Please initial:

Order Notation Summary

O-notation: $f(n) \in O(g(n))$ if there exist constants $c > 0$ and $n_0 > 0$ such that $0 \le f(n) \le c\, g(n)$ for all $n \ge n_0$.

Ω-notation: $f(n) \in \Omega(g(n))$ if there exist constants $c > 0$ and $n_0 > 0$ such that $0 \le c\, g(n) \le f(n)$ for all $n \ge n_0$.

Θ-notation: $f(n) \in \Theta(g(n))$ if there exist constants $c_1, c_2 > 0$ and $n_0 > 0$ such that $0 \le c_1\, g(n) \le f(n) \le c_2\, g(n)$ for all $n \ge n_0$.

o-notation: $f(n) \in o(g(n))$ if for all constants $c > 0$, there exists a constant $n_0 > 0$ such that $0 \le f(n) < c\, g(n)$ for all $n \ge n_0$.

ω-notation: $f(n) \in \omega(g(n))$ if for all constants $c > 0$, there exists a constant $n_0 > 0$ such that $0 \le c\, g(n) < f(n)$ for all $n \ge n_0$.

Techniques for Order Notation

Suppose that $f(n) > 0$ and $g(n) > 0$ for all $n \ge n_0$. Suppose that

$$L = \lim_{n\to\infty} \frac{f(n)}{g(n)} \quad \text{(in particular, the limit exists).}$$

Then

$$f(n) \in \begin{cases} o(g(n)) & \text{if } L = 0 \\ \Theta(g(n)) & \text{if } 0 < L < \infty \\ \omega(g(n)) & \text{if } L = \infty. \end{cases}$$

The required limit can often be computed using *l'Hôpital's rule*. Note that this result gives *sufficient* (but not necessary) conditions for the stated conclusions to hold.

Useful Sums

Arithmetic sequence:

$\sum_{i=0}^{n-1} i = ???$ $\qquad \sum_{i=0}^{n-1}(a + di) = na + \frac{dn(n-1)}{2} \in \Theta(n^2) \quad \text{if } d \neq 0.$

Geometric sequence:

$\sum_{i=0}^{n-1} 2^i = ???$

$$\sum_{i=0}^{n-1} a\, r^i = \begin{cases} a\dfrac{r^n - 1}{r - 1} & \in \Theta(r^{n-1}) & \text{if } r > 1 \\ na & \in \Theta(n) & \text{if } r = 1 \\ a\dfrac{1 - r^n}{1 - r} & \in \Theta(1) & \text{if } 0 < r < 1. \end{cases}$$

Harmonic sequence:

$\sum_{i=1}^{n} \frac{1}{i} = ???$ $\qquad H_n := \sum_{i=1}^{n} \frac{1}{i} = \ln n + \gamma + o(1) \in \Theta(\log n)$

A few more:

$\sum_{i=1}^{n} \frac{1}{i^2} = ???$ $\qquad \sum_{i=1}^{n} \frac{1}{i^2} = \frac{\pi^2}{6} \in \Theta(1)$

$\sum_{i=1}^{n} i^k = ???$ $\qquad \sum_{i=1}^{n} i^k \in \Theta(n^{k+1}) \quad \text{for } k \ge 0$

Useful Math Facts

Logarithms:

- $c = \log_b(a)$ means $b^c = a$. E.g. $n = 2^{\log n}$.
- $\log(a)$ (in this course) means $\log_2(a)$
- $\log(a \cdot c) = \log(a) + \log(c)$, $\log(a^c) = c \log(a)$
- $\log_b(a) = \frac{\log_c a}{\log_c b} = \frac{1}{\log_a(b)}$, $a^{\log_b c} = c^{\log_b a}$
- $\ln(x) =$ natural $\log = \log_e(x)$, $\frac{d}{dx} \ln x = \frac{1}{x}$
- concavity: $\alpha \log x + (1-\alpha)\log y \le \log(\alpha x + (1-\alpha) y)$ for $0 \le \alpha \le 1$

Factorial:

- $n! := n(n-1)(n-2)\cdots 2 \cdot 1 =$ # ways to permute n elements
- $\log(n!) = \log n + \log(n-1) + \cdots + \log 2 + \log 1 \in \Theta(n \log n)$

Probability and moments:

- $E[aX] = aE[X]$, $E[X + Y] = E[X] + E[Y]$ (linearity of expectation)

Please initial:

Some Recurrence Relations

Recursion	resolves to	example
$T(n) = T(n/2) + \Theta(1)$	$T(n) \in \Theta(\log n)$	Binary search
$T(n) = 2T(n/2) + \Theta(n)$	$T(n) \in \Theta(n \log n)$	Mergesort
$T(n) = 2T(n/2) + \Theta(\log n)$	$T(n) \in \Theta(n)$	Heapify ($\to$ later)
$T(n) = T(cn) + \Theta(n)$ for some $0 < c < 1$	$T(n) \in \Theta(n)$	Selection ($\to$ later)
$T(n) = 2T(n/4) + \Theta(1)$	$T(n) \in \Theta(\sqrt{n})$	Range Search ($\to$ later)
$T(n) = T(\sqrt{n}) + \Theta(1)$	$T(n) \in \Theta(\log \log n)$	Interpolation Search ($\to$ later)

- Once you know the result, it is (usually) easy to prove by induction.
- Many more recursions, and some methods to find the result, in cs341.

HeapSort

- Idea: *PQ-sort* with heaps.
- But: Use same input-array A for storing heap.

```
HeapSort(A, n)
    // heapify
    n ← A.size()
    for i ← parent(last(n)) downto 0 do
        fix-down(A, n, i)

    // repeatedly find maximum
    while n > 1
        // delete the maximum
        swap items at A[root()] and A[last(n)])
        decrease n
        fix-down(A, n, root())
```

The for-loop takes $\Theta(n)$ time and the while-loop takes $O(n \log n)$ time.

Efficient In-Place partition (Hoare)

Idea: Keep swapping the outer-most wrongly-positioned pairs.

A	$\leq v$	?	$\geq v$	v
		i	j	

```
partition(A, p)
A: array of size n,   p: integer s.t. 0 ≤ p < n
1.  swap(A[n − 1], A[p])
2.  i ← −1,   j ← n − 1,   v ← A[n − 1]
3.  loop
4.      do i ← i + 1 while i < n and A[i] < v
5.      do j ← j − 1 while j > 0 and A[j] > v
6.      if i ≥ j then break   (goto 9)
7.      else swap(A[i], A[j])
8.  end loop
9.  swap(A[n − 1], A[i])
10. return i
```

Running time: $\Theta(n)$.

LSD-Radix-Sort

```
LSD-radix-sort(A)
A: array of size n, contains m-digit radix-R numbers
    for d ← m down to 1 do
        key-indexed-count-sort(A, d)
```

	$(d = 3) \Longrightarrow$		$(d = 2) \Longrightarrow$		$(d = 1) \Longrightarrow$	
12③		2③0		①01		021
23⓪		3②0		②10		101
02①		2①0		③20		123
32⓪		0②1		⓪21		210
21⓪		1⓪1		①23		230
23②		2③2		②30		232
10①		1②3		②32		320

- Loop-invariant: A is sorted w.r.t. digits $d, \ldots, m$ of each entry.
- **Time cost**: $\Theta(m(n + R))$ **Auxiliary space**: $\Theta(n + R)$