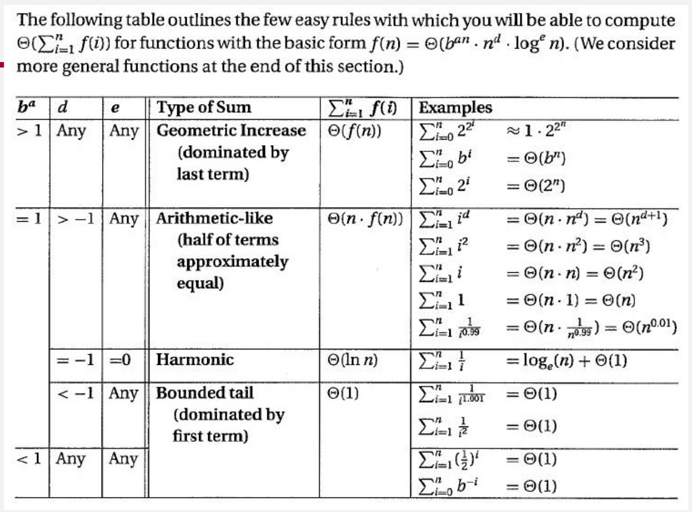


The following table outlines the few easy rules with which you will be able to compute $\Theta(\sum_{i=1}^{n} f(i))$ for functions with the basic form $f(n) = \Theta(b^{an} \cdot n^d \cdot \log^e n)$. (We consider more general functions at the end of this section.)

b^a	d	e	Type of Sum	$\sum_{i=1}^{n} f(i)$	Examples	
> 1	Any	Any	Geometric Increase (dominated by last term)	$\Theta(f(n))$	$\sum_{i=0}^{n} 2^{2^i}$	$\approx 1 \cdot 2^{2^n}$
					$\sum_{i=0}^{n} b^i$	$= \Theta(b^n)$
					$\sum_{i=0}^{n} 2^i$	$= \Theta(2^n)$
$= 1$	> -1	Any	Arithmetic-like (half of terms approximately equal)	$\Theta(n \cdot f(n))$	$\sum_{i=1}^{n} i^d$	$= \Theta(n \cdot n^d) = \Theta(n^{d+1})$
					$\sum_{i=1}^{n} i^2$	$= \Theta(n \cdot n^2) = \Theta(n^3)$
					$\sum_{i=1}^{n} i$	$= \Theta(n \cdot n) = \Theta(n^2)$
					$\sum_{i=1}^{n} 1$	$= \Theta(n \cdot 1) = \Theta(n)$
					$\sum_{i=1}^{n} \frac{1}{i^{0.99}}$	$= \Theta(n \cdot \frac{1}{n^{0.99}}) = \Theta(n^{0.01})$
	$= -1$	$=0$	Harmonic	$\Theta(\ln n)$	$\sum_{i=1}^{n} \frac{1}{i}$	$= \log_e(n) + \Theta(1)$
	< -1	Any	Bounded tail (dominated by first term)	$\Theta(1)$	$\sum_{i=1}^{n} \frac{1}{i^{1.001}}$	$= \Theta(1)$
					$\sum_{i=1}^{n} \frac{1}{i^2}$	$= \Theta(1)$
< 1	Any	Any			$\sum_{i=1}^{n} (\frac{1}{2})^i$	$= \Theta(1)$
					$\sum_{i=0}^{n} b^{-i}$	$= \Theta(1)$

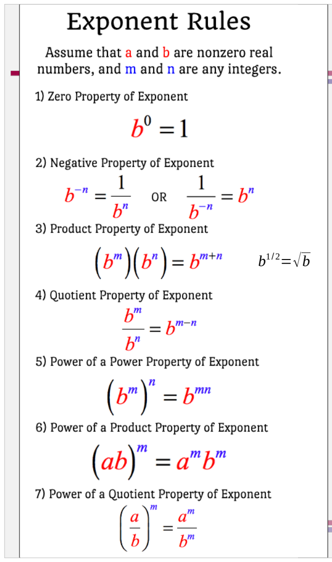
from Jeff Edmonds, *How to Think About Algorithms*

Big-Oh for Sums

Use the big-Oh for sums table to find the Θ approximation for the sum $\sum_{i=1}^{n} i \log i$.

2. [W] Give the Θ approximation for each of the following sums. Use the big-Oh for sums table.

   a. $\Sigma_{i=1..n}$ (log i)
   b. $\Sigma_{i=1..n}$ $(1/2^i)$
   c. $\Sigma_{i=1..\log n}$ $(n\ i^2)$
   d. $\Sigma_{i=1..n}$ $\Sigma_{j=1..i^2}$ (ij log i)



Exponent Rules

Assume that a and b are nonzero real numbers, and m and n are any integers.

1) Zero Property of Exponent

$$b^0 = 1$$

2) Negative Property of Exponent

$$b^{-n} = \frac{1}{b^n} \quad \text{OR} \quad \frac{1}{b^{-n}} = b^n$$

3) Product Property of Exponent

$$\left(b^m\right)\left(b^n\right) = b^{m+n} \qquad b^{1/2} = \sqrt{b}$$

4) Quotient Property of Exponent

$$\frac{b^m}{b^n} = b^{m-n}$$

5) Power of a Power Property of Exponent

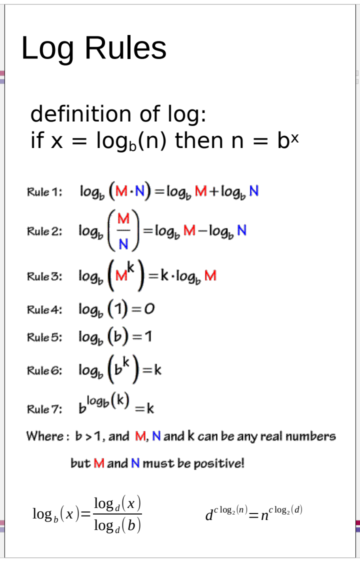
$$\left(b^m\right)^n = b^{mn}$$

6) Power of a Product Property of Exponent

$$(ab)^m = a^m b^m$$

7) Power of a Quotient Property of Exponent

$$\left(\frac{a}{b}\right)^m = \frac{a^m}{b^m}$$



Log Rules

definition of log:
if x = $\log_b(n)$ then n = b^x

Rule 1: $\log_b(M \cdot N) = \log_b M + \log_b N$

Rule 2: $\log_b\left(\frac{M}{N}\right) = \log_b M - \log_b N$

Rule 3: $\log_b\left(M^k\right) = k \cdot \log_b M$

Rule 4: $\log_b(1) = 0$

Rule 5: $\log_b(b) = 1$

Rule 6: $\log_b\left(b^k\right) = k$

Rule 7: $b^{\log_b(k)} = k$

Where: $b > 1$, and M, N and k can be any real numbers but M and N must be positive!

$$\log_b(x) = \frac{\log_d(x)}{\log_d(b)} \qquad d^{c \log_2(n)} = n^{c \log_2(d)}$$

Logarithms and Exponents

For the following pairs of functions, indicate whether f=O(g), f=Ω(g), or f=Θ(g).

- $f(n) = \log n^2, g(n) = 2^{\log n}$ [pairA]
- $f(n) = \log_{10} n, g(n) = 10n$ [pairB]
- $f(n) = \log_{10} n, g(n) = \log_2 2n$ [pairC]

- tips
  - know the growth rate ordering of common functions: 1, log n, n, n log n, n^2, 2^n, n!
  - simplify other functions to make them more familiar

Solving Recurrence Relations

$T(n) = a\,T(n-b) + f(n)$ where $f(n) = \Theta(n^c \log^d n)$

Cases are based on the number of subproblems and f(n).

a	f(n)	behavior	solution
> 1	any	base case dominates (too many leaves)	$T(n) = \Theta(a^{n/b})$
1	≥ 1	all levels are important	$T(n) = \Theta(n\,f(n))$

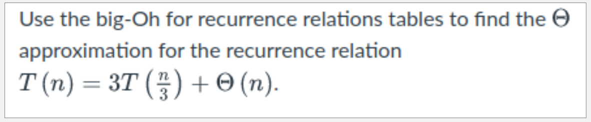
Solving Recurrence Relations

$T(n) = a\,T(n/b) + f(n)$ where $f(n) = \Theta(n^c \log^d n)$

Cases are based on the relationship between the number of subproblems, the problem size, and f(n).

(log a)/(log b) vs c	d	behavior	solution
<	any	**top level dominates** – more work splitting/combining than in subproblems (root too expensive)	$T(n) = \Theta(f(n))$
=	> -1	**all levels are important** – log n steps to get to base case, and roughly same amount of work in each level	$T(n) = \Theta(f(n) \log n)$
=	< -1	**base cases dominate** – so many subproblems that taking care of all the base cases is more work than splitting/combining (too many leaves)	$T(n) = \Theta(n^{(\log a)/(\log b)})$
>	any		

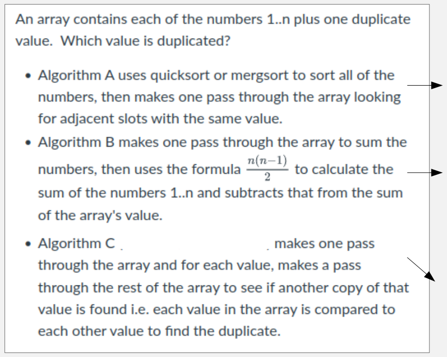
Big-Oh for Recurrence Relations


Use the big-Oh for recurrence relations tables to find the Θ approximation for the recurrence relation $T(n) = 3T\left(\frac{n}{3}\right) + \Theta(n)$.

- $T(n) = 2T(n/2) + \Theta(\log n)$
- $T(n) = 3T(n/9) + \Theta(n)$
- $T(n) = 8T(n/2) + \Theta(n^2)$
- $T(n) = T(n-1) + \Theta(1)$

Big-Oh From Algorithms

use the known typical tasks


An array contains each of the numbers 1..n plus one duplicate value. Which value is duplicated?

- Algorithm A uses quicksort or mergsort to sort all of the numbers, then makes one pass through the array looking for adjacent slots with the same value.
- Algorithm B makes one pass through the array to sum the numbers, then uses the formula $\frac{n(n-1)}{2}$ to calculate the sum of the numbers 1..n and subtracts that from the sum of the array's value.
- Algorithm C makes one pass through the array and for each value, makes a pass through the rest of the array to see if another copy of that value is found i.e. each value in the array is compared to each other value to find the duplicate.

sort, then examine each object a fixed number of times → $\Theta(n \log n) + \Theta(n) = \Theta(n \log n)$

examine each object a fixed number of times, then examine only a fixed number of things → $\Theta(n) + \Theta(1) = \Theta(n)$

for each object, examine each object a fixed number of times → $\Theta(n) \times \Theta(n) = \Theta(n^2)$

Big-Oh From Algorithms

An array contains each of the numbers 1..n plus one duplic
value. Which value is duplicated?

- Algorithm A uses quicksort or mergsort to sort all of the numbers, then makes one pass through the array looking for adjacent slots with the same value.
- Algorithm B makes one pass through the array to sum the numbers, then uses the formula $\frac{n(n-1)}{2}$ to calculate the sum of the numbers 1..n and subtracts that from the sum of the array's value.
- Algorithm C makes one pass through the array and for each value, makes a pass through the rest of the array to see if another copy of that value is found i.e. each value in the array is compared to each other value to find the duplicate.

```
sort(arr)
for i ← 0..n-2
  if arr[i] == arr[i+1]
    dup ← arr[i]
    break
```

```
sum ← 0
for i ← 0..n-1
  sum += arr[i]
dup ← sum-n(n-1)/2
```

```
for i ← 0..n-1
  for j ← i+1..n-1
    if arr[i] == arr[j]
      dup ← arr[j]
      break
```

Big-Oh From Algorithms

- We grow an array by increasing its length by 1 each time.

```
double[] numbers = new double[1];
for ( int i = 0 ; i < n ; i++ ) {
  if ( i >= numbers.length ) {
    numbers = Arrays.copyOf(numbers,numbers.length+1);
  }
  numbers[i] = Math.random();
}
```

- We grow an array by doubling its length each time.

```
double[] numbers = new double[1];
for ( int i = 0 ; i < n ; i++ ) {
  if ( i >= numbers.length ) {
    numbers = Arrays.copyOf(numbers,2*numbers.length);
  }
  numbers[i] = Math.random();
}
```

Big-Oh From Algorithms

```
• void hanoi ( int n, int src, int dst, int spare ) {
    if ( n == 1 ) {
      System.out.println("move disk from "+src+" to "+dst);
    } else {
      hanoi(n-1,src,spare,dst);
      System.out.println("move disk from "+src+" to "+dst);
      hanoi(n-1,spare,dst,src);
    }
  }
```

Big-Oh From Algorithms

- Mergesort.

```
void mergesort ( int[] arr, int left, int right ) {
  if ( right > left ) {
    int middle = (left+right)/2;
    mergesort(arr,left,middle);
    mergesort(arr,middle+1,right);
    merge(arr,left,middle,right);
  }
}

void merge ( int[] arr, int left, int middle, int right ) {
  int[] merged = new int[right-left+1];
  int int i = left, j = middle+1, k = 0;
  for ( ; i <= middle && j <= right ; k++ ) {
    if ( arr[i] < arr[j] ) { merged[k] = arr[i]; i++; }
    else { merged[k] = arr[j]; j++; }
  }
  for ( ; i <= middle ; i++, k++ ) {
    merged[k] = arr[i];
  }
  for ( ; j <= right ; j++, k++ ) {
    merged[k] = arr[i];
  }
  System.arraycopy(merged,0,arr,left,merged.length);
}
```