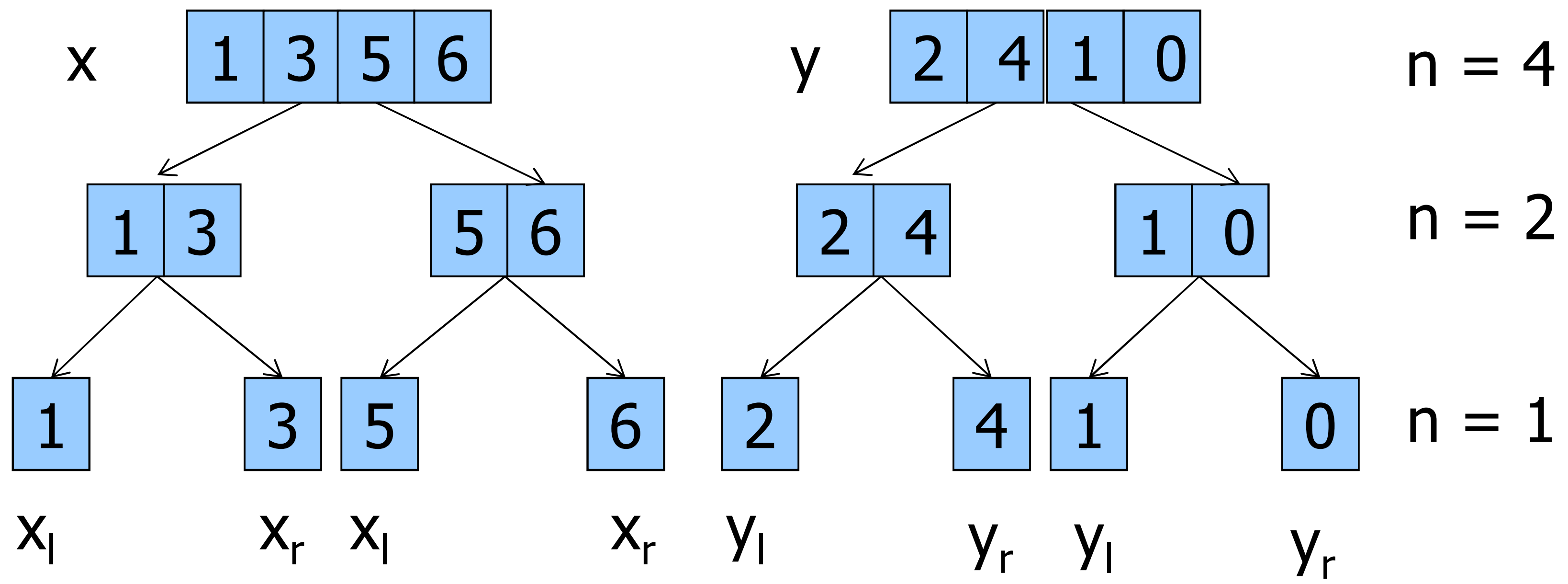
Solution

$$1356 * 2410 = 3.267.960$$



Solution

$$1356 * 2410 = 3.267.960$$



Solution

$$1356 * 2410 = 3.267.960$$

x | 1 | 3 | 5 | 6 | y | 2 | 4 | 1 | 0

| 1 | 3 | * | 2 | 4 |

$n = 2$

$$\begin{array}{cccccccccccccccccccc} 10^n & & x_l & & y_l & & 10^{n/2} & & x_l & & y_r & & x_r & & y_l & & & x_r & & y_r \\ 10^2 & * & \boxed{1} & * & \boxed{2} & + & 10^1 & *(& \boxed{1} & * & \boxed{4} & + & \boxed{3} & * & \boxed{2} &) & + & \boxed{3} & * & \boxed{4} \end{array}$$

$$13 * 24 = 10^2 * 2 + 10^1 * 10 + 12 = 312$$

Solution

1356 * 2410 = 3.267.960

x $\boxed{1}\boxed{3}\boxed{5}\boxed{6}$ y $\boxed{2}\boxed{4}\boxed{1}\boxed{0}$

$\boxed{1}\boxed{3}$ * $\boxed{1}\boxed{0}$ n = 2

$$\overset{10^n}{10^2} * \overset{x_l}{\boxed{1}} * \overset{y_l}{\boxed{1}} + \overset{10^{n/2}}{10^1} * (\overset{x_l}{\boxed{1}} * \overset{y_r}{\boxed{0}} + \overset{x_r}{\boxed{3}} * \overset{y_l}{\boxed{1}}) + \overset{x_r}{\boxed{3}} * \overset{y_r}{\boxed{0}}$$

$13 * 10 = 10^2 * 1 + 10^1 * 3 + 0 = 130$

Solution

$$1356 * 2410 = 3.267.960$$

x	1	3	5	6	y	2	4	1	0

5	6	*	2	4	n = 2

10^n		x_l		y_l	$10^{n/2}$		x_l		y_r		x_r		y_l		x_r		y_r
10^2	*	$\boxed{5}$	*	$\boxed{2}$	$+10^1$	$*($	$\boxed{5}$	*	$\boxed{4}$	+	$\boxed{6}$	*	$\boxed{2}$	$)+$	$\boxed{6}$	*	$\boxed{4}$

$$56 * 24 = 10^2 * 10 + 10^1 * 32 + 24 = 1344$$

Solution

1356 * 2410 = 3.267.960

$x \quad \boxed{1}\boxed{3}\boxed{5}\boxed{6} \qquad\qquad y \quad \boxed{2}\boxed{4}\boxed{1}\boxed{0}$

$\boxed{5}\boxed{6} \quad * \quad \boxed{1}\boxed{0} \qquad n = 2$

$$\overset{10^n}{10^2} * \overset{x_l}{\boxed{5}} * \overset{y_l}{\boxed{1}} + \overset{10^{n/2}}{10^1} * (\overset{x_l}{\boxed{5}} * \overset{y_r}{\boxed{0}} + \overset{x_r}{\boxed{6}} * \overset{y_l}{\boxed{1}}) + \overset{x_r}{\boxed{6}} * \overset{y_r}{\boxed{0}}$$

$56 * 10 = 10^2 * 5 + 10^1 * 6 + 0 = 560$

Solution

$$1356 * 2410 = 3.267.960$$

x	1	3	5	6	*	y	2	4	1	0	n = 4

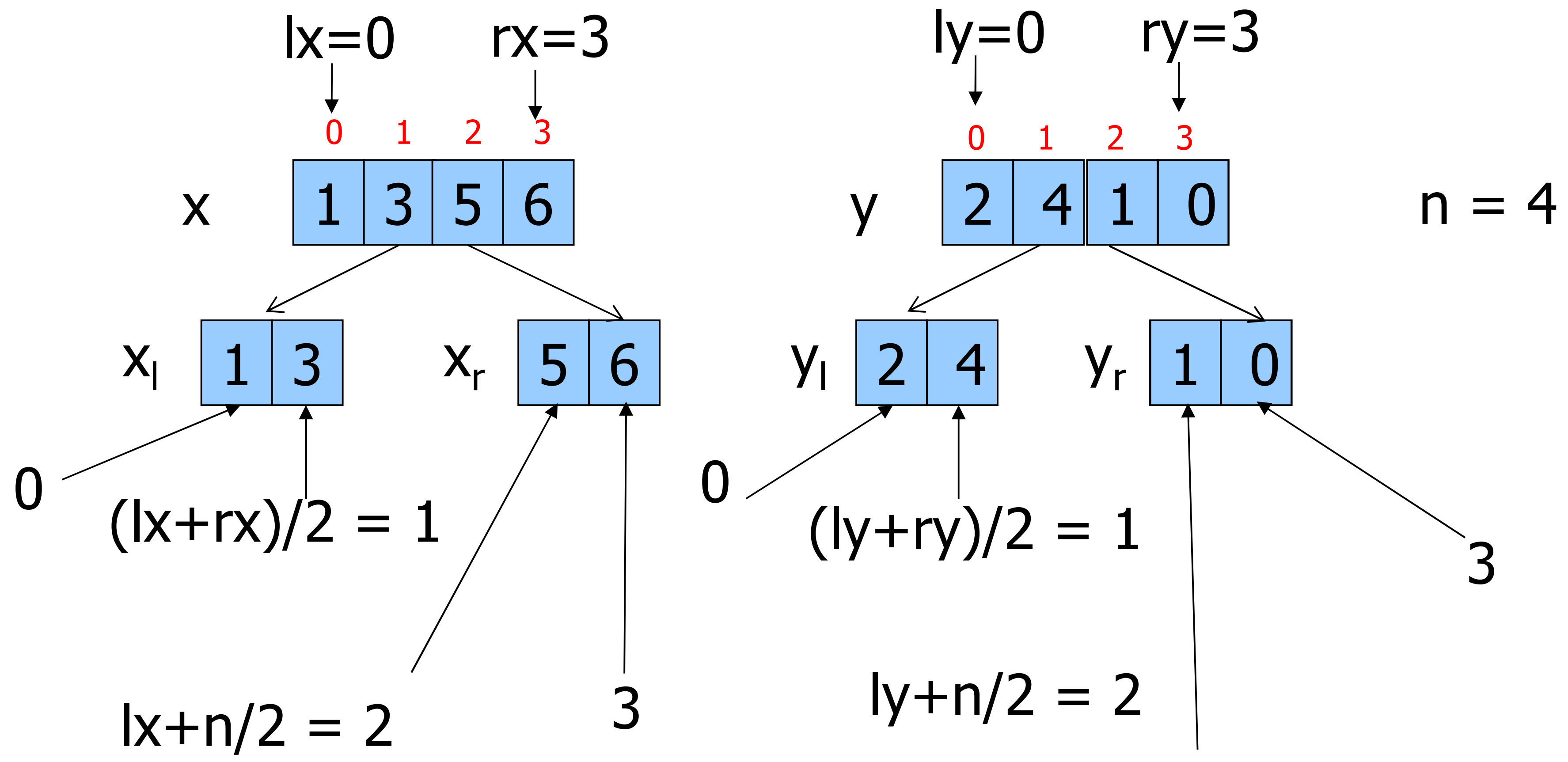
$$\begin{array}{cccccccccc} 10^n & x_l & y_l & 10^{n/2} & x_l & y_r & x_r & y_l & x_r & y_r \end{array}$$

$$10^4 * 13 * 24 + 10^2 * (13 * 10 + 56 * 24) + 56 * 10$$

$$1356 * 2410 = 10^4 * 312 + 10^2 * (130 + 1344) + 560 = 3.267.960$$

Solution

Identifying left and right subarrays:



Solution

```
long prod(int *x,int lx,int rx,int *y,int ly,int ry,int n) {
  int  i, t1, t2, t3;
  if (n > 1) {
    t1 = prod(x, lx, (lx+rx)/2, y, ly, (ly+ry)/2, n/2);
    t2 = prod(x, lx, (lx+rx)/2, y, ly+n/2, ry, n/2)
         + prod(x, lx+n/2, rx, y, ly, (ly+ry)/2, n/2);
    t3 = prod(x, lx+n/2, rx, y, ly + n/2, ry, n/2);
    return t1 * pow(10,n) + t2 * pow (10, n/2) + t3;
    }
  else
    return (x[lx]*y[ly]);
}
```

Complexity Analysis

- Multiplication by a power of 10
  - Left shift (unit cost) (2 multiplications)
- Addition of numbers on N digits
  - Cost linear in N (3 sums)
- Multiplication
  - Cost of recursion (4 multiplications)

Complexity Analysis

- Complexity anslysis
  - $D(n) = \Theta(1)$, $C(n) = \Theta(n)$
  - $D(n) + C(n) = \Theta(n)$
  - $a = 4$, $b = 2$
- Recurrence equation
  - $T(n) = D(n) + a\,T(n/b) + C(n)$
- That is
  - $T(n) = 4T(n/2) + n \qquad n > 1$
  - $T(1) = 1 \qquad n=1$

Complexity Analysis

Termination:
$n/2^i = 1$
$i = \log_2 n$

- Resolution by unfolding
  - T(n/2) = 4T(n/4) +n/2
  - T(n/4) = 4T(n/8) +n/4
  - etc.

$\sum_{i=0}^{k} x^i = (x^{k+1} - 1)/(x-1)$

- Then
  - $T(n) = n + 4*(n/2) + 4^2 *(n/4) + 4^3 *T(n/8)$

    $= \Sigma_{0\leq i \leq \log 2n} 4^i / 2^i * n = n * \Sigma_{0\leq i \leq \log 2n} 2^i$

    = n*(2log2n +1-1)/(2-1) = n *(2 * 2log2n -1)

    = 2n2-n
  - T(n) = O(n2)

Alternative Solution

- Karatsuba's algorithm (1962)
- Reducing the number of multiplications
  - $x_l * y_r + x_r * y_l = x_l * y_l + x_r * y_r - (x_l - x_r) * (y_l - y_r)$
  - 3 recursive multiplications instead of 4

Divide and conquer
a = 3 b = 2

Complexity Analysis

- Complexity analysis
  - $D(n) = \Theta(1)$, $C(n) = \Theta(n)$
  - $D(n) + C(n) = \Theta(n)$
  - $a = 3$, $b = 2$
- Recurrence equation
  - $T(n) = D(n) + a\,T(n/b) + C(n)$
- That is
  - $T(n) = 3T(n/2) + n \qquad n > 1$
  - $T(1) = 1 \qquad n = 1$

Complexity Analysis

Termination:
$n/2^i = 1$
$i = \log_2 n$

- Resolution by unfolding

$\sum_{i=0}^{k} x^i = (x^{k+1} - 1)/(x-1)$

$T(n/2) = 3T(n/4) + n/2$

$T(n/4) = 3T(n/8) + n/4$ etc.

$T(n) = n + 3*(n/2) + 3^2 *(n/4) + 3^3 *T(n/8)$

$= \Sigma_{0 \leq i \leq \log 2n} 3^i / 2^i * n = n * \Sigma_{0 \leq i \leq \log 2n} (3/2)^i$

$= n*((3/2)^{\log_2 n + 1} - 1)/(3/2 - 1)$

$= 2n * 3/2 * ((3^{\log_2 n} / 2^{\log_2 n}) - 1)$

$= 3n*(n^{\log_2 3} / n - 1)$

$= 3n^{\log_2 3} - n$

$T(n) = O(n^{\log_2 3})$

$a^{\log_b n} = n^{\log_b a}$