Algorithms

(Recursion)

Pramod Ganapathi
Department of Computer Science
State University of New York at Stony Brook

February 26, 2021

Contents

Recursion


demontroll.deviantart.com

What is recursion?

- Recursion is self-repetition or self-reproduction or self-reference.
- To understand recursion, you must understand *recursion*.
- Every nonrecursive algorithm can be written as a recursive algorithm. Every recursive algorithm can be written as a nonrecursive algorithm.
- There are typically multiple ways of writing recursive algorithms to solve a problem.

Why care for recursion?

- Nature $\rightarrow$ repetition in unicellular organisms
- Nature $\rightarrow$ reproduction in multicellular organisms
- Nature $\rightarrow$ fractals
- Natural languages $\rightarrow$ sentences
- Mathematics $\rightarrow$ recursive functions
- Computer science $\rightarrow$ recursive functions
- Computer science $\rightarrow$ algorithm design techniques
  - decrease-and-conquer
  - divide-and-conquer
  - dynamic programming
  - backtracking

Recursive Algorithms

Types of recursive algorithms

- General recursive algorithms
- Decrease-and-conquer
- Divide-and-conquer
- Backtracking

Factorial

- Factorial of a whole number n is defined as

$$n! = \begin{cases} 1 & \text{if } \; n = 0, \\ n \cdot (n-1) \cdot \dots \cdot 3 \cdot 2 \cdot 1 & \text{if } \; n \geq 1. \end{cases}$$

- Recursive definition of the factorial function is

$$n! = \begin{cases} 1 & \text{if } \; n = 0, \\ n \cdot (n-1)! & \text{if } \; n \geq 1. \end{cases}$$

Factorial: Iterative algorithm

FACTORIAL(n)

$factorial \leftarrow 1$
for $i \leftarrow 1$ **to** n **do**
  $factorial \leftarrow factorial \times i$
return $factorial$

```java
public static int factorial(int n) throws IllegalArgumentException {
  if (n < 0)
    throw new IllegalArgumentException(); // argument must be nonnegative
  else
    factorial = 1;
    for (int i = 2; i <= n; i++)
      factorial = factorial * i;
    return factorial;
}
```

Factorial: Decrease-and-conquer

FACTORIAL(n)

1. **if** $n = 0$ **then**
2. **return** 1
3. **else**
4. **return** $n \times$ FACTORIAL$(n-1)$

```java
public static int factorial(int n) throws IllegalArgumentException {
  if (n < 0)
    throw new IllegalArgumentException(); // argument must be nonnegative
  else if (n == 0)
    return 1;                          // base case
  else
    return n * factorial(n - 1);   // recursive case
}
```

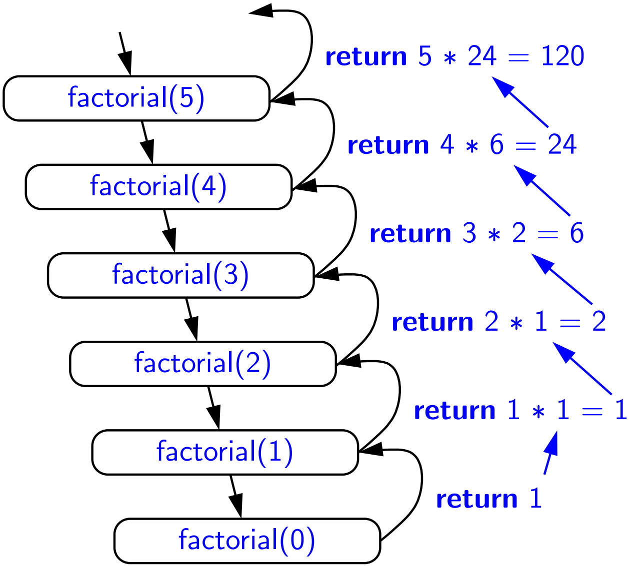
Factorial: Decrease-and-conquer

Compute factorial(5)

- Suppose the main function calls factorial(5)
- factorial$(5) = 5 \times$ factorial(4) recursive case
- factorial$(4) = 4 \times$ factorial(3) recursive case
- factorial$(3) = 3 \times$ factorial(2) recursive case
- factorial$(2) = 2 \times$ factorial(1) recursive case
- factorial$(1) = 1 \times$ factorial(0) recursive case
- factorial$(0) = 1$ base case
- factorial$(1) = 1 \times 1 = 1$ return
- factorial$(2) = 2 \times 1 = 2$ return
- factorial$(3) = 3 \times 2 = 6$ return
- factorial$(4) = 4 \times 6 = 24$ return
- factorial$(5) = 5 \times 24 = 120$ return
- factorial(5) returns 120 to the main function

120

Factorial: Decrease-and-conquer



Factorial: Decrease-and-conquer

- Time complexity.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 0, \\ T(n-1) + \Theta(1) & \text{if } n \geq 1. \end{cases}$$

Solving, $T(n) \in \Theta(n)$

- Space complexity.

$$S(n) = \begin{cases} \Theta(1) & \text{if } n = 0, \\ S(n-1) + \Theta(1) & \text{if } n \geq 1. \end{cases}$$

Solving, $S(n) \in \Theta(n)$ stack space

Array search

Problem

Two related problems:

1. Search/find/locate a given element in an unsorted array.
2. Search/find/locate a given element in a sorted array.

Unsorted array search: Iterative algorithm

$\text{LINEAR-SEARCH}(A[], target)$

1. **for** $i \leftarrow 0$ **to** $A.length - 1$ **do**
2. **if** $target = A[i]$ **then**
3. **return** i
4. **return** -1

```
public static int linearSearch(int[] data, int target) {
  for (int i = 0; i < data.length; i++)
    if (target == data[i])
      return i;   // return the first position where target can be found
  return -1;      // -1 denotes that target is not in the array
}
```

Runtime $\in \Theta(n)$

Unsorted array search: Decrease-and-conquer

$\textsc{Linear-Search}(A[i..n-1], target)$

1. **if** $i = n$ **then return** -1
2. **else if** $A[i] = target$ **then return** i
3. **else return** $\textsc{Linear-Search}(A[(i+1)..(n-1)], target)$

```java
public static int linearSearch(int[] data, int target) {
  linearSearch(data, 0, target);
}
```

```java
public static int linearSearch(int[] data, int target) {
  if (i == data.length)        return -1;
  else if (data[i] == target)  return i;
  else                         return linearSearch(data, i + 1, target);
}
```

Unsorted array search: Decrease-and-conquer

- Time complexity.

$$T(n) = \begin{cases} \Theta\left(1\right) & \text{if } n = 0, \\ T(n-1) + \Theta\left(1\right) & \text{if } n \geq 1. \end{cases}$$

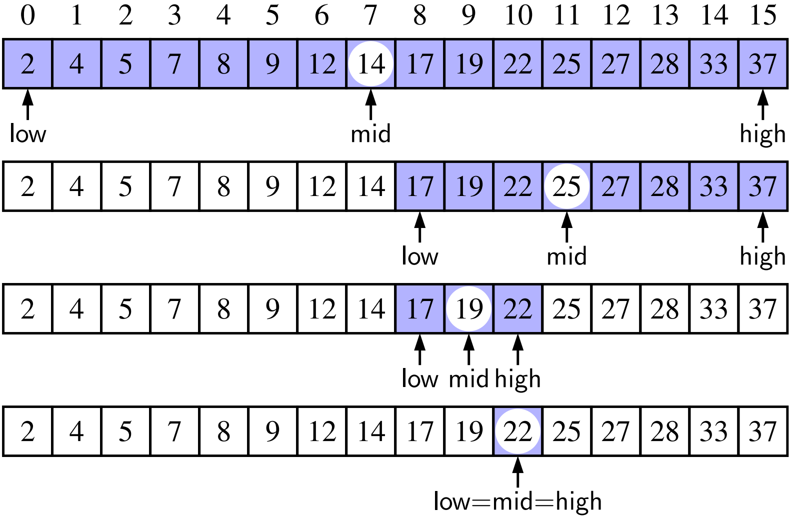
Solving, $T(n) \in \Theta\left(n\right)$

- Space complexity.

$$S(n) = \begin{cases} \Theta\left(1\right) & \text{if } n = 0, \\ S(n-1) + \Theta\left(1\right) & \text{if } n \geq 1. \end{cases}$$

Solving, $S(n) \in \Theta\left(n\right)$ stack space

Sorted array search: Decrease-and-conquer



Sorted array search: Decrease-and-conquer

$\textsc{Binary-Search}(A[], target)$

return $\textsc{Binary-Search}(A[], target, 0, A.length - 1)$

$\textsc{Binary-Search}(A[], target, low, high)$

if $low > high$ **then**
  return -1
else
  $mid \leftarrow (low + high)/2$
  if $target = A[mid]$ **then**
    return mid
  else if $target < A[mid]$ **then**
    return $\textsc{Binary-Search}(A[], target, low, mid - 1)$
  else if $target > A[mid]$ **then**
    return $\textsc{Binary-Search}(A[], target, mid + 1, high)$

Sorted array search: Decrease-and-conquer

```java
public static int binarySearch(int[ ] data, int target) {
  return binarySearch(data, target, 0, data.length - 1);
}
```

```java
public static int binarySearch(int[ ] data, int target, int low, int high) {
  if (low > high)                         // interval empty; no match
    return -1;
  else
  {
    int mid = (low + high) / 2;
    if (target == data[mid])          // found a match
      return mid;
    else if (target < data[mid])      // recur left
      return binarySearch(data, target, low, mid - 1);
    else if (target > data[mid])      // recur right
      return binarySearch(data, target, mid + 1, high);
  }
}
```

Sorted array search: Decrease-and-conquer

- Time complexity.

$$T(n) = \begin{cases} \Theta\left(1\right) & \text{if } n = 1, \\ T(n/2) + \Theta\left(1\right) & \text{if } n > 1. \end{cases}$$

Solving, $T(n) \in \Theta\left(\log n\right)$

- Space complexity.

$$S(n) = \begin{cases} \Theta\left(1\right) & \text{if } n = 1, \\ S(n/2) + \Theta\left(1\right) & \text{if } n > 1. \end{cases}$$

Solving, $S(n) \in \Theta\left(\log n\right)$ stack space

Sorted array search: Iterative algorithm

```
BINARY-SEARCH(A[], target)
low ← 0
high ← A.length − 1
while low ≤ high do
  mid ← (low + high)/2
  if target = A[mid] then
    return mid
  else if target < A[mid] then
    high ← mid − 1
  else if target > A[mid] then
    low ← mid + 1
return −1
```

Sorted array search: Iterative algorithm

```java
public static int binarySearch(int[ ] data, int target) {
  int low = 0;
  int high = data.length - 1;

  while (low <= high) {
    int mid = (low + high) / 2;

    if (target == data[mid])
      return mid;
    else if (target < data[mid])
      high = mid + 1;
    else
      low = mid + 1;
  }

  return -1;
}
```

Product

Problem

- Multiply two whole numbers without using the multiplication operator.

Product: Iterative algorithm

$\textsc{Product}(a, b)$

1. $product \leftarrow 0$
2. **for** $i \leftarrow 1$ **to** b **do**
3. $\quad product \leftarrow product + a$
4. **return** $product$

$$\text{Runtime} \in \Theta\left(b\right)$$

- What if b is much larger than a?

Product: Iterative algorithm

$\textsc{Product}(a, b)$

1. $product \leftarrow 0$
2. **for** $i \leftarrow 1$ **to** a **do**
3. $\quad product \leftarrow product + b$
4. **return** $product$

$$\text{Runtime} \in \Theta\left(a\right)$$

- What if a is much larger than b?

Product: Iterative algorithm

$\textsc{Product}(a, b)$

1. $product \leftarrow 0$
2. **for** $i \leftarrow 1$ **to** $\textsc{Min}(a, b)$ **do**
3. $\quad product \leftarrow product + \textsc{Max}(a, b)$
4. **return** $product$

$$\text{Runtime} \in \Theta\left(\min(a, b)\right)$$

- Can we do better?

Product: Decrease-and-conquer

$\textsc{Product}(a, b)$

1. **if** $b = 0$ **then**
2. **return** 0
3. **else**
4. **return** $a + \textsc{Product}(a, b - 1)$

$$\text{Runtime} = T(n) \leq \begin{Bmatrix} \Theta(1) & \text{if } n = 0, \\ T(n-1) + \Theta(1) & \text{if } n > 0. \end{Bmatrix} \in \Theta(b)$$

- What if b is much larger than a?

Product: Decrease-and-conquer

$\textsc{Product}(a, b)$

1. **if** $a = 0$ **then**
2. **return** 0
3. **else**
4. **return** $b + \textsc{Product}(a-1, b)$

$$\text{Runtime} = T(n) \leq \left.\begin{cases} \Theta(1) & \text{if } n = 0, \\ T(n-1) + \Theta(1) & \text{if } n > 0. \end{cases}\right\} \in \Theta(a)$$

- What if a is much larger than b?

Product: Decrease-and-conquer

$\textsc{Mult}(a, b)$

$max \leftarrow \textsc{Max}(a, b)$
$min \leftarrow \textsc{Min}(a, b)$
return $\textsc{Product}(max, min)$

$\textsc{Product}(a, b)$

if $b = 0$ **then**
  return 0
else
  return $a + \textsc{Product}(a, b - 1)$

$$\text{Runtime} = T(n) \leq \left.\begin{cases} \Theta\left(1\right) & \text{if } n = 0, \\ T(n-1) + \Theta\left(1\right) & \text{if } n > 0. \end{cases}\right\} \in \Theta\left(\min(a, b)\right)$$

- Can we do better?

Product: Divide-and-conquer

```
MULT(a, b)
max ← MAX(a, b)
min ← MIN(a, b)
return PRODUCT(max, min)
```

```
PRODUCT(a, b)
if b = 0 then return 0
else
  part1 ← PRODUCT(a, b/2)
  part2 ← PRODUCT(a, b/2)
  if b % 2 = 1 then
    return part1 + part2 + a
  else
    return part1 + part2
```

$$\text{Runtime} = T(n) \leq \left.\begin{cases} \Theta\left(1\right) & \text{if } n = 0, \\ 2T(n/2) + \Theta\left(1\right) & \text{if } n > 0. \end{cases}\right\} \in \mathcal{O}\left(\min(a, b)\right)$$

- Can we do better?

Product: Decrease-and-conquer

```
MULT(a, b)
max ← MAX(a, b); min ← MIN(a, b)
return PRODUCT(max, min)
```

```
PRODUCT(a, b)
if b = 0 then return 0
else
  partial ← PRODUCT(a, b/2)
  if b % 2 = 1 then
    return partial + partial + a
  else
    return partial + partial
```

$$\text{Runtime} = T(n) \leq \left\{ \begin{array}{ll} \Theta(1) & \text{if } n = 0, \\ T(n/2) + \Theta(1) & \text{if } n > 0. \end{array} \right\} \in \mathcal{O}(\log_2 \min(a, b))$$

- Can we do better?

Product: Decrease-and-conquer

$\textsc{Mult}(a, b)$

$max \leftarrow \textsc{Max}(a, b)$; $min \leftarrow \textsc{Min}(a, b)$
return $\textsc{Product}(max, min)$

$\textsc{Product}(a, b)$

if $b = 0$ **then return** 0
else
  $partial \leftarrow \textsc{Product}(a, b/3)$
  if $b \,\%\, 3 = 1$ **then**
    return $partial + partial + partial + a$
  if $b \,\%\, 3 = 2$ **then**
    return $partial + partial + partial + a + a$
  else
    return $partial + partial + partial$

$$\text{Runtime} = T(n) \le \left\{ \begin{array}{ll} \Theta(1) & \text{if } n = 0, \\ T(n/3) + \Theta(1) & \text{if } n > 0. \end{array} \right\} \in \mathcal{O}\left(\log_3 \min(a, b)\right)$$

- Can we do better?

Exponentiation

Problem

- How do you compute a^n for $n \geq 0$? (e.g. 17^{8943})
  Generalization: The element a can be a number, a matrix, or a polynomial.

Exponentiation: Iterative algorithm

$\text{POWER}(a, n)$

Input: Real number a and whole number n

Output: a^n

1. $result \leftarrow 1$
2. **for** $i \leftarrow 1$ **to** n **do**
3. $\quad result \leftarrow result \times a$
4. **return** $result$

```
public static double power(double a, int n) {
  double result = 1.0;
  for (i = 1; i <= n; i++)
    result = result * a;
  return result;
}
```

$$\text{Runtime} = \Theta\left(n\right)$$

Exponentiation: Decrease-and-conquer

$\textsc{Power}(a, n)$

Input: Real number a and whole number n
Output: a^n
1. **if** $n = 0$ **then**
2. **return** 1
3. **else**
4. **return** $a \times \textsc{Power}(a, n-1)$

```
public static double power(double a, int n) {
  if (n == 0)
    return 1;
  else
    return a * power(a, n-1);
}
```

$$\text{Runtime} = T(n) \leq \left\{ \begin{array}{ll} \Theta(1) & \text{if } n = 0, \\ T(n-1) + \Theta(1) & \text{if } n > 0. \end{array} \right\} \in \Theta(n)$$

Exponentiation: Decrease-and-conquer

- Observation.

$$\begin{aligned} a^{10} &= (a^5)^2 \\ a^5 &= (a^2)^2 \times a \\ a^2 &= (a^1)^2 \\ a^1 &= (a^0)^2 \times a \\ a^0 &= 1 \end{aligned}$$

- Core idea.

$$a^n = \begin{cases} 1 & \text{if } n = 0, \\ (a^{\lfloor n/2 \rfloor})^2 & \text{if } n \geq 1 \text{ and } n \text{ is even,} \\ (a^{\lfloor n/2 \rfloor})^2 \times a & \text{if } n \geq 1 \text{ and } n \text{ is odd.} \end{cases}$$

- This idea is called repeated squaring/doubling.

Exponentiation: Decrease-and-conquer

$\textsc{Power}(a, n)$

if $n = 0$ **then return** 1
else
  $result \leftarrow \textsc{Power}(a, \lfloor n/2 \rfloor)$
  $result \leftarrow result \times result$
  if n is odd **then** $result \leftarrow result \times a$
  return $result$

```java
public static double power(double a, int n) {
  if (n == 0) return 1;
  else {
    double partial = power(a, n/2); // rely on truncated division of n
    double result = partial * partial;
    if (n % 2 == 1) result = result * a;
    return result;
  }
}
```

$$\text{Runtime} = T(n) \leq \begin{Bmatrix} \Theta(1) & \text{if } n = 0, \\ T(\lfloor n/2 \rfloor) + \Theta(1) & \text{if } n > 0. \end{Bmatrix} \in \Theta(\log n)$$

Exponentiation: Real exponent

Problem

- How can we compute a^n when n is a rational/real number? Please note that there can be multiple solutions to a^n, when n is a real number but not an integer. E.g. $a^{1/100}$ has 100 roots.

Array sum

Problem

- Compute the sum of elements of a given array.

Array sum: Iterative algorithm

ARRAY-SUM$(A[0..n-1])$

1. $sum \leftarrow 0$
2. **for** $i \leftarrow 0$ **to** $n-1$ **do**
3. $\quad sum \leftarrow sum + A[i]$
4. **return** sum

```
public static double arraySum(int[] data) {
  int sum = 0;
  for (i = 0; i < data.length; i++)
    sum = sum + data[i];
  return sum;
}
```

$$\text{Runtime} = \Theta(n)$$

Array sum: Decrease-and-conquer

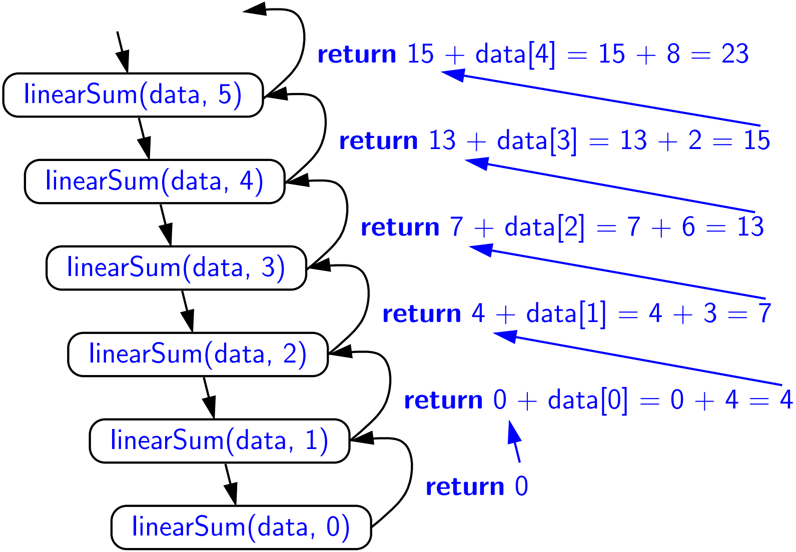
$\textsc{Array-Sum}(A[0..(n-1)])$

if $n = 1$ **then**
  return $A[0]$
else
  return $\textsc{Array-Sum}(A[0..(n-2)]) + A[n-1]$

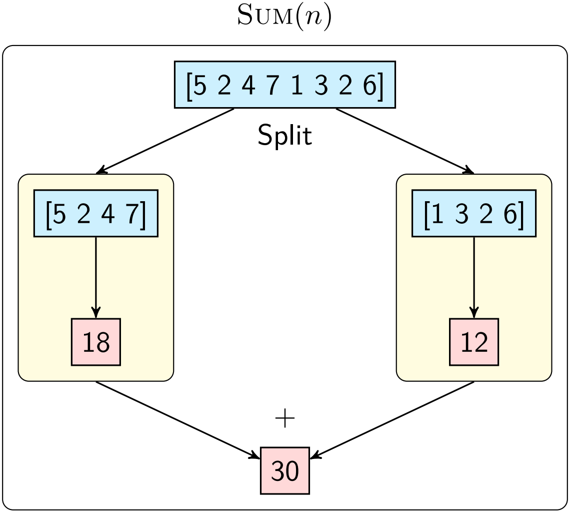
```java
public static double arraySum(int[] data) {
  arraySum(data, data.length);
}
public static double arraySum(int[] data, int n) {
  if (n == 0)
    return 0;
  else
    return arraySum(data, n - 1) + data[n - 1];
}
```

$$\text{Runtime} = T(n) \leq \left\{ \begin{array}{ll} \Theta\left(1\right) & \text{if } n = 0, \\ T(n-1) + \Theta\left(1\right) & \text{if } n > 0. \end{array} \right\} \in \Theta\left(n\right)$$

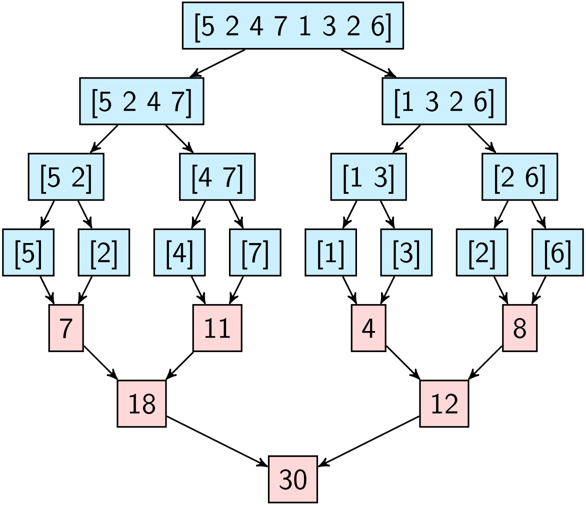
Array sum: Decrease-and-conquer



Array sum: Divide-and-conquer: Core idea



Array sum: Divide-and-conquer: Example



Array sum: Divide-and-conquer: Algorithm

ARRAY-SUM($A[low..high]$)

if $low > high$ **then**
  return 0
else if $low = high$ **then**
  return $A[mid]$
else if $low < high$ **then**
  return $mid \leftarrow (low + high)/2$
  $part1 \leftarrow$ ARRAY-SUM($A[low..mid]$)
  $part2 \leftarrow$ ARRAY-SUM($A[(mid + 1)..high]$)
  return $part1 + part2$

Array sum: Divide-and-conquer: Code

```
public static double arraySum(int[] data) {
  arraySum(data, 0, data.length - 1);
}
```

```
public static int arraySum(int[ ] data, int low, int high) {
  if (low > high)       // zero elements in subarray
    return 0;
  else if (low == high) // one element in subarray
    return data[low];
  else {
    int mid = (low + high) / 2;
    return arraySum(data, low, mid) + arraySum(data, mid + 1, high);
  }
}
```

Array sum: Divide-and-conquer: Complexity

$$\text{Runtime} = T(n) \leq \left.\begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(\lceil n/2 \rceil) + \Theta(1) & \text{if } n > 1. \end{cases}\right\} \in \Theta(n)$$

Fibonacci number

- Compute the nth Fibonacci number F_n, defined as:

$$F_n = \begin{cases} 0 & \text{if} \quad n = 0, \\ 1 & \text{if} \quad n = 1, \\ F_{n-1} + F_{n-2} & \text{if} \quad n \geq 2. \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
F_n	0	1	1	2	3	5	8	13	21	34	55	89	144	$\cdots$

Fibonacci number: Recursive algorithm

F(n)

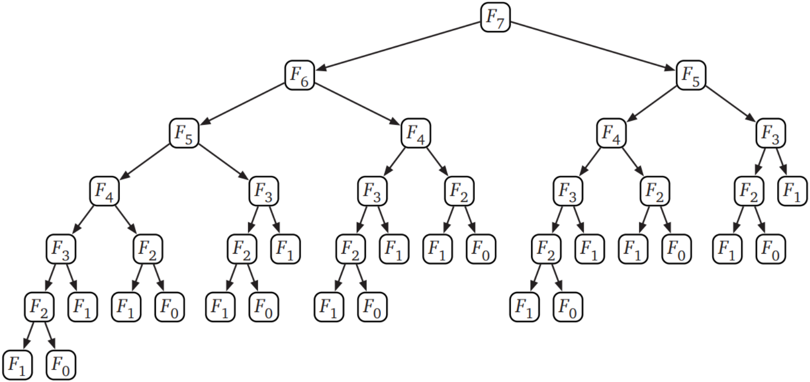
1. **if** $n = 0$ **or** $n = 1$ **then**
2. **return** n
3. **else**
4. **return** $\mathrm{F}(n-1) + \mathrm{F}(n-2)$

```java
public static long F(int n) {
  if (n <= 1)
    return n;
  else
    return (F(n-1) + F(n-2));
}
```

$$\text{Runtime} = T(n) \leq \left\{ \begin{array}{ll} \Theta(1) & \text{if } n = 0 \text{ or } 1, \\ T(n-1) + T(n-2) + \Theta(1) & \text{if } n > 1. \end{array} \right\}$$

$$\in \Theta(\phi^n), \text{ where } \phi \text{ is the golden ratio.}$$

Fibonacci number: Recursive algorithm



Source: Jeff Erickson's Algorithms textbook

Fibonacci number: Top-down DP

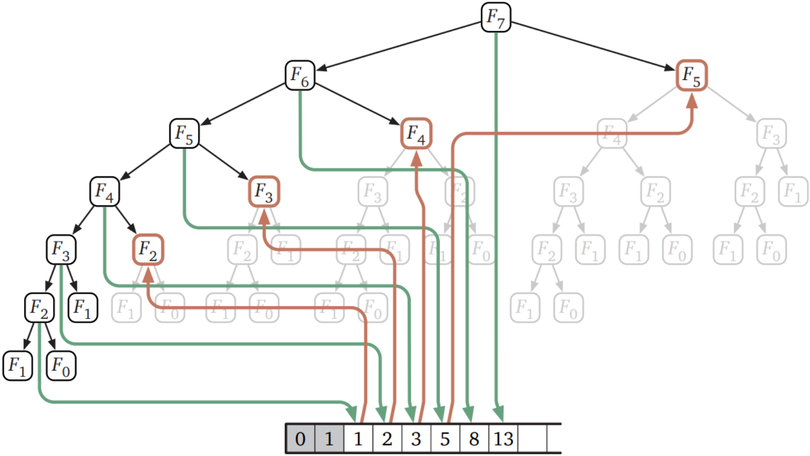
F(n)

1. **if** $n \leq 1$ **then return** n
2. **if** $f[n]$ is undefined **then**
3. $\quad f[n] \leftarrow$ F$(n-1)+$ F$(n-2)$
4. **return** $f[n]$

```java
public static long F(int n) {
  f[0] = 0; f[1] = 1;
  for (int i = 2; i <= n; i++) f[i] = -1;  // represents empty cell
  return G(n);
}
public static long G(int n) {
  if (f[n] == -1)                          // if empty cell, compute
    f[n] = G(n-1) + G(n-2);
  return f[n];                             // if nonempty cell, return value
}
```

Runtime $\in \Theta(n)$

Fibonacci number: Top-down DP



Source: Jeff Erickson's Algorithms textbook

Fibonacci number: Inefficient Bottom-up DP

$\mathrm{F}(n)$

1. **if** $n \leq 1$ **then**
2. **return** $\{n, 0\}$
3. **else**
4. $temp[] \leftarrow \mathrm{F}(n-1)$
5. **return** $\{temp[0] + temp[1], temp[0]\}$

```
public static long[] F(int n) {
  if (n <= 1) {
    long[] answer = {n, 0};
    return answer;
  } else {
    long[] temp = F(n-1);                           // returns {Fn-1, Fn-2}
    long[] answer = {temp[0] + temp[1], temp[0]}; // we want {Fn, Fn-1}
    return answer;
  }
}
```

Runtime $\in \Theta(n)$

Fibonacci number: Bottom-up DP

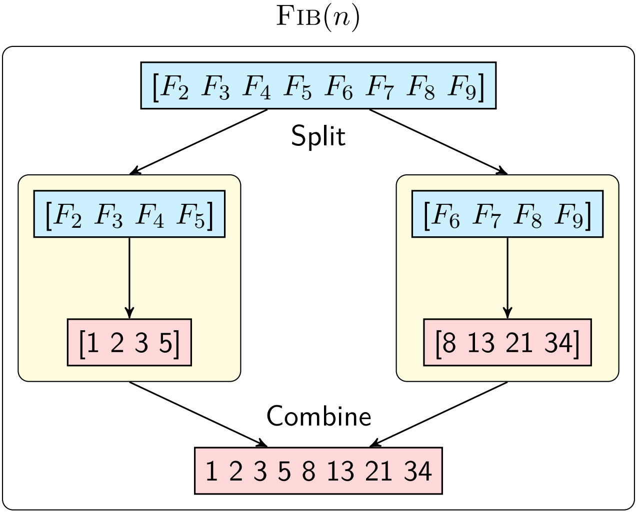
$\mathrm{F}(n)$

1. $f[0] \leftarrow 0;\ f[1] \leftarrow 1$
2. **for** $i \leftarrow 2$ **to** n **do**
3. $\quad f[i] \leftarrow f[i-1] + f[i-2]$
4. **return** $f[n]$

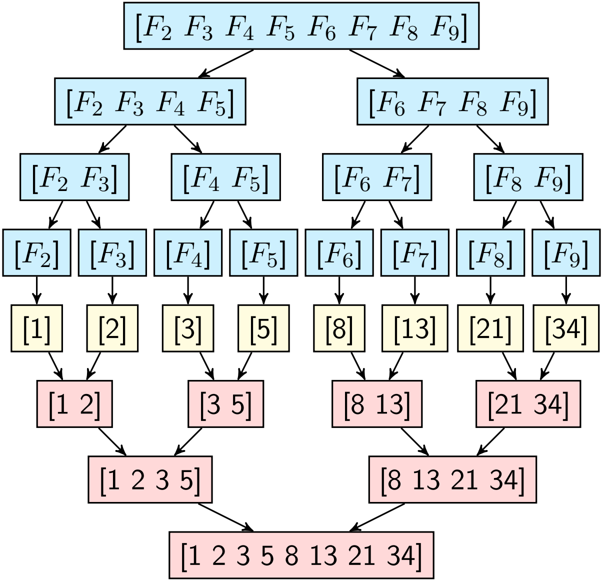
```java
public static long F(int n)
{
  long[] f = new long[100];
  f[0] = 0; f[1] = 1;
  for (int i = 2; i <= n; i++)
    f[i] = f[i-1] + f[i-2];
  return f[n];
}
```

Runtime $\in \Theta(n)$

Fibonacci: Bottom-up D&C DP: Core idea



Fibonacci: Bottom-up D&C DP: Example



Fibonacci: Bottom-up D&C DP: Algorithm

$\mathrm{F}(n)$

Input: Whole number n
Output: nth Fibonacci number F_n

1. **if** $n = 0$ **or** $n = 1$ **then return** n
2. **else**
3. $f[0] \leftarrow 0$; $f[1] \leftarrow 1$
4. $\mathrm{F\text{-}D\&C}(f[2..n])$
5. **return** $f[n]$

$\mathrm{F\text{-}D\&C}(F[low..high])$

Input: Empty array $f[low..high]$ such that $2 \leq low \leq high$
Output: Array $f[low..high]$ filled with Fibonacci numbers

1. **if** $low = high$ **then**
2. $f[low] \leftarrow f[low-1] + f[low-2]$
3. **else**
4. $mid \leftarrow (low + high)/2$
5. $\mathrm{F\text{-}D\&C}(f[low..mid])$
6. $\mathrm{F\text{-}D\&C}(f[mid+1..high])$

Fibonacci: Bottom-up D&C DP: Complexity

- Time complexity.

$$T(n) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1, \\ 2T(n/2) + \mathcal{O}(1) & \text{if } n > 1. \end{cases}$$

Solving, $T(n) \in \Theta(n)$

- Space complexity.

$$S(n) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1, \\ 2S(n/2) & \text{if } n > 1. \end{cases}$$

Solving, total space $S(n) \in \Theta(n)$

Fibonacci number: Efficient Bottom-up DP

F(n)

1. $curr \leftarrow 0$, $prev \leftarrow 1$
2. **for** $i \leftarrow 1$ **to** n **do**
3. $\quad next \leftarrow curr + prev$, $prev \leftarrow curr$, $curr \leftarrow next$
4. **return** $curr$

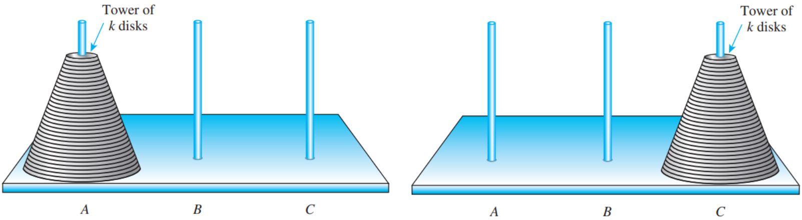
```java
public static long F(int n)
{
  long curr = 0, prev = 1, next;
  for (int i = 1; i <= n; i++)
  { next = curr + prev; prev = curr; curr = next; }
  return curr;
}
```

Runtime $\in \Theta(n)$

Towers of Hanoi: Problem

Problem

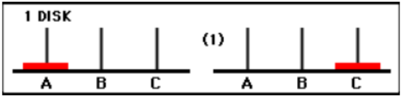
- Constraints: There are k disks on peg A. You can use peg B as an auxiliary peg. At any time, you cannot place a larger disk on a smaller disk.
- How do you move all k disks from peg A to peg C with the minimum number of moves?



Towers of Hanoi: Solution

Solution

Suppose $k = 1$. Then, the 1-step solution is:
1. Move disk 1 from peg A to peg C.



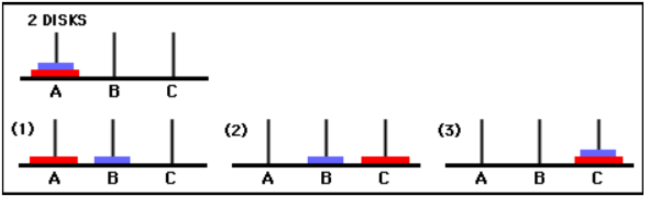
Source: `http://mathforum.org/dr.math/faq/faq.tower.hanoi.html`

Towers of Hanoi: Solution

Solution

Suppose $k = 2$. Then, the 3-step solution is:

1. Move disk 1 from peg A to peg B.
2. Move disk 2 from peg A to peg C.
3. Move disk 1 from peg B to peg C.



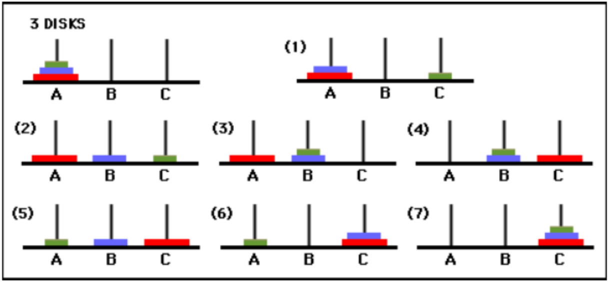
Source: `http://mathforum.org/dr.math/faq/faq.tower.hanoi.html`

Towers of Hanoi: Solution

Solution

Suppose $k = 3$. Then, the 7-step solution is:

1. Move disk 1 from peg A to peg C.
2. Move disk 2 from peg A to peg B.
3. Move disk 1 from peg C to peg B.
4. Move disk 3 from peg A to peg C.
5. Move disk 1 from peg B to peg A.
6. Move disk 2 from peg B to peg C.
7. Move disk 1 from peg A to peg C.



Towers of Hanoi: Solution

Solution

Suppose $k = 4$. Then, the 15-step solution is:

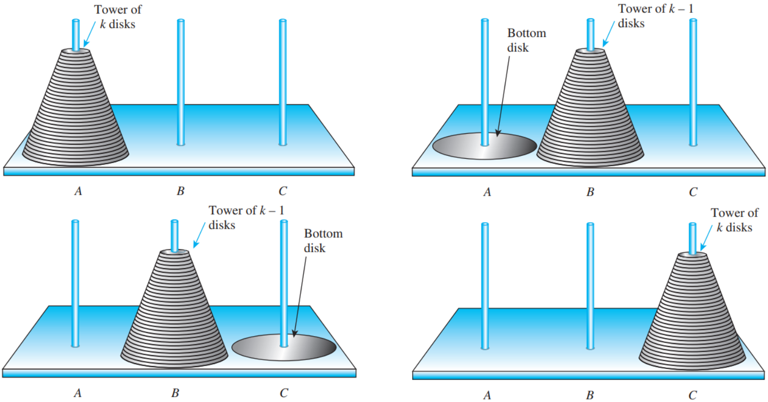
1. Move disk 1 from peg A to peg B.
2. Move disk 2 from peg A to peg C.
3. Move disk 1 from peg B to peg C.
4. Move disk 3 from peg A to peg B.
5. Move disk 1 from peg C to peg A.
6. Move disk 2 from peg C to peg B.
7. Move disk 1 from peg A to peg B.
8. Move disk 4 from peg A to peg C.
9. Move disk 1 from peg B to peg C.
10. Move disk 2 from peg B to peg A.
11. Move disk 1 from peg C to peg A.
12. Move disk 3 from peg B to peg C.
13. Move disk 1 from peg A to peg B.
14. Move disk 2 from peg A to peg C.
15. Move disk 1 from peg B to peg C.

Towers of Hanoi: Solution

Solution

For any $k \geq 2$, the recursive solution is:

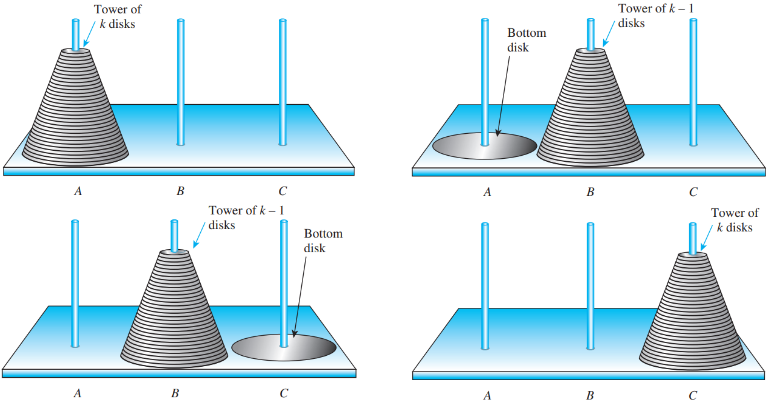
1. Transfer the top $k-1$ disks from peg A to peg B.
2. Move the bottom disk from peg A to peg C.
3. Transfer the top $k-1$ disks from peg B to peg C.



Towers of Hanoi: Algorithm

$\textsc{Towers-of-Hanoi}(k, A, C, B)$

if $k = 1$ **then**
    Move disk k from A to C.
else if $k \geq 2$ **then**
    $\textsc{Towers-of-Hanoi}(k-1, A, B, C)$
    Move disk k from A to C.
    $\textsc{Towers-of-Hanoi}(k-1, B, C, A)$



Towers of Hanoi: Complexity

- Time complexity.
  Let $M(n)$ denote the minimum number of moves required to move n disks from one peg to another peg. Then

$$M(n) = \begin{cases} 1 & \text{if } \ n = 1, \\ 2 \cdot M(n-1) + 1 & \text{if } \ n \geq 2. \end{cases}$$

  Solving, we get $M(n) = 2^n - 1$. So, $T(n) = \Theta\left(2^n\right)$.
- Space complexity.
  $S(n) = \Theta\left(n\right)$.

Greatest common divisor

Definition

- The greatest common divisor (GCD) of two integers a and b is the largest integer that divides both a and b.
- A simple way to compute GCD:
  1. Find the divisors of the two numbers
  2. Find the common divisors
  3. Find the greatest of the common divisors

Examples

- GCD$(2, 100) = 2$
- GCD$(3, 99) = 3$
- GCD$(3, 4) = 1$
- GCD$(12, 30) = 6$
- GCD$(1071, 462) = 21$

Greatest common divisor: Core idea

- Recurrence relation: Suppose $a > b$.
  $$\mathsf{GCD}(a, b) = \begin{cases} a & \text{if } \ b = 0, \\ \mathsf{GCD}(b, a \bmod b) & \text{if } \ b \geq 1. \end{cases}$$
- $\mathsf{GCD}(1071, 462)$
  $= \mathsf{GCD}(462, 1071 \bmod 462)$
  $= \mathsf{GCD}(462, 147) \qquad (\because 1071 = 2 \cdot 462 + 147)$
  $= \mathsf{GCD}(147, 462 \bmod 147)$
  $= \mathsf{GCD}(147, 21) \qquad (\because 462 = 3 \cdot 147 + 21)$
  $= \mathsf{GCD}(21, 147 \bmod 21)$
  $= \mathsf{GCD}(21, 0) \qquad (\because 147 = 7 \cdot 21 + 0)$
  $= 21$
- https://upload.wikimedia.org/wikipedia/commons/1/1c/Euclidean_algorithm_1071_462.gif

Greatest common divisor: Algorithm

$\mathrm{GCD}(a, b)$

Input: Nonnegative integers a and b such that $a > b$.
Output: Greatest common divisor of a and b.

1. **if** $b = 0$ **then**
2.   **return** a
3. **else**
4.   **return** $\mathrm{GCD}(b, a \bmod b)$

Greatest common divisor: Complexity

- Time complexity.

$$T(a, b) = \log \min(a, b)$$

- Space complexity.

$$S(a, b) = \log \min(a, b)$$

Recursion Caveats

Infinite recursion

```java
public static int square(int n) {
  return square(n); // After all square(n) does equal square(n)
}
```

```java
public static int binarySearch(int[ ] data, int target, int low, int high) {
  if (low > high)                       // interval empty; no match
    return -1;
  else
  {
    int mid = (low + high) / 2;
    if (target == data[mid])          // found a match
      return mid;
    else if (target < data[mid])      // recur left
      return binarySearch(data, target, low, mid - 1);
    else if (target > data[mid])      // recur right: infinite recursion
      return binarySearch(data, target, mid, high);
  }
}
```