TOPIC 1: ALGORITHM ANALYSIS

PERFORMANCE OF ALGORITHMS

	Empirically	Analytically
How	Execute the implementation multiple times	Analyzing the data flow of the algorithm/pseudocode
Pros	Representative of real world performance Useful when algorithm is not available	Independent of implementation details, programming language, machine, etc. Asymptotically true
Cons	Requires time to test Results dependent on implementation details, programming language, machine, etc.	Does not take into account I/O or memory operations Not always reflective of practical settings (such as large constants)

What are methods to compare algorithms ? How do we measure them ?

GENERAL RULES FOR COMPUTING PERFORMANCE

Rule 1 Consecutive Statements

- Add the time of each statement

Rule 2 Loops

- The running time of the statements inside the loop times number of iterations

Rule 3 Nested Loops

- Analyze from the inner most loop
- The total running time inside a group of nested loops is the product of time per loop

Rule 4 If/Else

- Take the section with the larger time

EXAMPLES

-
  ```
  for ( int i = 0; i < n; i ++)
      for ( int j = 0; j < n; j ++)
          sum++;
  ```

-
  ```
  for ( int i = 0; i < n; i ++)
      for ( int j = 0; j < n*n; j ++)
          sum ++;
  ```

-
  ```
  for ( int i = 0; i < n; i ++)
      for ( int j = 0; j < i; j ++)
          sum++;
  ```

-
  ```
  for ( int i = 0; i < n; i ++)
      for ( int j = 0; j < i; j ++)
          for (int k = 0; k < j; k++)
              sum++;
  ```

-
  ```
  If(n <=1)
        return 1
      Else
        return n*fact(n-1)
  ```

ASYMPTOTIC COMPUTATIONAL COMPLEXITY

When computing the complexity analytically, we should factor out the effect of implementation details

We assume if the size of the problem is large enough, then the variation in time due to the implementation details can be folded into constants

- $C_1 f_1(n)+C_2 f_2(n)+\ldots.+C_x f_x(n)$

As the value of n increases n->∝, the higher value function dominates, i.e. n^2 grows faster than n

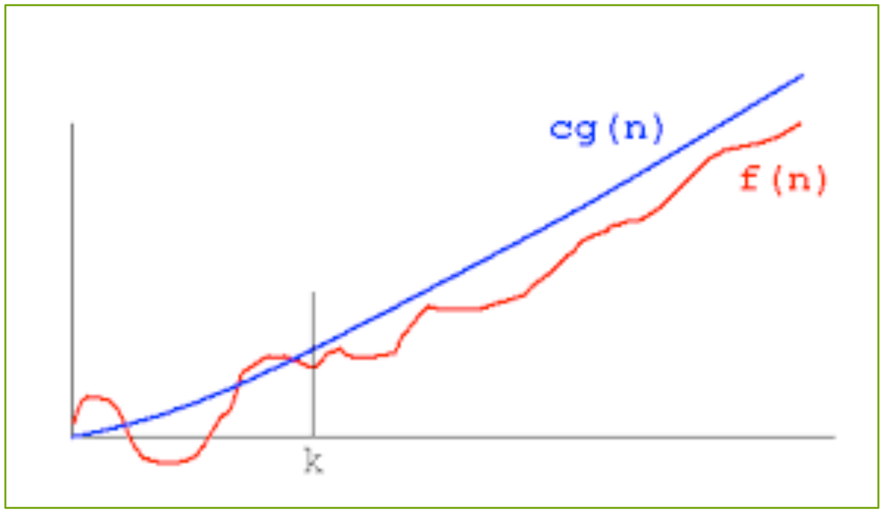
Asymptotic means going to very high values

Thus when computing the complexity asymptotically, we can ignore the constants and the lower valued functions

COMPUTATIONAL COMPLEXITY

T(N) time to execute a program with input of size N

	Big O (Upper Bound)	**Omega (Lower Bound)**	**Theta (Upper and Lower Bound)**
Definition (remember this)			
Meaning (understand this)			



Actual time f(n)
Big O complexity O(g(n))
Image from quora.com

Common Functions used for Complexity

C constant

Log(n) logarithmic

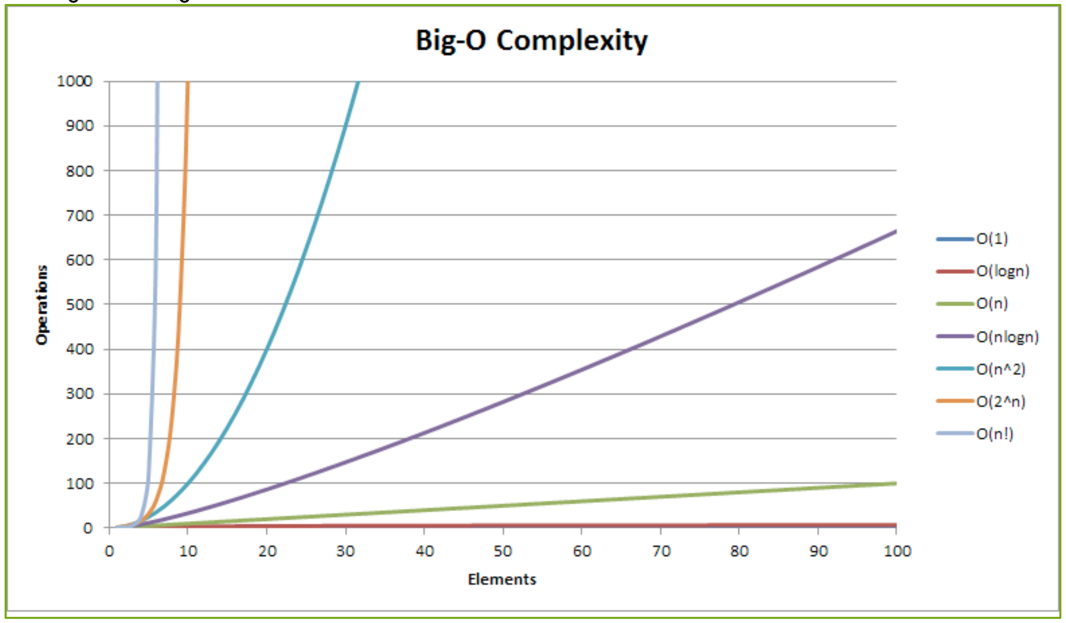
N linear

N log(N)

N^k polynomial (k=2 Quadratic, k=3 Cubic)

C^N Exponential

Big-O complexity of different functions
Image from bigocheatsheet.com



COMPUTING COMPLEXITY IN BIG-OH

Given the function of the running time it is generally easy to determine the Big Oh function

How do we determine the n0 (initial point) and c (constant)

- $n0 \geq 0$ (nonnegative integer)
- $c \geq 1$ (positive integer)

By appropriate algebraic manipulation show that if $n \geq n0$ then $T(n) \leq cf(n)$ for the n0 and c selected

EXAMPLE

Let $T(n)=(n+1)^2$

$T(n)=n^2+2n+1$

Rule of thumb: Add the constants to get c. Set n0 to the constant of n^0

- Here n0=1and c=4 we can state that $T(n)=O(n^2)$

Other c and n0 can also apply

- Lower n0 allows us to start from a lower cut-off
- Higher C allows for a larger envelope

How about n0=3and c=2 ?

How about selecting n0=0?

We can use induction to prove these statements

PROOF BY INDUCTION

To show the statement is true for all n≥k

- Basis Case: Show that the statement is true for the base value of k.
- Inductive Hypothesis: Assume that the statement S(n) will be true for all consecutive n ≤k
- Inductive Step: Show that if the statement is true for n then it is true for n+1; S(n)=> S(n+1)

EXAMPLES

Prove that summation of the powers of 2 from 0 to n is 2^{n+1}-1 for n≥0

For any positive integer n, 6^n-1 is divisible by 5

Given that r^2=r+1. Show that F(i) ≥ r $^{n-2,}$ where F(i) is the ith element in the Fibonacci sequence given in the previous slide

FIND THE COMPLEXITY

$T1(n)=5n+6n^2$

$T2(n)=4n\log(n)+5(n)$

$T3(n)=T1(n)+T2(n)$

$T4(n)=n^4+n^2+1$

$T5(n)=2^n$

What are the n0 and c terms for each case ?

BOUNDS THAT ARE NOT ASYMPTOTICALLY TIGHT

Asymptotically tight bounds is the best function that can fit

- For T(n)=n2+2n+1; O(n3) is an upper bound, as is O(n2)
- However, O(n2) is the more close fit so it is asymptotically tight
- O(n3) is NOT asymptotically tight

Not asymptotically tight upper bound (small o)

Note use of any positive constant as opposed to there exists a positive constant

- *T(n)=o (g(n)): for any positive constants c>0 such that 0≤T(n)< cg(n), when n≥n0*
  - T(n) becomes insignificant relative to g(n) as n approaches infinity
- *T(n)=ω (g(n)): for any positive constants c>0 such that 0 ≤cg(n)< T(n), when n≥n0*
  - T(n) becomes arbitrarily large relative to g(n) as n approaches infinity

RECURSIVE FUNCTIONS

Time for recursive functions can be written using a recurrence relation.

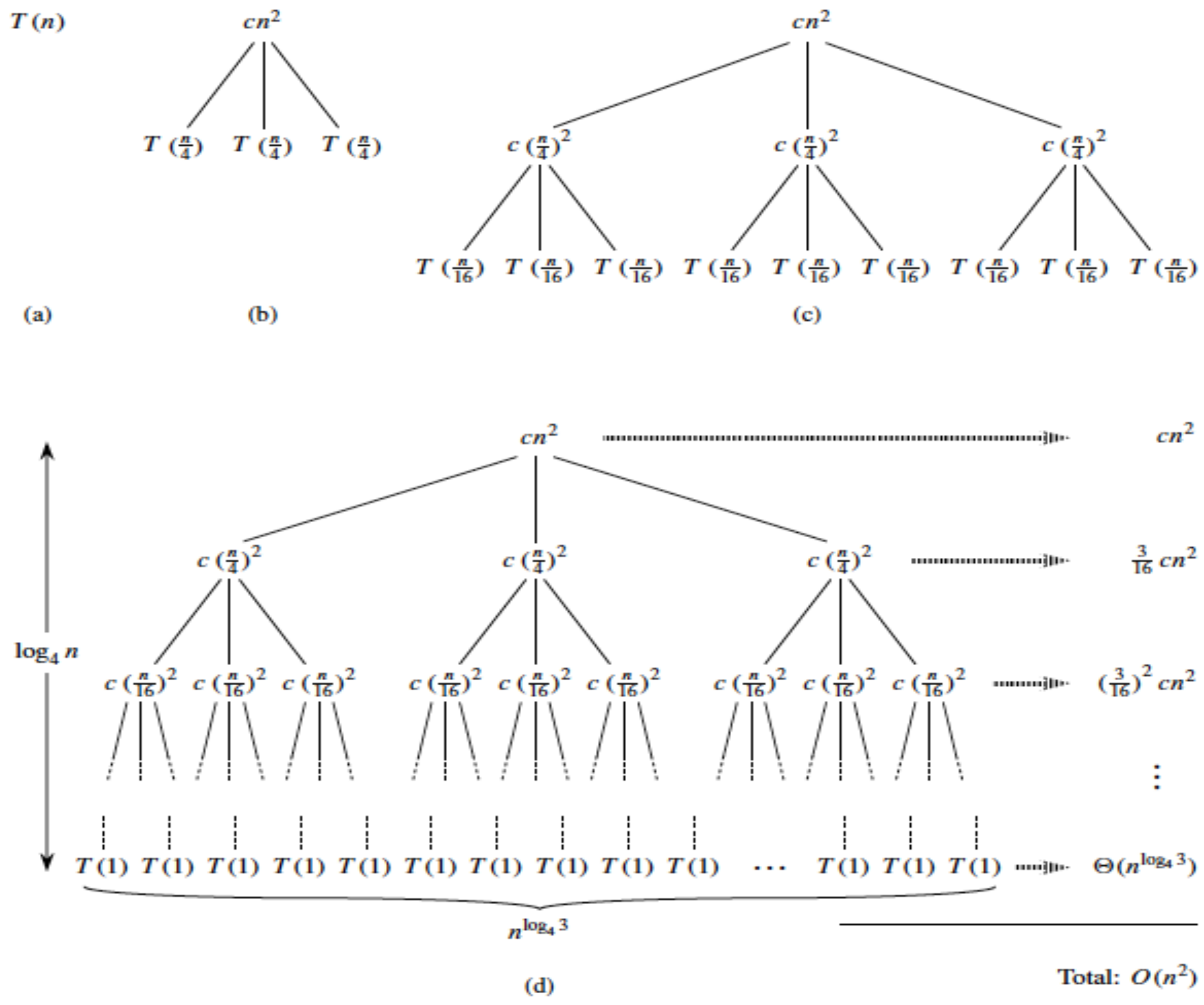
Recall for the factorial

- Constant time for multiplication plus time for factorial of n-1
- $T(n)=C+T(n-1)$

SOLVING RECURSIVE FUNCTIONS

- To compute the complexity for recursive functions
- $T(n)=aT(n/b)+f(n)$
- Methods
  - Iterative Method (Expand the form and express it as a summation of terms) –use recursion trees
  - Master's Theorem—cookbook method

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



$T(n)$
cn^2
$T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$
$c(\frac{n}{4})^2$
$T(\frac{n}{16})$
(a)
(b)
(c)
$\log_4 n$
$c(\frac{n}{16})^2$
$T(1)$
$\cdots$
$\frac{3}{16}cn^2$
$(\frac{3}{16})^2 cn^2$
$\Theta(n^{\log_4 3})$
$n^{\log_4 3}$
Total: $O(n^2)$
(d)

ITERATION METHOD EXAMPLE

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

$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n-1} cn^2 + \Theta(n^{\log_4 3}) \\ &= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{(3/16)^{\log_4 n} - 1}{(3/16) - 1} cn^2 + \Theta(n^{\log_4 3}) . \end{aligned}$$

MASTER'S THEOREM

$T(n)=aT(n/b)+f(n)$ $a>0$; $b>1$

Case1: If $f(n)$ is $O(n^{\log_b a-\varepsilon})$; where $\varepsilon>0$ then $T(n)$ is $\theta(n^{\log_b a})$.

If $f(n)$ is smaller $O(n^{\log_b a})$ then $T(n)=\theta(n^{\log_b a})$.

- Example: $T(n)=2T(n/2)+1$

Case 2: If $f(n)$ is $O(n^{\log_b a})$; then $T(n)$ is $\theta(n^{\log_b a} \lg n)$.

If $f(n)$ is equal to $O(n^{\log_b a})$ then $T(n)=\theta(n^{\log_b a} \lg_b n)$.

- Example: $T(n)=2T(n/2)+n$
- General Case: $f(n)$ is $O(n^{\log_b a} (\lg_b n)^p)$; $p>-1$ $T(n)=O(n^{\log_b a} (\lg_b n)^{p+1})$;
- $f(n)$ is $O(n^{\log_b a} (\lg_b n)^p)$; $p==-1$ $T(n)=O(n^{\log_b a} (\lg_b \lg_b n))$;

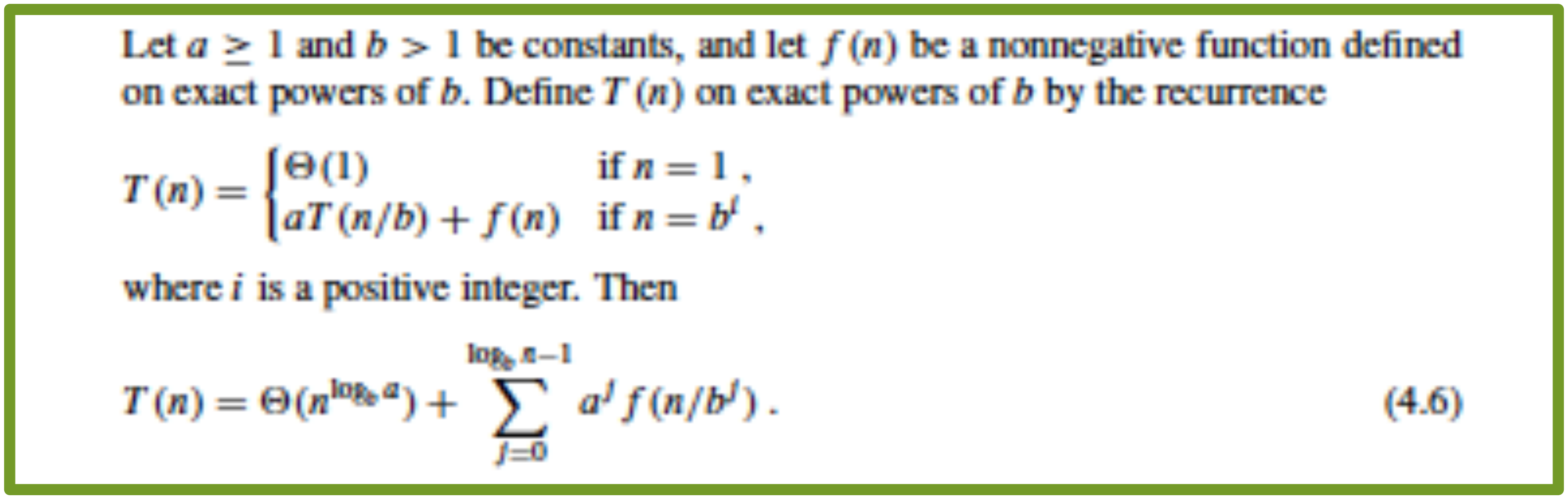
Case 3: If $f(n)$ is $\Omega(n^{\log_b a+\varepsilon})$; where $\varepsilon>0$ and if $af(n/b)\leq cf(n)$ for $c<1$ and sufficiently large n (regularity condition) then $T(n)$ is $\theta(f(n))$

If $f(n)$ is greater than $O(n^{\log_b a})$ AND $f(n)$ grows faster than $af(n/b)$ then $T(n)$ is $\theta(f(n))$

- Example: $T(n)=2T(n/2)+n^2$

PROOF OF MASTER THEOREM

We will only prove when n is an exact power of b

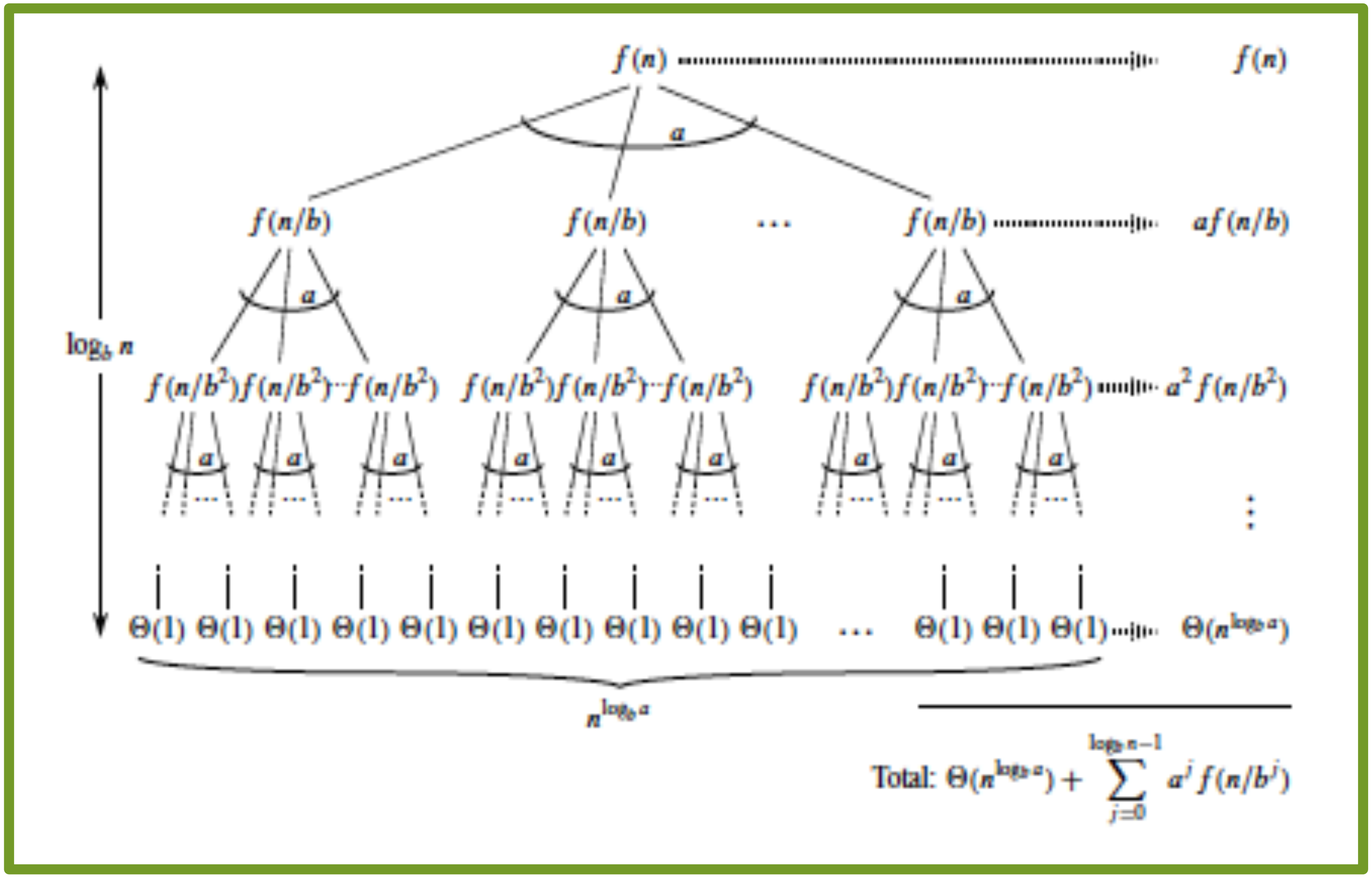


Let $a \geq 1$ and $b > 1$ be constants, and let $f(n)$ be a nonnegative function defined on exact powers of b. Define $T(n)$ on exact powers of b by the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ aT(n/b) + f(n) & \text{if } n = b^i , \end{cases}$$

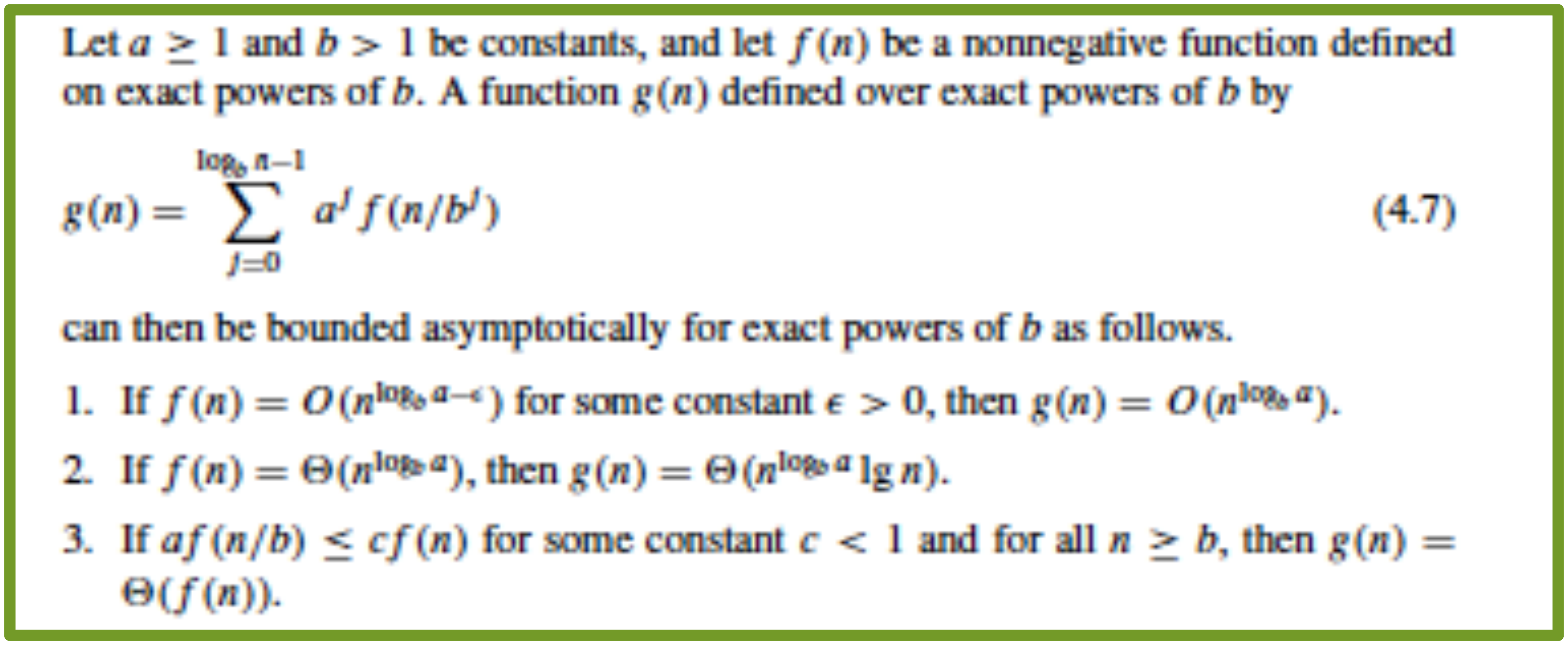
where i is a positive integer. Then

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) . \qquad (4.6)$$



$f(n)$
a
$f(n/b)$ $f(n/b)$ $\cdots$ $f(n/b)$
$af(n/b)$
$\log_b n$
$f(n/b^2)\,f(n/b^2)\cdots f(n/b^2)$
$a^2 f(n/b^2)$
$\Theta(1)$
$\Theta(n^{\log_b a})$
$n^{\log_b a}$
Total: $\Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$

PROOF OF MASTER THEOREM, CONTINUED



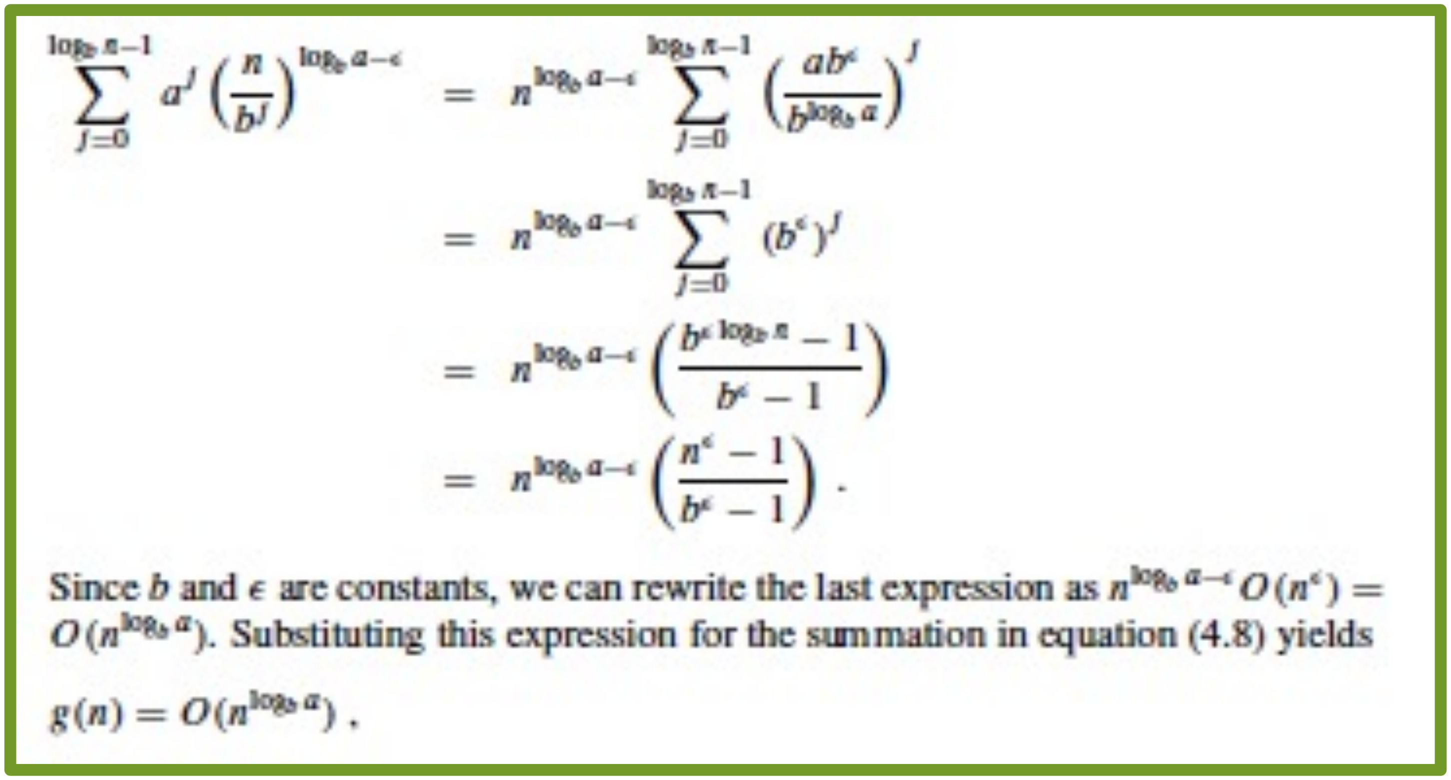
Let $a \geq 1$ and $b > 1$ be constants, and let $f(n)$ be a nonnegative function defined on exact powers of b. A function $g(n)$ defined over exact powers of b by

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) \qquad (4.7)$$

can then be bounded asymptotically for exact powers of b as follows.

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $g(n) = O(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $g(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $af(n/b) \leq cf(n)$ for some constant $c < 1$ and for all $n \geq b$, then $g(n) = \Theta(f(n))$.

CASE 1

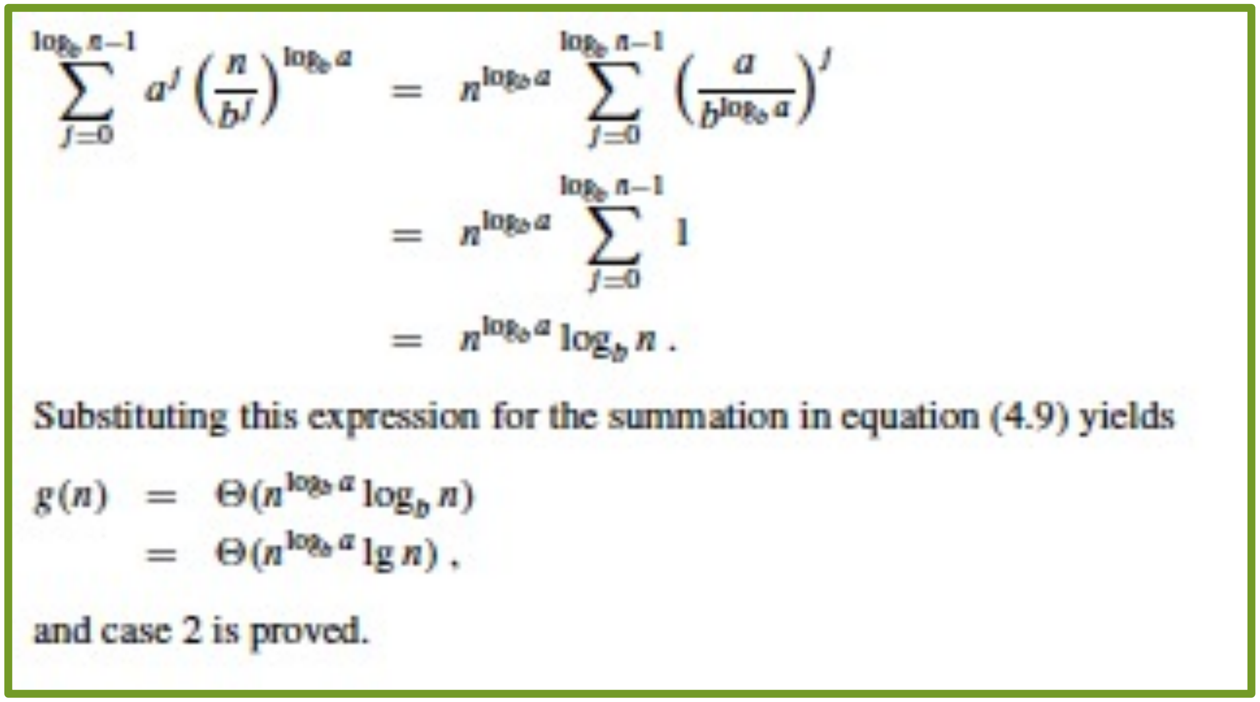


$$\begin{aligned}
\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a-\epsilon} &= n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} \left(\frac{ab^\epsilon}{b^{\log_b a}}\right)^j \\
&= n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} (b^\epsilon)^j \\
&= n^{\log_b a-\epsilon} \left(\frac{b^{\epsilon \log_b n}-1}{b^\epsilon-1}\right) \\
&= n^{\log_b a-\epsilon} \left(\frac{n^\epsilon-1}{b^\epsilon-1}\right) .
\end{aligned}$$

Since b and ϵ are constants, we can rewrite the last expression as $n^{\log_b a-\epsilon} O(n^\epsilon) = O(n^{\log_b a})$. Substituting this expression for the summation in equation (4.8) yields

$$g(n) = O(n^{\log_b a}) .$$

CASE 2



$$
\begin{aligned}
\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} &= n^{\log_b a} \sum_{j=0}^{\log_b n-1} \left(\frac{a}{b^{\log_b a}}\right)^j \\
&= n^{\log_b a} \sum_{j=0}^{\log_b n-1} 1 \\
&= n^{\log_b a} \log_b n \,.
\end{aligned}
$$

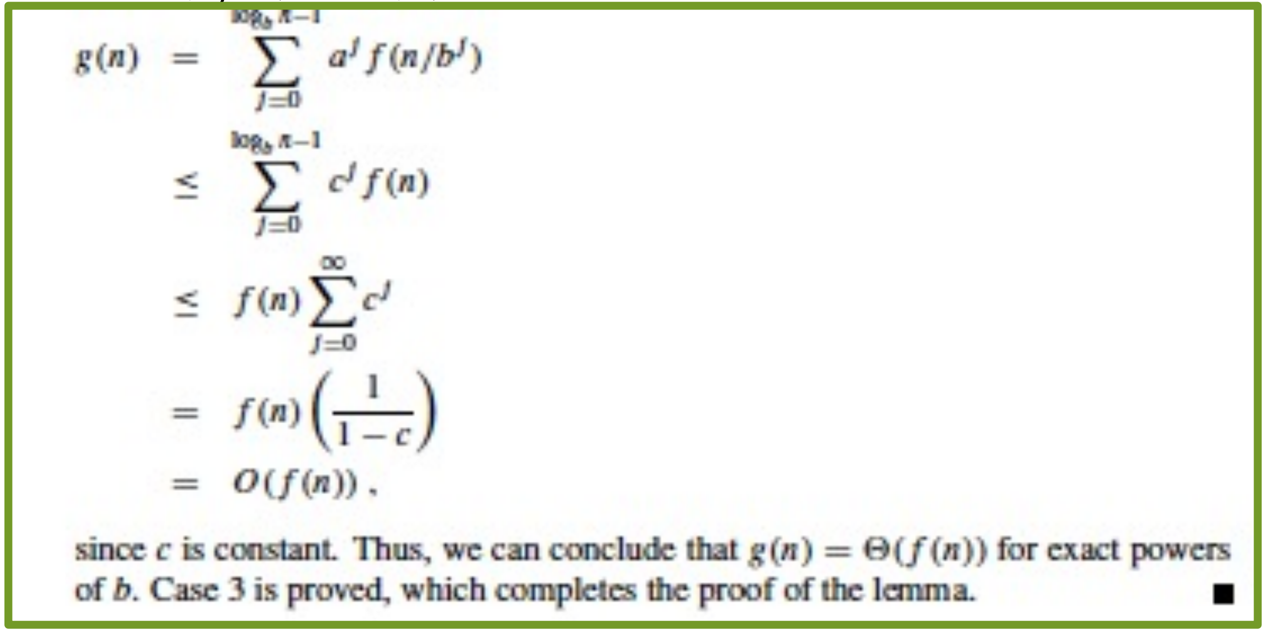
Substituting this expression for the summation in equation (4.9) yields

$$
\begin{aligned}
g(n) &= \Theta(n^{\log_b a} \log_b n) \\
&= \Theta(n^{\log_b a} \lg n) \,,
\end{aligned}
$$

and case 2 is proved.

CASE 3

Given af(n/b) ≤ cf(n), for some c <1 and n≥ b



$$
\begin{aligned}
g(n) &= \sum_{j=0}^{\log_b n-1} a^j f(n/b^j) \\
&\le \sum_{j=0}^{\log_b n-1} c^j f(n) \\
&\le f(n) \sum_{j=0}^{\infty} c^j \\
&= f(n)\left(\frac{1}{1-c}\right) \\
&= O(f(n)) ,
\end{aligned}
$$

since c is constant. Thus, we can conclude that $g(n) = \Theta(f(n))$ for exact powers of b. Case 3 is proved, which completes the proof of the lemma. ■

EXTENSION FOR CASE 2

- f(n)is $O(n^{\log_b a} (\lg_b n)^p)$;
  - (i) $p>-1$ $T(n)= O(n^{\log_b a} (\lg_b n)^{p+1})$;
  - (ii) $p==-1$ $T(n)= O(n^{\log_b a} (\lg_b \lg_b n))$;
  - (iii) $p<-1$ $T(n)= O(n^{\log_b a})$;

PROOFS FOR EXTENSION

1. $T(n)=aT(n/b)+f(n)$

$T(n) = f(n) + af(n/b) + a^2 f(n/b^2) + \ldots + a^{(k-1)} f(n/b^{(k-1)}) + a^k T(1).$

$f(n)=O(n^{\log_b a}(\lg_b n)^p)= cn^E(\lg_b n)^p;\ E=\log_b a$

2. $T(n) \approx cn^E(\log_b n)^p + ac(n/b)^E(\log_b(n/b))^p + a^2c(n/b^2)^E(\log_b(n/b^2))^p + \ldots + a^kT(1)$

$n = b^k$, => $\log_b(n/b^i) = \log_b b^{(k-i)} = k-i$

$a^k=n^E.$ $a=b^E.$ $a^i / b^{iE} = 1$

3. $T(n) \approx cn^E k^p + cn^E(k-1)^p + cn^E(k-2)^p + \ldots + cn^E 1^p + T(1)n^E$

$= cn^E \sum_{i=1}^{n} i^\alpha + T(1)n^E$

$\sum_{i=1}^{n} i^\alpha \approx \frac{n^{\alpha+1}}{(\alpha+1)}$ $\boldsymbol{\alpha} > -1$

$\sum_{i=1}^{n} i^\alpha \approx \ln(n) + C$ $\boldsymbol{\alpha} = -1$

(harmonic series)

$\sum_{i=1}^{n} i^\alpha = C;$ $\boldsymbol{\alpha} < -1$

MASTER'S THEOREM CANNOT BE APPLIED WHEN…

Recursion Relation: $T(n)=aT(n/b)+f(n)$

a is not a constant

- $T(n)=2^nT(n/2)+n^2$

$a<1$

- $T(n)=.5T(n/2)+n^2$

$f(n)> n^{\log_b a}$ but regularity condition not violated; there is no $c< 1$ for which $af(n/b)<cf(n)$

- $T(n)=T(n/2)+n(2-\cos n)$

f(n) is not positive

- $T(n)=2T(n/2)-n^2$

EXAMPLES

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

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

$T(n)=T(n/2)+2^{n\surd\surd}$

$T(n)=16T(n/4)+n$

$T(n)=2T(n/2)+n\log n$

$T(n)=3T(n/3)+\sqrt{n}$

$T(n)=2T(n/2)+n/\log n$

WHY PARALLEL PROGRAMMING

Faster and Larger Computations

Solve computing-intensive problems faster

Solve larger problems in same amount of time

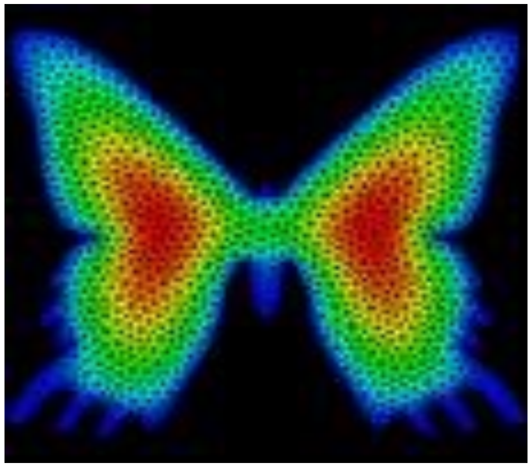
Reduce design time

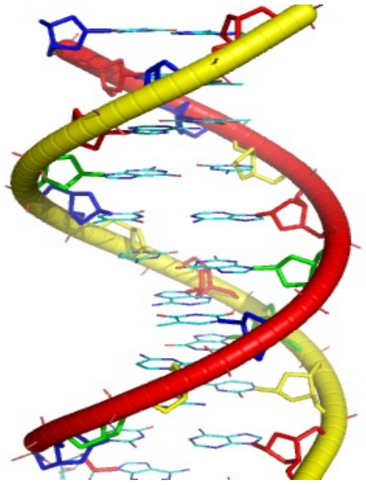
Improve solution precision

Many divide and conquer problems can be expressed as parallel programs

PARALLEL COMPUTING

Using many computers all together to solve large problems, faster

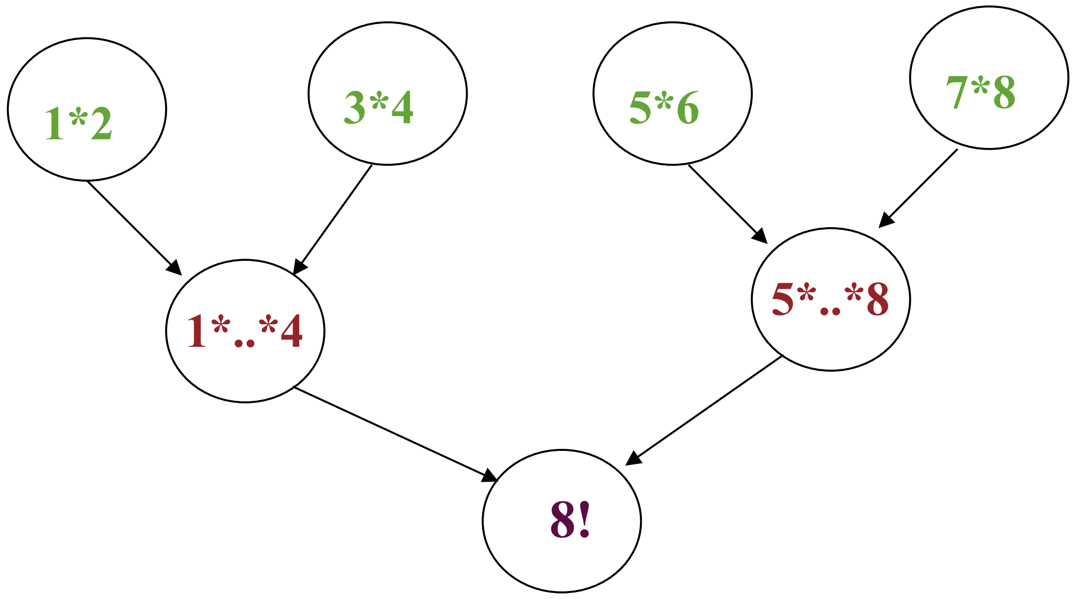




EXAMPLE

Find n!

- Divide the numbers into k processors
- Compute sequential products on elements in each processor
- Multiply the results from each processor

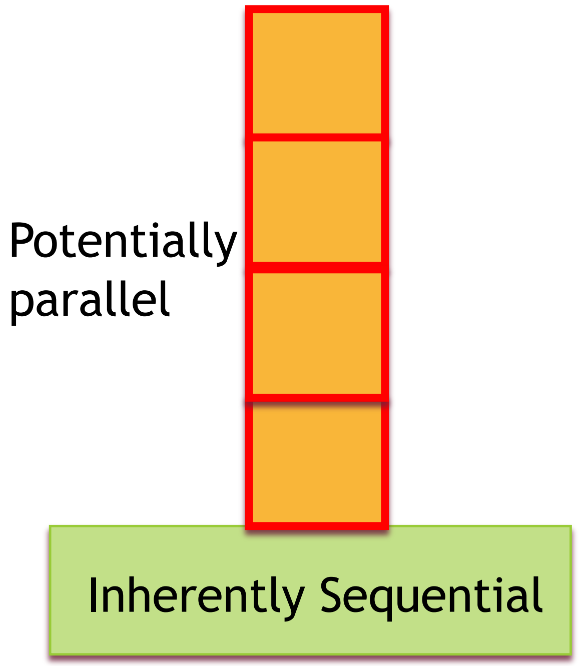


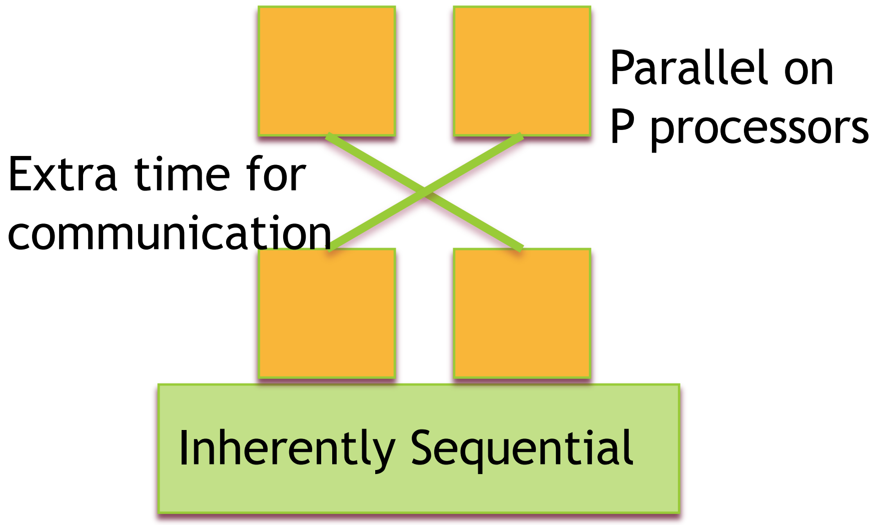
EXECUTION TIME COMPONENTS

Inherently sequential computations: $\sigma(n)$

Potentially parallel computations: $\varphi(n)$

Communication operations: $\kappa(n,p)$

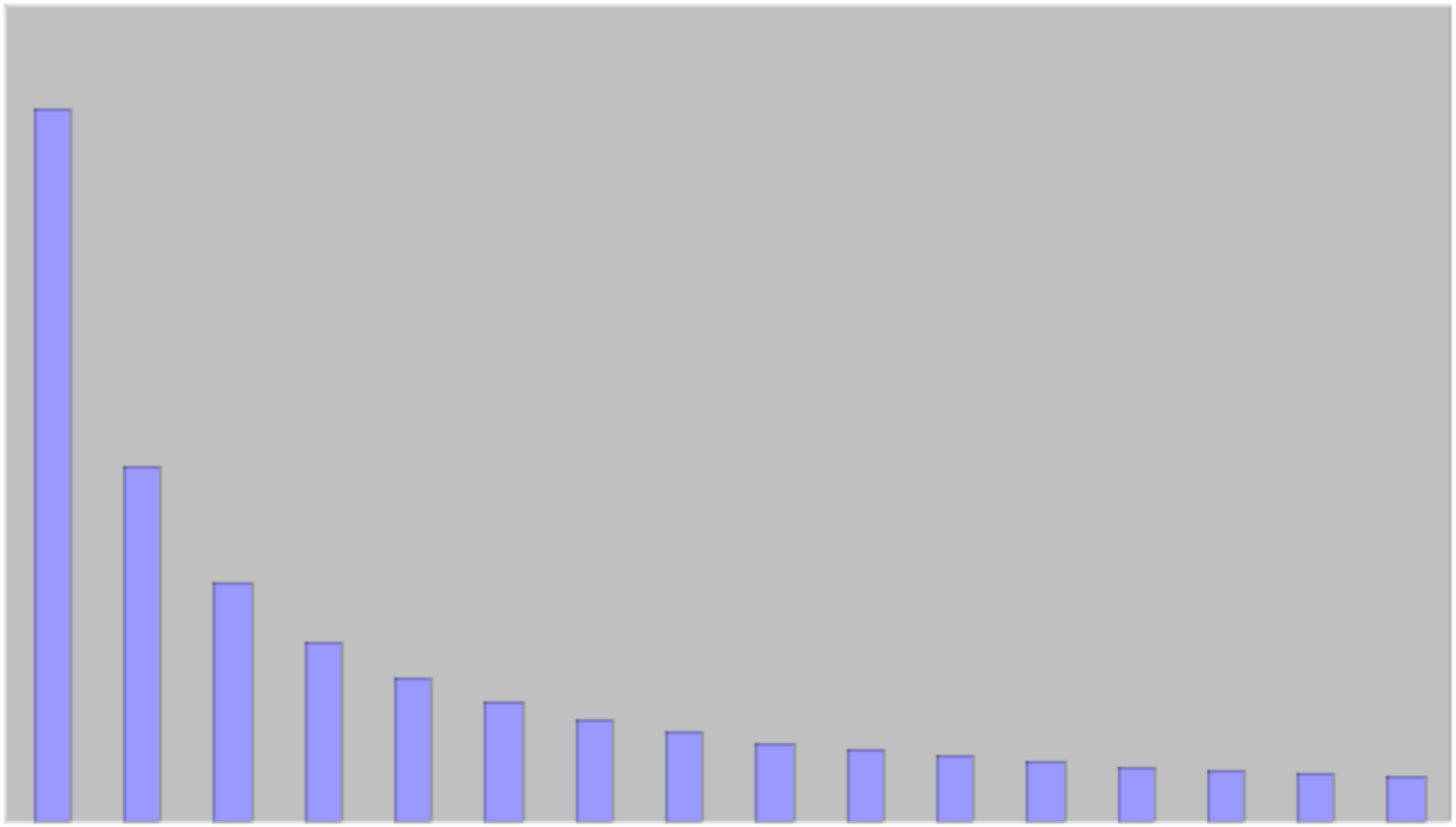




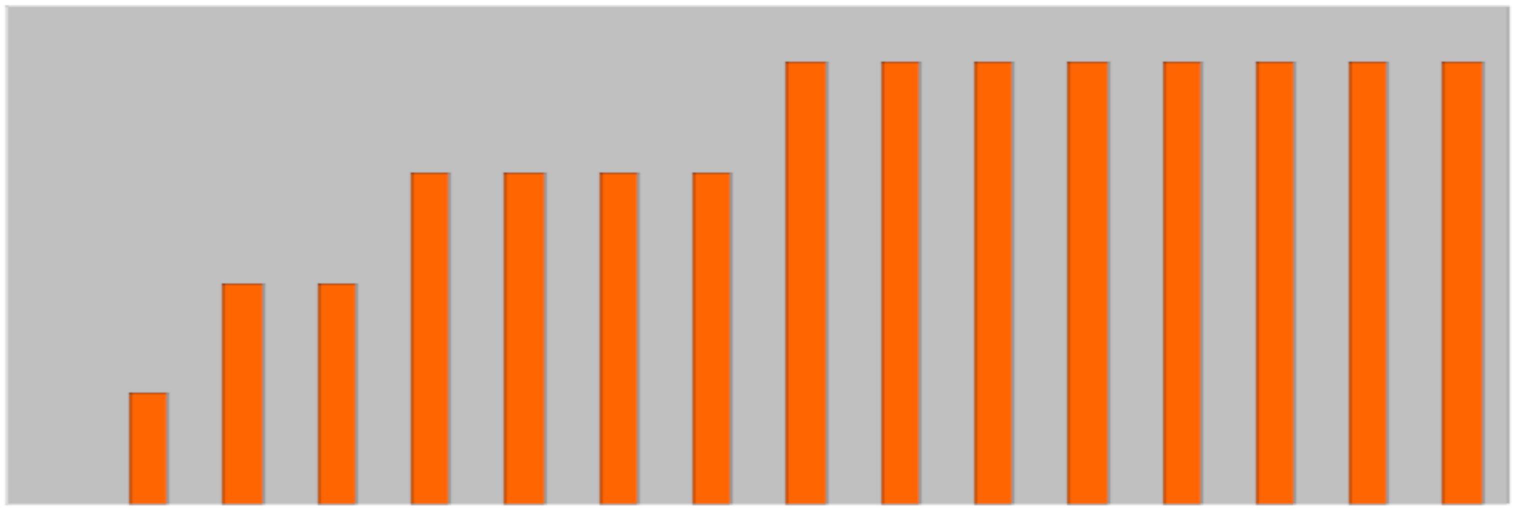
SPEEDUP EXPRESSION

$$\psi(n,p) \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)}$$

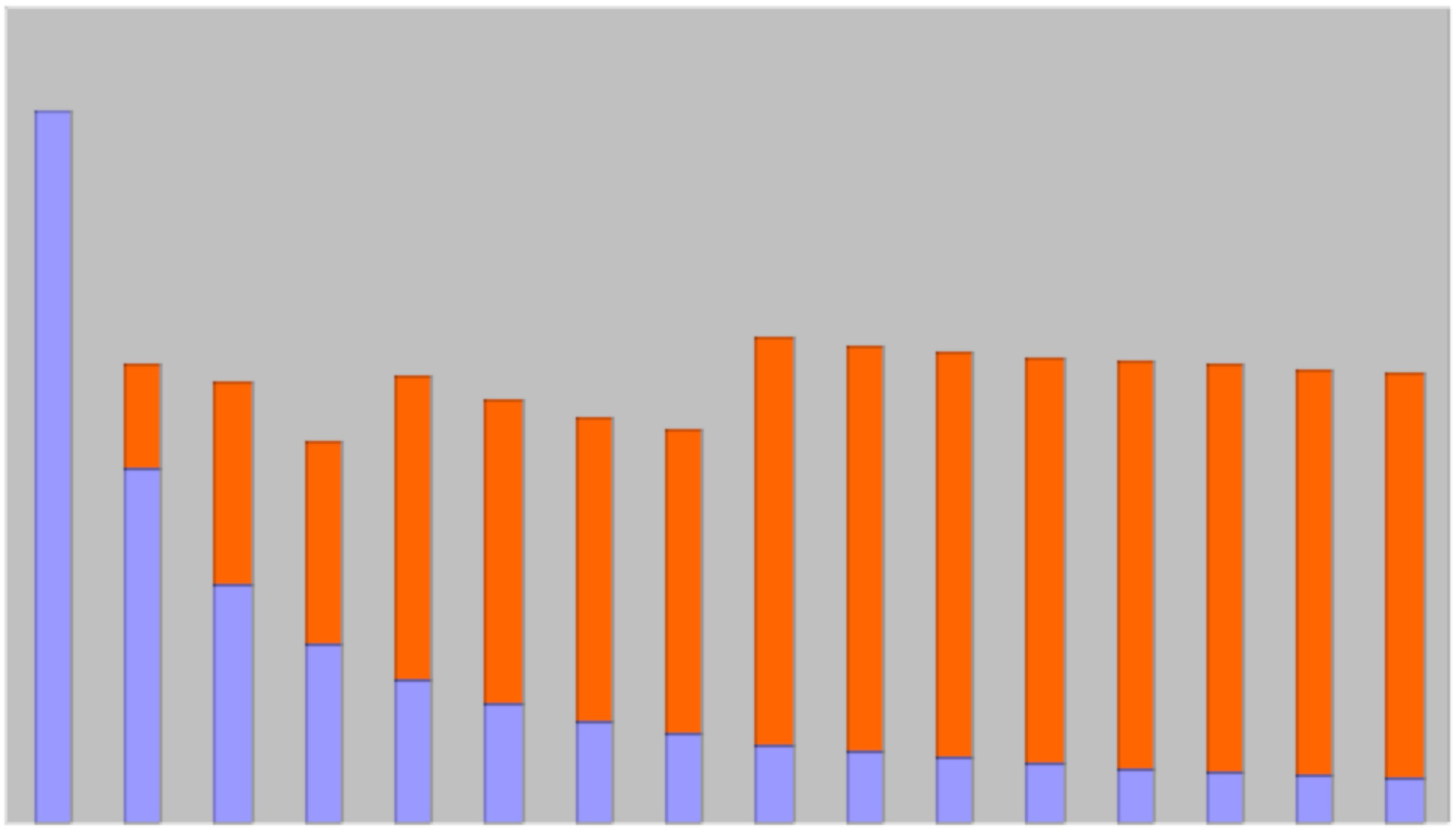
$\varphi(M)/P$



$\kappa(N,P)$



$\varphi(N)/P + \kappa(N,P)$



AMDAHL'S LAW

$$\psi(n,p) \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)}$$

$$\le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p}$$

Let $f = \sigma(n)/(\sigma(n)+\varphi(n))$ Percentage of sequential time

Let f be the fraction of operations that must be performed sequentially. The maximum speedup achievable by a parallel computer with p processors is given by

$$\psi \le \frac{1}{f+(1-f)/p}$$

EXAMPLE 1

95% of a program's execution time occurs inside a loop that can be executed in parallel. What is the maximum speedup we should expect from a parallel version of the program executing on 8 CPUs?

$$\psi \le \frac{1}{0.05 + (1 - 0.05)/8} \cong 5.9$$

EXAMPLE 2

20% of a program's execution time is spent within inherently sequential code. What is the limit to the speedup achievable by a parallel version of the program?

$$\lim_{p \to \infty} \frac{1}{0.2 + (1 - 0.2)/p} = \frac{1}{0.2} = 5$$

COMPLEXITY AND SPEEDUP

A parallel program of size n can run in x seconds. If the speed of the computer is made 4 times faster, what is the size of the program that can run in the same time (x seconds) in the new machine. Given the complexities of the algorithms are;

$\theta(n^4)$

$\theta(n^2)$

$\theta(n)$

$\theta(n\log n)$

NEXT CLASS

Data Structures

- Balanced Binary Trees (recap)
- B-Trees
- Splay tree
- Heaps, Priority Queues (recap)
- Bloom Filters
- Hashing