CS514: Design and Analysis of Algorithms

Recursion

Arijit Mondal

Dept of CSE

arijit@iitp.ac.in

https://www.iitp.ac.in/~arijit/

Integer Multiplication - 1

- Input: Two non-negative integers, x, y
- Output: The product, $x \times y$

Integer Multiplication - 1

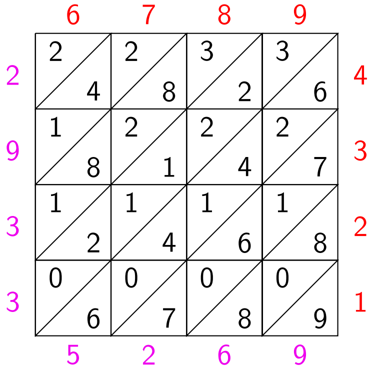
- Input: Two non-negative integers, x, y
- Output: The product, $x \times y$

```
          6 7 8 9
        x 4 3 2 1
---------------
          6 7 8 9
      1 3 5 7 8 x
    2 0 3 6 7 x x
  2 7 1 5 6 x x x
---------------
  2 9 3 3 5 2 6 9
```

Integer Multiplication - 1

- Input: Two non-negative integers, x, y
- Output: The product, $x \times y$

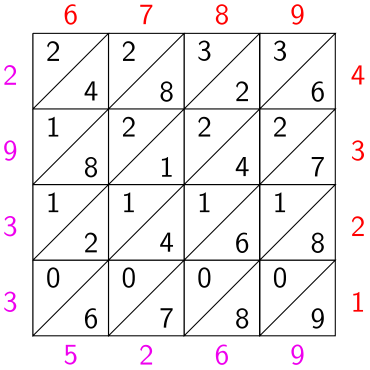
```
      6 7 8 9
    x 4 3 2 1
---------------
      6 7 8 9
    1 3 5 7 8 x
  2 0 3 6 7 x x
2 7 1 5 6 x x x
---------------
2 9 3 3 5 2 6 9
```



Integer Multiplication - 1

- Input: Two non-negative integers, x, y
- Output: The product, $x \times y$

```
        6 7 8 9
      x 4 3 2 1
---------------
        6 7 8 9
    1 3 5 7 8 x
  2 0 3 6 7 x x
2 7 1 5 6 x x x
---------------
2 9 3 3 5 2 6 9
```

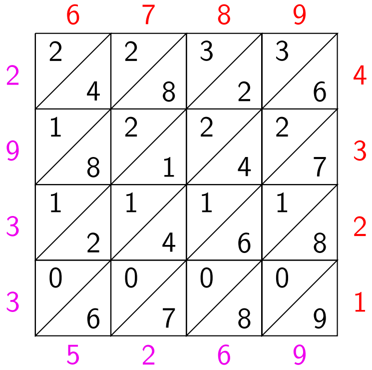


- Number of operations performed?

Integer Multiplication - 1

- Input: Two non-negative integers, x, y
- Output: The product, $x \times y$

```
        6 7 8 9
      x 4 3 2 1
---------------
        6 7 8 9
      1 3 5 7 8 x
    2 0 3 6 7 x x
  2 7 1 5 6 x x x
---------------
2 9 3 3 5 2 6 9
```



- Number of operations performed?
- Time complexity?

Integer Multiplication - 2

- mult1(x,y,n):

1. if n==0 return 0;
2. $x' =$ x[n,2]; $x_1 =$ x[1];
3. m=mult1(x', y, n-1);
4. return $10 \times m + x_1 \times y$

Integer Multiplication - 2

- mult1(x,y,n):
  1. if n==0 return 0;
  2. $x' =$ x[n,2]; $x_1 =$ x[1];
  3. m=mult1(x', y, n-1);
  4. return $10 \times m + x_1 \times y$
- mult2(x,y,n)
  1. if y==0 return 0
  2. z = mult2(x, $\lfloor y/2 \rfloor$)
  3. if y is even then return $2 \times m$
  4. else return $2 \times m + x$

Integer Multiplication - 3

- mult3(x,y,n):
  1. if n==1 return $x * y$;
  2. $x_l, x_r =$ left and right half of x
  3. $y_l, y_r =$ left and right half of y
  4. m_1=mult3$(x_l, y_l, n/2)$;
  5. m_2=mult3$(x_r, y_r, n/2)$;
  6. m_3=mult3$(x_l, y_r, n/2)$;
  7. m_4=mult3$(x_r, y_l, n/2)$;
  8. return $2^n * m_1 + 2^{n/2} * (m_3 + m_4) + m_2$

Integer Multiplication - 4

- mult4(x,y,n):

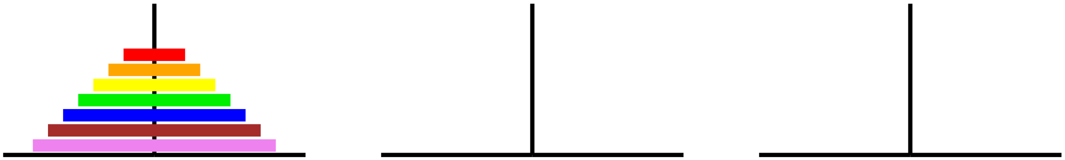
1. if n==1 return $x * y$;
2. $x_l, x_r =$ left and right half of x
3. $y_l, y_r =$ left and right half of y
4. m_1=mult4$(x_l, y_l, n/2)$;
5. m_2=mult4$(x_r, y_r, n/2)$;
6. m_3=mult4$(x_l + x_r, y_l + y_r, n/2)$;
7. return $2^n * m_1 + 2^{n/2} * (m_3 - m_1 - m_2) + m_2$

Recursion Example

- Tower of Hanoi
- GCD
- Fibonacci numbers
- Combination

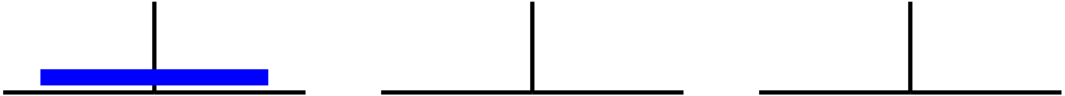
Tower of Hanoi

- Problem:
  - There are three pegs and 8 circular disks.
  - Each disk has a different size and a smaller disk can rest on a larger disk
  - At a time only one disk can be moved from one peg to the other
  - What is the minimum number of moves required?
- How many moves will be required when the number of disk is 64? How much time will it take to execute?
  - Known as Tower of Brahma



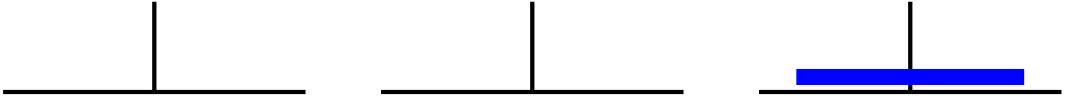
Tower of Hanoi - 1

- Moves when there is 1 disk



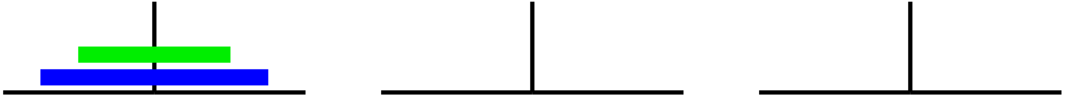
Tower of Hanoi - 1

- Moves when there is 1 disk



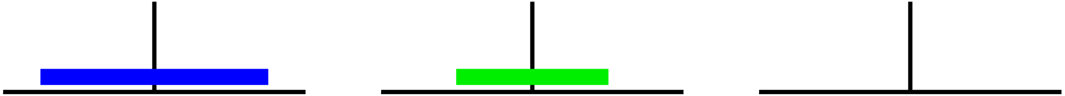
Tower of Hanoi - 2

- Moves when there are 2 disks



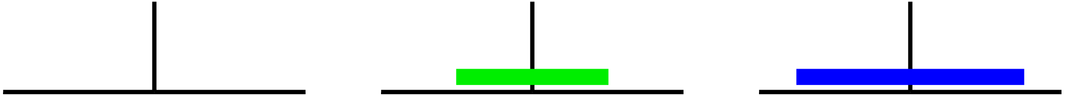
Tower of Hanoi - 2

- Moves when there are 2 disks



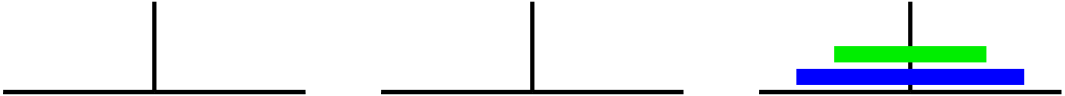
Tower of Hanoi - 2

- Moves when there are 2 disks



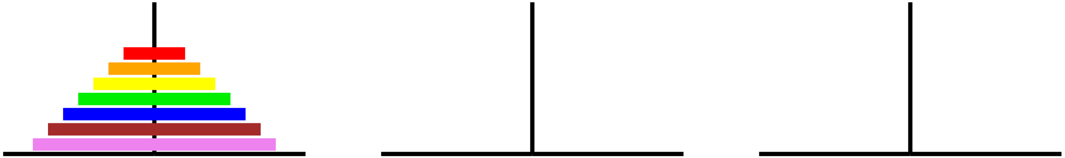
Tower of Hanoi - 2

- Moves when there are 2 disks



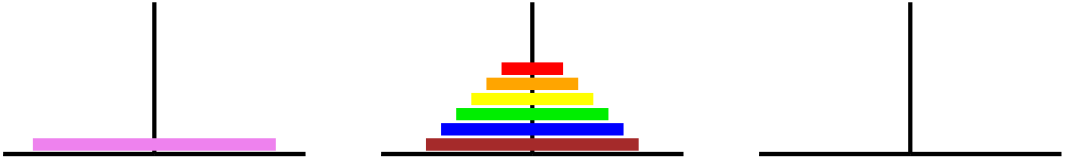
Tower of Hanoi

- Problem:
- What is the strategy to move n disks? How many moves will be required when the number of disk is n?



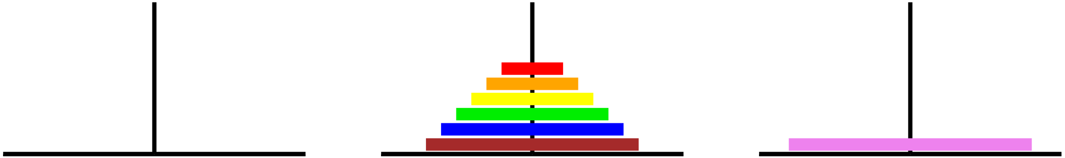
Tower of Hanoi

- Problem:
- What is the strategy to move n disks? How many moves will be required when the number of disk is n?



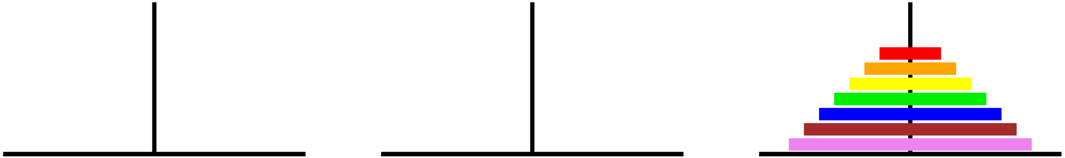
Tower of Hanoi

- Problem:
- What is the strategy to move n disks? How many moves will be required when the number of disk is n?



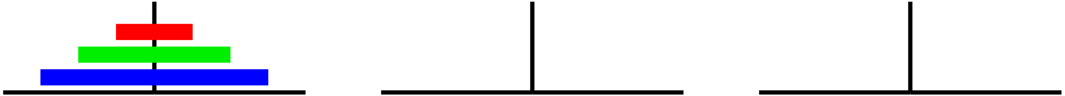
Tower of Hanoi

- Problem:
- What is the strategy to move n disks? How many moves will be required when the number of disk is n?



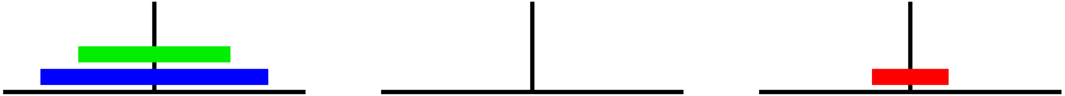
Tower of Hanoi - 3

- Moves when there are 3 disks



Tower of Hanoi - 3

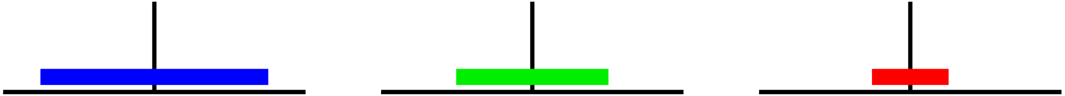
- Moves when there are 3 disks



Move A to C

Tower of Hanoi - 3

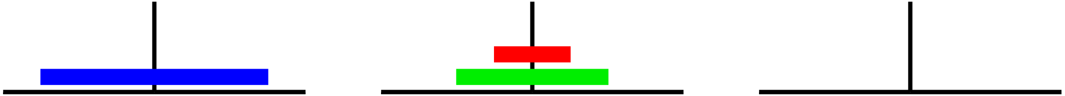
- Moves when there are 3 disks



Move A to C
Move A to B

Tower of Hanoi - 3

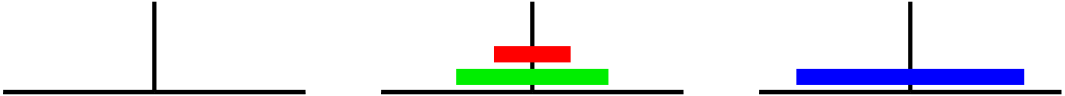
- Moves when there are 3 disks



Move A to C
Move A to B
Move C to B

Tower of Hanoi - 3

- Moves when there are 3 disks



Move A to C
Move A to B
Move C to B
Move A to C

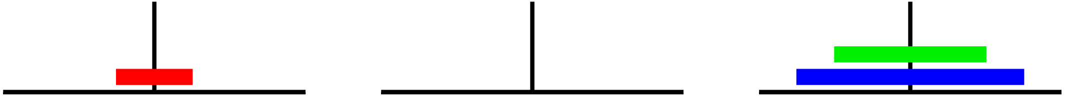
Tower of Hanoi - 3

- Moves when there are 3 disks

Move A to C
Move A to B
Move C to B
Move A to C
Move B to A

Tower of Hanoi - 3

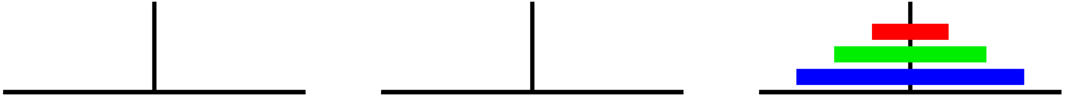
- Moves when there are 3 disks



Move A to C
Move A to B
Move C to B
Move A to C
Move B to A
Move B to C

Tower of Hanoi - 3

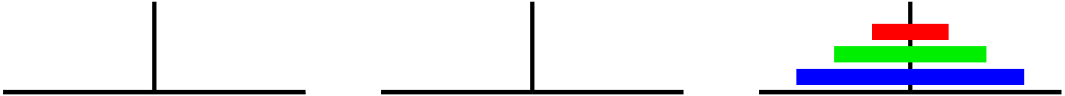
- Moves when there are 3 disks



Move A to C
Move A to B
Move C to B
Move A to C
Move B to A
Move B to C
Move A to C

Tower of Hanoi - 3

- Moves when there are 3 disks



Move A to C
Move A to B
Move C to B
Move A to C
Move B to A
Move B to C
Move A to C

```
toh(n, src, dest, aux)
    if n = 1 then move disk from src to dest
    toh(n − 1, src, aux, dest)
    move n-th disk from src to dest
    toh(n − 1, aux, dest, src)
```

Asymptotic analysis

- Run time depends on - machine specification (instruction set, memory, cache, cpu speed, etc.), algorithms, input size
- Need a framework for machine independent comparison
- Suppress constant factors and lower order terms

Asymptotic analysis

- Run time depends on - machine specification (instruction set, memory, cache, cpu speed, etc.), algorithms, input size
- Need a framework for machine independent comparison
- Suppress constant factors and lower order terms

```
max(array, size):
    m = array[1]
    for i = 2 to size:
        if array[i] > m:
            m = array[i]
    return m
```

Asymptotic analysis

- Run time depends on - machine specification (instruction set, memory, cache, cpu speed, etc.), algorithms, input size
- Need a framework for machine independent comparison
- Suppress constant factors and lower order terms

```
max(array, size):
    m = array[1]
    for i = 2 to size:
        if array[i] > m:
            m = array[i]
    return m
```

```
max:
   lw $v0, 0($a0)
   li $t0, 1
loop: bge $t0, $a1, end
   sll $t1, $t0, 2
   addu $t2, $a0, $t1
   lw $t3, 0($t2)
   ble $t3, $v0, updt
   move $v0, $t3
updt: addiu $t0, $t0, 1
   j loop
end: jr $ra
```

Asymptotic notations

- Big-O
  - $T(n) = O(f(n))$ if and only if $T(n)$ is eventually bounded above by a constant multiple of $f(n)$

Asymptotic notations

- Big-O
  - $T(n) = O(f(n))$ if and only if $T(n)$ is eventually bounded above by a constant multiple of $f(n)$
  - $T(n) = O(f(n))$ if and only if there exist positive constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$

Asymptotic notations

- Big-O
  - $T(n) = O(f(n))$ if and only if $T(n)$ is eventually bounded above by a constant multiple of $f(n)$
  - $T(n) = O(f(n))$ if and only if there exist positive constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$
  - Example: $\{n, 5n, 10n + \lg n, n + 10000\sqrt{n}\} \in O(n)$

Asymptotic notations

- Big-O
  - $T(n) = O(f(n))$ if and only if $T(n)$ is eventually bounded above by a constant multiple of $f(n)$
  - $T(n) = O(f(n))$ if and only if there exist positive constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$
  - Example: $\{n, 5n, 10n + \lg n, n + 10000\sqrt{n}\} \in O(n)$
  - Example: $\{n^{1.2}, n \lg n, 0.000001n^2\} \notin O(n)$

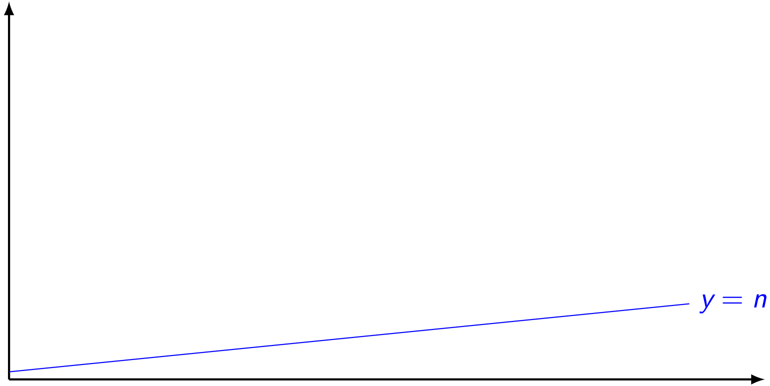
Asymptotic notations

- Big-O
  - $T(n) = O(f(n))$ if and only if $T(n)$ is eventually bounded above by a constant multiple of $f(n)$
  - $T(n) = O(f(n))$ if and only if there exist positive constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$
  - Example: $\{n, 5n, 10n + \lg n, n + 10000\sqrt{n}\} \in O(n)$
  - Example: $\{n^{1.2}, n \lg n, 0.000001n^2\} \notin O(n)$
  - Example: $\{n, 5n, 10n + \lg n, n + 10000\sqrt{n}, n^2 + 10000 \lg n, n^{\lg 4}, n^{1.99}\} \in O(n^2)$

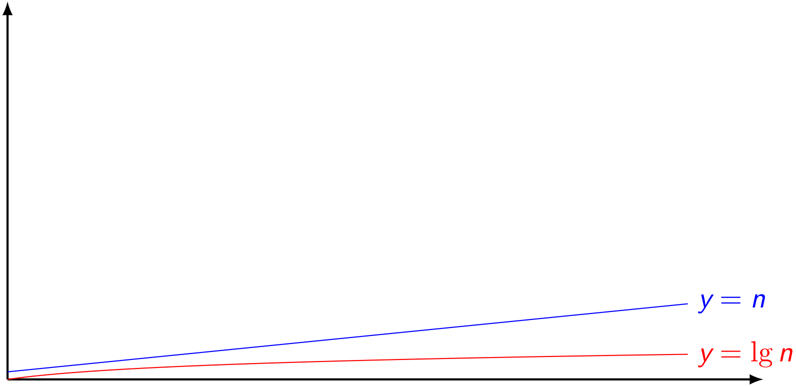
Asymptotic notations

- Big-O
  - $T(n) = O(f(n))$ if and only if $T(n)$ is eventually bounded above by a constant multiple of $f(n)$
  - $T(n) = O(f(n))$ if and only if there exist positive constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$
  - Example: $\{n, 5n, 10n + \lg n, n + 10000\sqrt{n}\} \in O(n)$
  - Example: $\{n^{1.2}, n \lg n, 0.000001n^2\} \notin O(n)$
  - Example: $\{n, 5n, 10n + \lg n, n + 10000\sqrt{n}, n^2 + 10000 \lg n, n^{\lg 4}, n^{1.99}\} \in O(n^2)$
  - Example: $\{2^n, n^2 \lg n, n^{2.01}\} \notin O(n^2)$

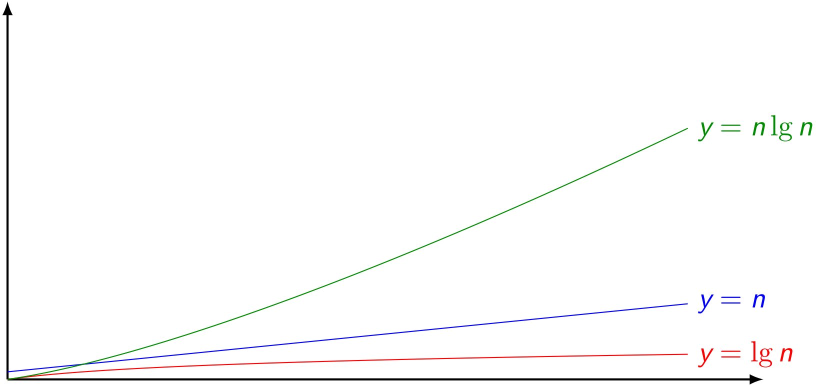
Growth of functions



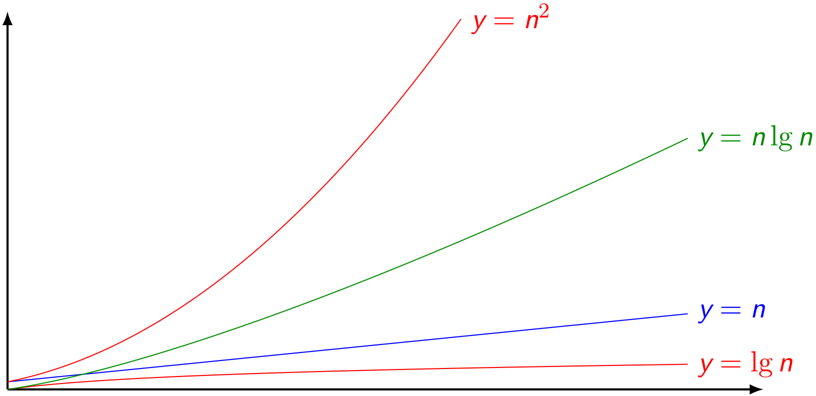
Growth of functions



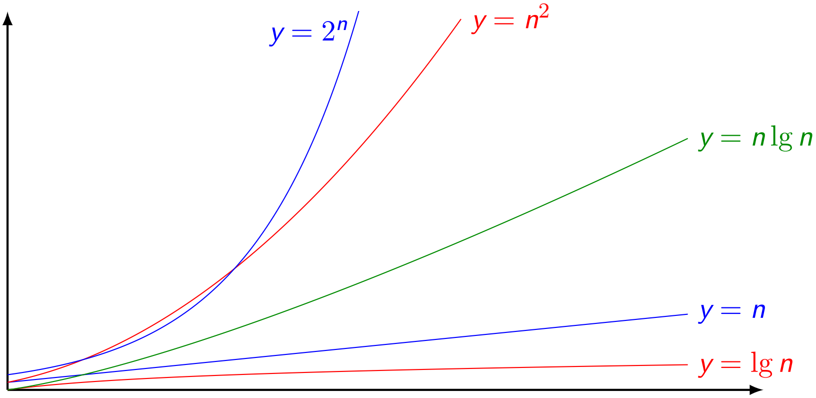
Growth of functions



Growth of functions



Growth of functions



Big-Omega

- Ω
  - $T(n) = \Omega(f(n))$ if and only if $T(n)$ is eventually bounded below by a constant multiple of $f(n)$

Big-Omega

- Ω
  - $T(n) = \Omega(f(n))$ if and only if $T(n)$ is eventually bounded below by a constant multiple of $f(n)$
  - $T(n) = \Omega(f(n))$ if and only if there exist positive constants c and n_0 such that $T(n) \geq c \cdot f(n)$ for all $n \geq n_0$

Big-Omega

- Ω
  - $T(n) = \Omega(f(n))$ if and only if $T(n)$ is eventually bounded below by a constant multiple of $f(n)$
  - $T(n) = \Omega(f(n))$ if and only if there exist positive constants c and n_0 such that $T(n) \geq c \cdot f(n)$ for all $n \geq n_0$
  - Example: $\{n, 5n, 10n + \lg n, n + 10000\sqrt{n}\} \in \Omega(n)$

Big-Omega

- Ω
  - $T(n) = \Omega(f(n))$ if and only if $T(n)$ is eventually bounded below by a constant multiple of $f(n)$
  - $T(n) = \Omega(f(n))$ if and only if there exist positive constants c and n_0 such that $T(n) \geq c \cdot f(n)$ for all $n \geq n_0$
  - Example: $\{n, 5n, 10n + \lg n, n + 10000\sqrt{n}\} \in \Omega(n)$
  - Example: $\{n^{1.2}, n \lg n, 0.000001n^2\} \in \Omega(n)$

Big-Omega

- Ω
  - $T(n) = \Omega(f(n))$ if and only if $T(n)$ is eventually bounded below by a constant multiple of $f(n)$
  - $T(n) = \Omega(f(n))$ if and only if there exist positive constants c and n_0 such that $T(n) \geq c \cdot f(n)$ for all $n \geq n_0$
  - Example: $\{n, 5n, 10n + \lg n, n + 10000\sqrt{n}\} \in \Omega(n)$
  - Example: $\{n^{1.2}, n \lg n, 0.000001n^2\} \in \Omega(n)$
  - Example: $n, 5n, 10n + \lg n, n + 10000\sqrt{n}, n^2 + 10000 \lg n, n^{\lg 4}, n^{1.99}, 2^n, n^2 \lg n, n^{2.01}$ — $\Omega(n), \Omega(n^2)$?

Big-Theta

- Θ
  - $T(n) = \Theta(f(n))$ means $T(n) = \Omega(f(n))$ and $T(n) = O(f(n))$

Big-Theta

- Θ
  - $T(n) = \Theta(f(n))$ means $T(n) = \Omega(f(n))$ and $T(n) = O(f(n))$
  - $T(n) = \Theta(f(n))$ if and only if there exist positive constants c_1, c_2 and n_0 such that $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$

Big-Theta

- Θ
  - $T(n) = \Theta(f(n))$ means $T(n) = \Omega(f(n))$ and $T(n) = O(f(n))$
  - $T(n) = \Theta(f(n))$ if and only if there exist positive constants c_1, c_2 and n_0 such that $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$
  - $T(n) = \frac{1}{2}n^2 + 3n$ — $O(n), \Omega(n), \Theta(n^2), O(n^3)$?

Solving recurrences

- Substitution method: Guess the solution and prove by mathematical induction
  - $T(n) = 2T(\lfloor n/2 \rfloor) + O(n)$ — Try with $O(n^2)$, $O(n \lg n)$, $O(n)$

Solving recurrences

- Substitution method: Guess the solution and prove by mathematical induction
  - $T(n) = 2T(\lfloor n/2 \rfloor) + O(n)$ — Try with $O(n^2)$, $O(n \lg n)$, $O(n)$
- Recursion tree
  - $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$

Master theorem

- If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$, and $d \geq 0$ then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Excercise

- Solve the recurrence relation:
  - $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \lg n$

Excercise

- Solve the recurrence relation:
  - $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \lg n$
  - $T(n) = T(n/3) + T(2n/3) + n$

Polynomial evaluation

- Given a sequence of real numbers $a_n, a_{n-1}, \ldots, a_1, a_0$, and a real number x, compute the value of the polynomial $P_n(x) = a_n x^n + a_{n-1} x^{n-I} + \ldots + a_1 x + a_0$.
- Recursive formulation:

Polynomial evaluation

- Given a sequence of real numbers $a_n, a_{n-1}, \ldots, a_1, a_0$, and a real number x, compute the value of the polynomial $P_n(x) = a_n x^n + a_{n-1} x^{n-l} + \ldots + a_1 x + a_0$.
- Recursive formulation:
  - $P_n(x) = P_{n-1}(x) + a_n x^n$, $[P_{n-1}(x) = a_{n-1} x^{n-1} + \ldots + a_0]$

Polynomial evaluation

- Given a sequence of real numbers $a_n, a_{n-1}, \ldots, a_1, a_0$, and a real number x, compute the value of the polynomial $P_n(x) = a_n x^n + a_{n-1} x^{n-l} + \ldots + a_1 x + a_0$.
- Recursive formulation:
  - $P_n(x) = P_{n-1}(x) + a_n x^n$, $[P_{n-1}(x) = a_{n-1} x^{n-1} + \ldots + a_0]$
  - $P_n(x) = x P'_{n-1}(x) + a_0$, $[p'_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} + \ldots + a_1]$

Polynomial evaluation

- Given a sequence of real numbers $a_n, a_{n-1}, \ldots, a_1, a_0$, and a real number x, compute the value of the polynomial $P_n(x) = a_n x^n + a_{n-1}x^{n-l} + \ldots + a_1 x + a_0$.
- Recursive formulation:
  - $P_n(x) = P_{n-1}(x) + a_n x^n$, $[P_{n-1}(x) = a_{n-1}x^{n-1} + \ldots + a_0]$
  - $P_n(x) = xP'_{n-1}(x) + a_0$, $[p'_{n-1}(x) = a_n x^{n-1} + a_{n-1}x^{n-2} + \ldots + a_1]$
- Time complexity?

Master theorem: Examples

- $T(n) = 3\,T(n/2) + n^2$

Master theorem: Examples

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

Master theorem: Examples

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

Master theorem: Examples

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

Master theorem: Examples

- $T(n) = 3T(n/2) + n^2$
- $T(n) = 4T(n/2) + n^2$
- $T(n) = T(n/2) + 2^n$
- $T(n) = 2^n T(n/2) + n^n$
- $T(n) = 16T(n/4) + n$

Master theorem: Examples

- $T(n) = 3T(n/2) + n^2$
- $T(n) = 4T(n/2) + n^2$
- $T(n) = T(n/2) + 2^n$
- $T(n) = 2^n T(n/2) + n^n$
- $T(n) = 16T(n/4) + n$
- $T(n) = 2T(n/2) + n \lg n$

Master theorem: Examples

- $T(n) = 3\,T(n/2) + n^2$
- $T(n) = 4\,T(n/2) + n^2$
- $T(n) = T(n/2) + 2^n$
- $T(n) = 2^n\,T(n/2) + n^n$
- $T(n) = 16\,T(n/4) + n$
- $T(n) = 2\,T(n/2) + n \lg n$
- $T(n) = 16\,T(n/4) + n!$

Master theorem: Examples

- $T(n) = 3T(n/2) + n^2$
- $T(n) = 4T(n/2) + n^2$
- $T(n) = T(n/2) + 2^n$
- $T(n) = 2^n T(n/2) + n^n$
- $T(n) = 16T(n/4) + n$
- $T(n) = 2T(n/2) + n \lg n$
- $T(n) = 16T(n/4) + n!$
- $T(n) = \sqrt{2}\,T(n/2) + \log n$

Master theorem: Examples

- $T(n) = 3\,T(n/2) + n^2$
- $T(n) = 4\,T(n/2) + n^2$
- $T(n) = T(n/2) + 2^n$
- $T(n) = 2^n\,T(n/2) + n^n$
- $T(n) = 16\,T(n/4) + n$
- $T(n) = 2\,T(n/2) + n\lg n$
- $T(n) = 16\,T(n/4) + n!$
- $T(n) = \sqrt{2}\,T(n/2) + \log n$
- $T(n) = 3\,T(n/3) + \sqrt{n}$

Master theorem: Examples

- $T(n) = 3T(n/2) + n^2$
- $T(n) = 4T(n/2) + n^2$
- $T(n) = T(n/2) + 2^n$
- $T(n) = 2^n T(n/2) + n^n$
- $T(n) = 16T(n/4) + n$
- $T(n) = 2T(n/2) + n \lg n$
- $T(n) = 16T(n/4) + n!$
- $T(n) = \sqrt{2}T(n/2) + \log n$
- $T(n) = 3T(n/3) + \sqrt{n}$
- $T(n) = 4T(n/2) + cn$

Master theorem: Examples

- $T(n) = 3T(n/2) + n^2$
- $T(n) = 4T(n/2) + n^2$
- $T(n) = T(n/2) + 2^n$
- $T(n) = 2^n T(n/2) + n^n$
- $T(n) = 16T(n/4) + n$
- $T(n) = 2T(n/2) + n \lg n$
- $T(n) = 16T(n/4) + n!$
- $T(n) = \sqrt{2}\,T(n/2) + \log n$
- $T(n) = 3T(n/3) + \sqrt{n}$
- $T(n) = 4T(n/2) + cn$
- $T(n) = 64T(n/8) - n^2 \log n$

Master theorem: Examples

- $T(n) = 3T(n/2) + n^2$
- $T(n) = 4T(n/2) + n^2$
- $T(n) = T(n/2) + 2^n$
- $T(n) = 2^n T(n/2) + n^n$
- $T(n) = 16T(n/4) + n$
- $T(n) = 2T(n/2) + n \lg n$
- $T(n) = 16T(n/4) + n!$
- $T(n) = \sqrt{2}T(n/2) + \log n$
- $T(n) = 3T(n/3) + \sqrt{n}$
- $T(n) = 4T(n/2) + cn$
- $T(n) = 64T(n/8) - n^2 \log n$
- $T(n) = 2T(n/4) + n^{0.51}$

Thank you!