

### Logarithm Rules and Properties

- $\log_a 1 = 0$
- $\log_a a = 1$
- $\log_a x^y = y \log_a x$
- $\log_a xy = \log_a x + \log_a y$
- $\log_a \frac{x}{y} = \log_a x - \log_a y$
- $a^{\log_b x} = x^{\log_b a}$
- $\log_a x = \frac{\log_b x}{\log_b a} = \log_a b \log_b x$

### Summations for Printing

#### Useful Formulae

Sum	Closed Form
$\sum_{k=0}^n ar^k; r \neq 0$	$\frac{ar^{n+1} - a}{r - 1}; r \neq 1$
$\sum_{k=1}^n k$	$\frac{n(n+1)}{2}$
$\sum_{k=1}^n k^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{k=1}^n k^3$	$\frac{n^2(n+1)^2}{4}$
$\sum_{k=0}^{\infty} x^k;  x  < 1$	$\frac{1}{1-x}$
$\sum_{k=1}^{\infty} kx^{k-1};  x  < 1$	$\frac{1}{(1-x)^2}$

### Summation Rules

- $\sum_{i=l}^u ca_i = c \sum_{i=l}^u a_i$
- $\sum_{i=l}^u (a_i \pm b_i) = \sum_{i=l}^u a_i \pm \sum_{i=l}^u b_i$
- $\sum_{i=l}^u a_i = \sum_{i=l}^m a_i + \sum_{i=m+1}^u a_i$ , where  $l \leq m < u$
- $\sum_{i=l}^u (a_i - a_{i-1}) = a_u - a_{l-1}$

### Don't panic



### Insertion Sort - Algorithm

1 3 7 2 0      [1 3 7] 2 0  
 [1] 3 7 2 0      [1 2 3 7] 0  
 [1 3] 7 2 0      [0 1 2 3 7]

Starting from one end, ensure the sub-array sorted in each pivot

### Selection sort - Algorithm

1 3 7 2 0      [0 1 2] 3 7  
 [0] 1 3 7 2      [0 1 2 3] 7  
 [0 1] 3 7 2      [0 1 2 3 7]

Starting from one end, ensure the sub-array sorted in each pivot

### Bubble sort - Algorithm

1 3 7 2 0      1 2(0 3) 7  
 1 3(2 7) 0      1(0 2) 3 7  
 1 3 2(0 7)      (0 1) 2 3 7  
 1(2 3) 0 7      0 1 2 3 7

Binary comparison and swap, shift each pivot by at most 1 position in an iteration

### Sorting Complexities

Algorithm	Time		Space
	Best	Worst	Worst
Bubble Sort	$O(n)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(1)$
Quicksort	$O(n \log(n))$	$O(n^2)$	$O(\log(n))$
Heapsort	$O(n \log(n))$	$O(n \log(n))$	$O(1)$
Mergesort	$O(n \log(n))$	$O(n \log(n))$	$O(n)$

### Asymptotic Notations

#### Big-O

$T(n) \in O(f(n))$  if there are constants  $c > 0$  and  $n_0$  such that  $T(n) \leq c f(n)$  for all  $n \geq n_0$

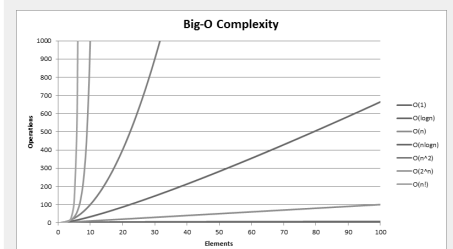
#### Big-Omega

$T(n) \in \Omega(f(n))$  if  $f(n) \in O(T(n))$

#### Big-Theta

$T(n) \in \Theta(f(n))$  if  $T(n) \in O(f(n))$  and  $T(n) \in \Omega(f(n))$

### Big-O Complexity Chart



### Queue ADT

#### Queue property

FIFO: First In First Out

#### Core operations

- enqueue - dequeue - is\_empty

### Priority Queue ADT

#### Queue property

Lower Priority Value Out First

#### Core operations

- insert - deleteMin - isEmpty

### Stack ADT

#### Stack property

LIFO: Last In First Out

#### Core operations

- push - pop - top - is\_empty

### d-Heap ADT

child  $(i-1)*d+2 \sim i*d+1$

parent  $\lfloor (i-2)/d \rfloor + 1$

root 1

next free size+1

### (Min) Heap Tree - ADT

#### Heap-order property

parent's key  $\leq$  children's keys

#### Structure property

nearly complete tree

#### Heapify algorithm

Heapify the tree from bottom up, percolate DOWN a node as deep as needed for each node.

#### Binary Heap Operations Complexity:

Heapify -  $O(n)$

Find Min -  $O(1)$

Insert -  $O(\log(n))$

Delete -  $O(\log(n))$

### Loop -> (Tail) Recursion

```
//Loop
int i = 0;
while (i < n)
    doFoo(i);
    i++;
//Recursion
void recDoFoo(int i, int n){
    if (i < n) {
        doFoo(i);
        recDoFoo(i + 1, n);}
}
recDoFoo(0, n);
```

#### Equivalent for loop:

for (int i=0; i<n; i++) doFoo(i);

### Tail Recursion -> Iteration

```
//Tail Recursion
int fact_acc (int n, int acc) {
    if (n)
        return fact_acc(n -1, acc * n);
    return acc;
}
//Iteration
int fact_acc (int n, int acc) {
    for (; n-- > 0)
        acc = acc * n;
    return acc;
}
```

### Recurrence Simplification Strategy

1) Find  $T(n)$  for the base cases.

2) Expand  $T(n)$  for the general patterns.

3) Drive the recursive term to the base case.

4) Solve and represent  $k$  with  $n$  by inequality(equality).

5) Substitute back to  $T(n)$ .

e.g.

$T[n \leq n_0] = C_1$

$T[n] = T[n/2] + C_2$

->  $T[n] = T[n*(1/2)] + C_2$

->  $T[n] = T[n*(1/2)^k] + k*C_2$

Let  $n*(1/2)^k \leq n_0 \rightarrow T(n*(1/2)^k) = C_1$

->  $n*(1/2)^k * (2)^k \leq n_0 * (2)^k$

->  $n/n_0 \leq (2)^k$

->  $\log_2(n/n_0) \leq \log_2((2)^k)$

->  $\log_2(n/n_0) \leq k$

->  $T[n] = C_1 + k*C_2 = C_1 + C_2*\log_2(n/n_0)$

### Summation Formulas

- $\sum_{i=1}^n 1 = 1 + 1 + \dots + 1 = n - 1 + 1 (i, n \text{ are integer limits, } i \leq n); \sum_{i=1}^n 1 = n$
- $\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$
- $\sum_{i=1}^n i^2 = 1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$
- $\sum_{i=1}^n i^k = 1^k + 2^k + \dots + n^k \approx \frac{1}{k+1}n^{k+1}$
- $\sum_{i=1}^n a^i = 1 + a + \dots + a^n = \frac{a^{n+1} - 1}{a - 1} (a \neq 1); \sum_{i=0}^n 2^i = 2^{n+1} - 1$
- $\sum_{i=1}^n i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \dots + n2^n = (n-1)2^{n+1} + 2$
- $\sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \dots + \frac{1}{n} \approx \ln n + \gamma$ , where  $\gamma \approx 0.5772\dots$  (Euler's constant)
- $\sum_{i=1}^n \lg i \approx n \lg n$

