

Asymptotic Notations Revisit

Asymptotic Complexity

- Running time of an algorithm is a function of input size n **for large n** .
- Expressed using only the **highest-order term** in the expression for the exact running time.
 - Instead of exact running time, say $\Theta(n^2)$.
- Describes behavior of function in the limit.
- Written using ***Asymptotic Notation***.

Asymptotic Notation

- $O, \Theta, \Omega, o, \omega$
- Defined for functions over the natural numbers.
 - Ex: $f(n) = \Theta(n^2)$.
 - Describes how $f(n)$ grows in comparison to n^2 .
- Define a **set** of functions; in practice used to compare two function sizes.
- The notations describe different rate-of-growth relations between the defining function and the defined set of functions.

O-notation

For function $g(n)$, we define $O(g(n))$,
big-O of n , as the set:

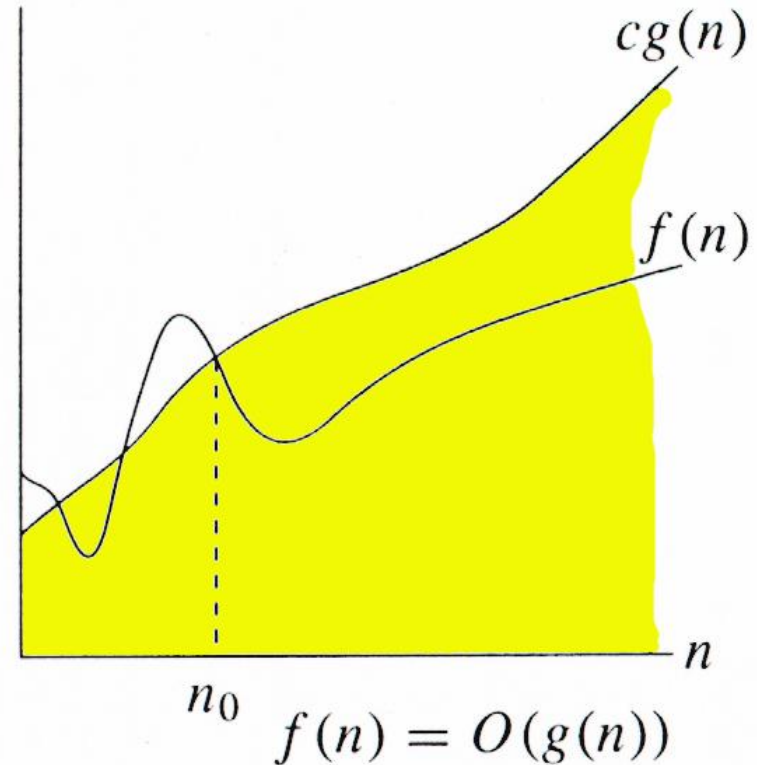
$O(g(n)) = \{f(n) :$
 \exists **positive constants c and n_0 , such**
that $\forall n \geq n_0$,
we have $0 \leq |f(n)| \leq c|g(n)|$ }

Intuitively: Set of all functions
whose *rate of growth* is the same
as or lower than that of $g(n)$.

$g(n)$ is an **asymptotic upper bound** for $f(n)$.

$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$.

$\Theta(g(n)) \subset O(g(n))$.



Intuition:

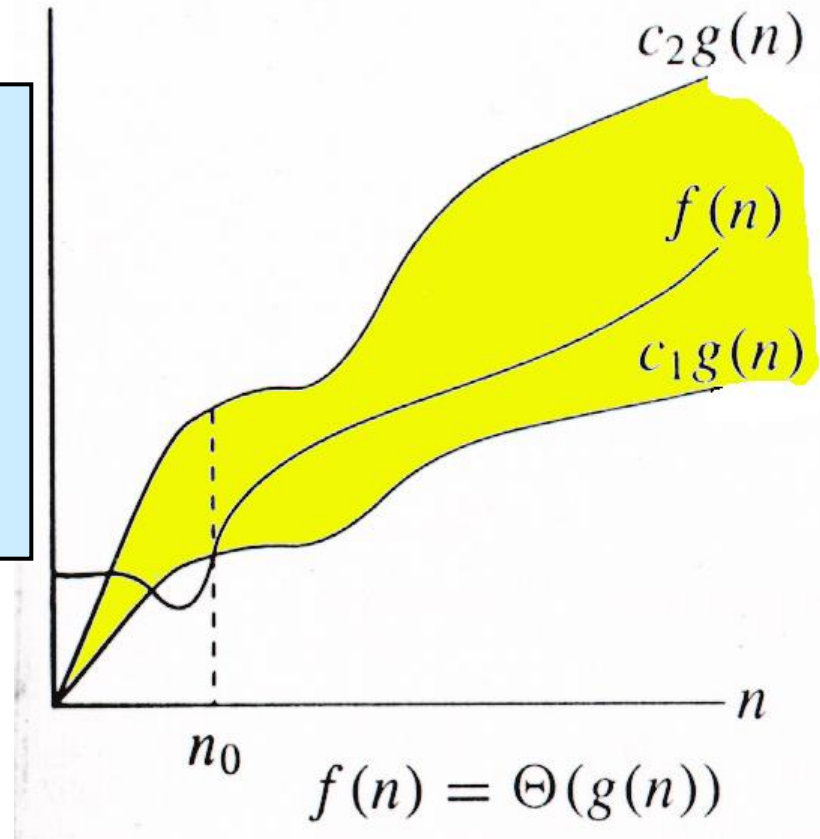
- $f(n) \in O(g(n))$ means $f(n)$ is “of order at most”, or “less than or equal to” $g(n)$ when we ignore small values of n and constants
- $f(n)$ is eventually trapped below (or = to) some constant multiple of $g(n)$
- some constant multiple of $g(n)$ is an upper bound for $f(n)$ (for large enough n)

Θ -notation

For function $g(n)$, we define $\Theta(g(n))$,
big-Theta of n , as the set:

$$\Theta(g(n)) = \{f(n) : \\ \exists \text{ positive constants } c_1, c_2, \text{ and } n_0, \\ \text{such that } \forall n \geq n_0, \text{ we have} \\ 0 \leq c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)| \\ \}$$

Intuitively: Set of all functions that
have the same *rate of growth* as $g(n)$.



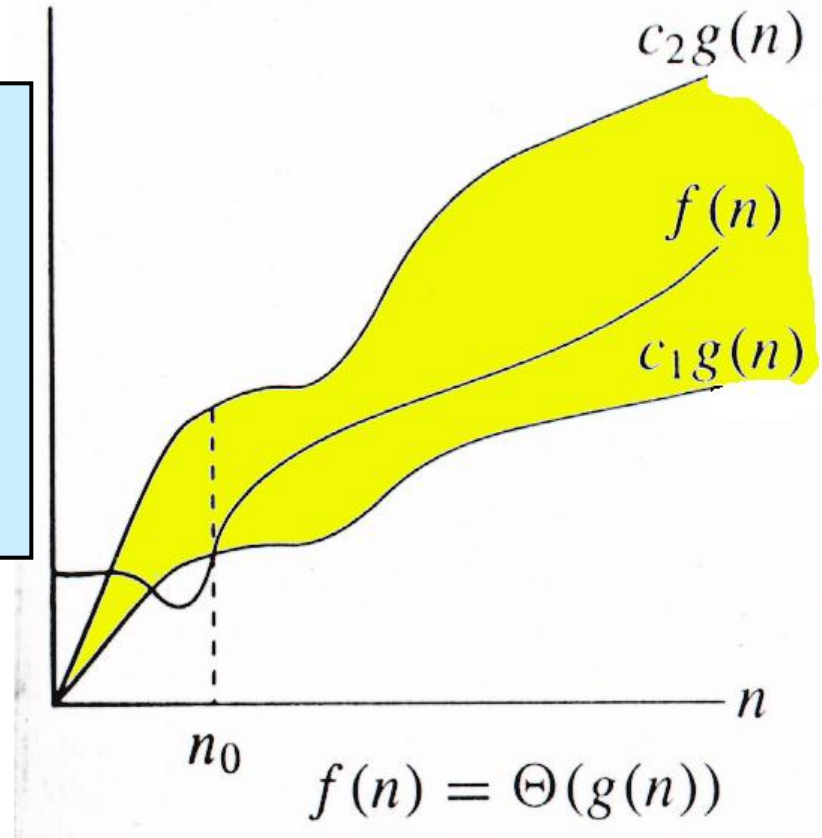
$g(n)$ is an **asymptotically tight bound** for $f(n)$.

Θ -notation

For function $g(n)$, we define $\Theta(g(n))$,
big-Theta of n , as the set:

$$\Theta(g(n)) = \{f(n) : \\ \exists \text{ positive constants } c_1, c_2, \text{ and } n_0, \\ \text{such that } \forall n \geq n_0, \text{ we have} \\ 0 \leq c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)| \\ \}$$

Technically, $f(n) \in \Theta(g(n))$.
Older usage, $f(n) = \Theta(g(n))$.



$f(n)$ and $g(n)$ are nonnegative, for large n .

Intuition:

- $f(n) \in \theta(g(n))$ means $f(n)$ is “of the same order as”, or “equal to” $g(n)$ when we ignore small values of n .
- $f(n)$ is eventually trapped between two constant multiples of $g(n)$

$\Theta(g(n)) = \{f(n) : \exists \text{ positive constants } c_1, c_2, \text{ and } n_0, \text{ such that } \forall n \geq n_0, 0 \leq c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)|\}$

Example

- $10n^2 - 3n = \Theta(n^2)$
- What constants for n_0 , c_1 , and c_2 will work?
- Make c_1 a little smaller than the leading coefficient, and c_2 a little bigger.
- *To compare orders of growth, look at the leading term.*
- Exercise: Prove that $n^2/2 - 3n = \Theta(n^2)$

Ω -notation

For function $g(n)$, we define $\Omega(g(n))$, **big-Omega** of n , as the set:

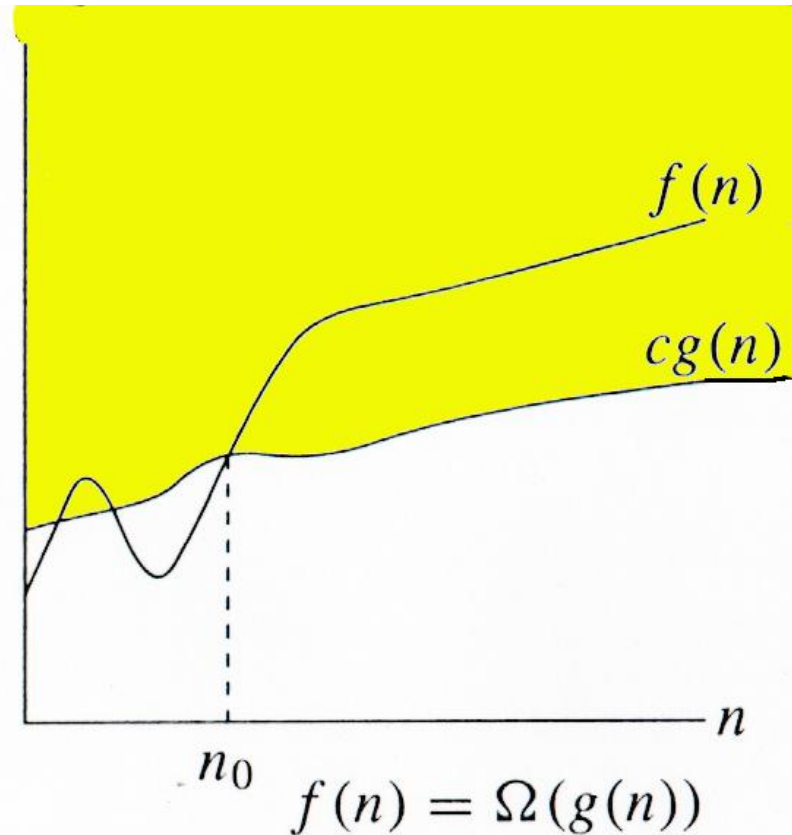
$\Omega(g(n)) = \{f(n) :$
 \exists **positive constants c and n_0 , such**
that $\forall n \geq n_0$,
we have $0 \leq c|g(n)| \leq |f(n)|\}$

Intuitively: Set of all functions whose *rate of growth* is the same as or higher than that of $g(n)$.

$g(n)$ is an **asymptotic lower bound** for $f(n)$.

$$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n)).$$

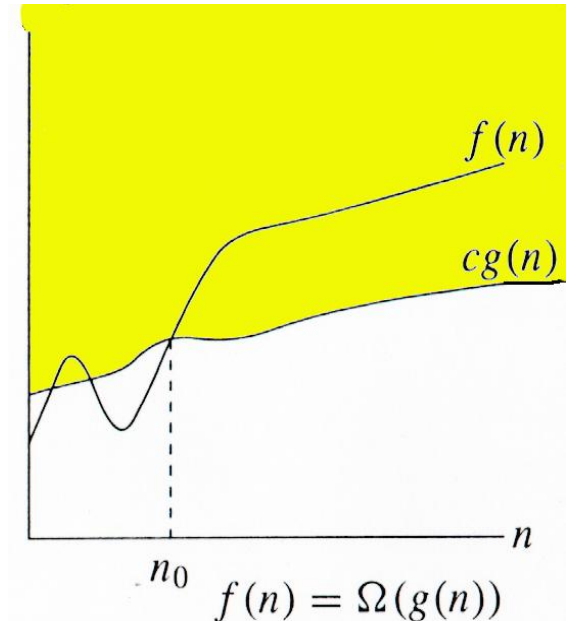
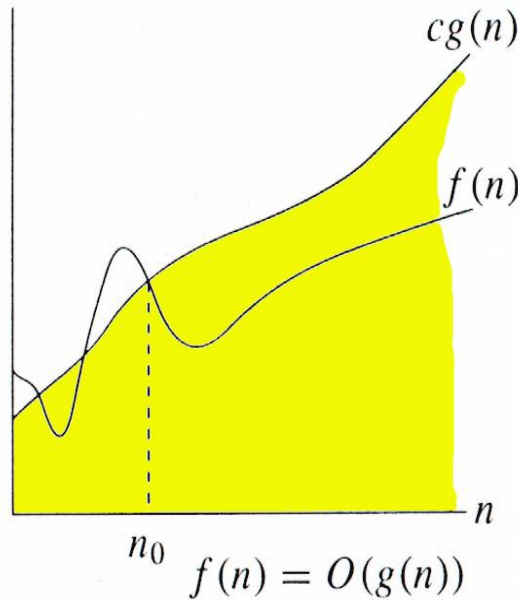
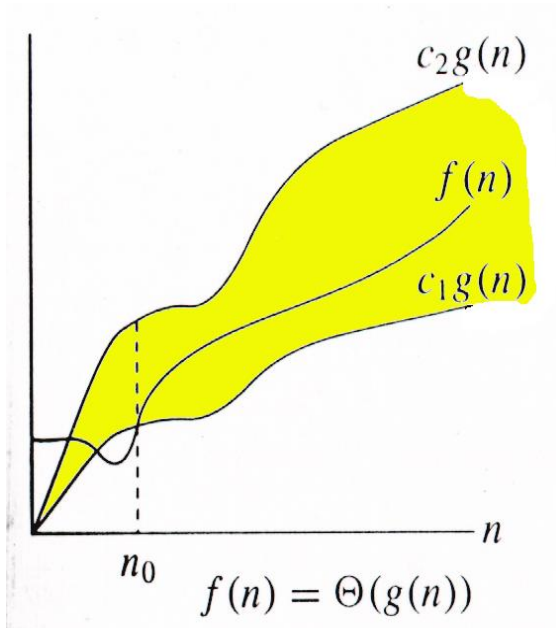
$$\Theta(g(n)) \subset \Omega(g(n)).$$



Intuition:

- $f(n) \in \Omega(g(n))$ means $f(n)$ is “of order at least” or “greater than or equal to” $g(n)$ when we ignore small values of n .
- $f(n)$ is eventually trapped above (or = to) some constant multiple of $g(n)$
- some constant multiple of $g(n)$ is a lower bound for $f(n)$ (for large enough n)

Relations Between Θ , O , Ω



Relations Between Θ , Ω , O

Theorem : For any two functions $g(n)$ and $f(n)$,
 $f(n) = \Theta(g(n))$ iff
 $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

- I.e., $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$
- In practice, asymptotically tight bounds are obtained from asymptotic upper and lower bounds.

Little-o

- $f(n) = o(g(n))$ as $n \rightarrow \infty \Leftrightarrow \lim_{n \rightarrow \infty} \left(\frac{f(n)}{g(n)} \right) = 0$
- That is, for any positive constant c , there exists an n_0 such that for all $n > n_0$, $|f(n)| < c |g(n)|$
- **Intuitive meaning:**
- As n gets very large, $f(n)$ becomes insignificant in comparison to $g(n)$. We say, "f is little-o of g."

Asymptotic Notations Table

Asymptotic Notation	Comparison Notation	Limit Definition (as $n \rightarrow \infty$)	Meaning
$f(n) = O(g(n))$	$f(n) \leq c \cdot g(n)$	$\limsup_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$	Upper bound (worst case)
$f(n) = \Omega(g(n))$	$f(n) \geq c \cdot g(n)$	$\liminf_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0$	Lower bound (best case)
$f(n) = \Theta(g(n))$	$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = k$ where $0 < k < \infty$	Tight bound (both upper & lower)
$f(n) = o(g(n))$	$f(n) < c \cdot g(n)$ (for all $c > 0$)	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$	Strictly smaller
$f(n) = \omega(g(n))$	$f(n) > c \cdot g(n)$ (for all $c > 0$)	$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$	Strictly greater

Comparing $f(n) = n$ and $g(n) = n^2$

Notation	Mathematical Expression	Explanation
$f(n) = O(g(n))$	$\frac{f(n)}{g(n)} = \frac{n}{n^2} = \frac{1}{n} \rightarrow 0$	Yes, $f(n) = O(n^2)$ because it grows slower than n^2 .
$f(n) = \Omega(g(n))$	$\frac{f(n)}{g(n)} = \frac{1}{n} \rightarrow 0$	✗ No, not $\Omega(n^2)$, because $f(n)$ does not grow at least as fast as n^2 .
$f(n) = \Theta(g(n))$	$\frac{f(n)}{g(n)} = \frac{1}{n} \rightarrow 0$	✗ No, not $\Theta(n^2)$, because $f(n) \ll g(n)$.
$f(n) = o(g(n))$	$\frac{f(n)}{g(n)} = \frac{1}{n} \rightarrow 0$	✓ Yes, $f(n) = o(n^2)$, since it is strictly smaller in growth rate.
$f(n) = \omega(g(n))$	$\frac{f(n)}{g(n)} = \frac{1}{n} \rightarrow 0$	✗ No, $f(n)$ is not greater than $g(n)$.

Each Asymptotic Notation Holds

Notation	Functions	Limit	Conclusion
Big O	$f(n) = 3n, g(n) = n^2$	$\frac{f(n)}{g(n)} = \frac{3n}{n^2} = \frac{3}{n} \rightarrow 0$	$f(n) = O(n^2)$ ✓
Big Omega	$f(n) = 5n^2, g(n) = n$	$\frac{f(n)}{g(n)} = \frac{5n^2}{n} = 5n \rightarrow \infty$	$f(n) = \Omega(n)$ ✓
Big Theta	$f(n) = 4n + 10, g(n) = n$	$\frac{f(n)}{g(n)} = \frac{4n+10}{n} = 4 + \frac{10}{n} \rightarrow 4$	$f(n) = \Theta(n)$ ✓
Little o	$f(n) = \log n, g(n) = n$	$\frac{\log n}{n} \rightarrow 0$	$\log n = o(n)$ ✓
Little omega	$f(n) = n \log n, g(n) = n$	$\frac{n \log n}{n} = \log n \rightarrow \infty$	$f(n) = \omega(n)$ ✓

Example

- ***Insertion sort*** takes $\Theta(n^2)$ in the worst case, so sorting (as a *problem*) is $O(n^2)$. Why?
- Any sort algorithm must look at each item, so sorting is $\Omega(n)$.
- In fact, using (e.g.) merge sort, sorting is $\Theta(n \lg n)$ in the worst case.

Asymptotic Notation in Equations

- Can use asymptotic notation in equations to replace expressions containing lower-order terms.
- For example,
$$4n^3 + 3n^2 + 2n + 1 = 4n^3 + 3n^2 + \Theta(n)$$
$$= 4n^3 + \Theta(n^2) = \Theta(n^3). \text{ How to interpret?}$$
- In equations, $\Theta(f(n))$ always stands for an ***anonymous function*** $g(n) \in \Theta(f(n))$
 - In the example above, $\Theta(n^2)$ stands for $3n^2 + 2n + 1$.

Exponentials

- **Useful Identities:**

$$a^{-1} = \frac{1}{a}$$

$$(a^m)^n = a^{mn}$$

$$a^m a^n = a^{m+n}$$

- **Exponentials and polynomials**

$$\lim_{n \rightarrow \infty} \frac{n^b}{a^n} = 0$$

$$\Rightarrow n^b = o(a^n)$$

- What is the time complexity of the segment that require $n! + 2^n$ operations?

n
1
2
3
4
5
6
7
8
9

n!
1
2
6
24
120
720
5040
40320
362880

2ⁿ
2
4
8
16
32
64
128
256
512

For all $n \geq 4(n_0)$ and $c = 1$, $T(n) = O(n!)$

- If $A(n) = a_m n^m + \dots + a_1 n + a_0$ is a polynomial of degree m then $A(n) = O(n^m)$
- Using the definition of $A(n)$ and simple inequality

$$|A(n)| \leq |a_m| n^m + \dots + |a_1| n + |a_0|$$

$$|A(n)| \leq \left(|a_m| + \frac{|a_{m-1}|}{n} \dots + \frac{|a_0|}{n^m} \right) n^m$$

$$|A(n)| \leq (|a_m| + \dots + |a_0|) n^m, n \geq 1$$

Choosing $c = |a_m| + \dots + |a_1| + |a_0|$ and $n_0 = 1$

Summations – Review

Review on Summations

- Why do we need summation formulas?

For computing the running times of iterative constructs (loops).

Example: Maximum Subvector

Given an array $A[1...n]$ of numeric values (can be positive, zero, and negative) determine the subvector $A[i...j]$ ($1 \leq i \leq j \leq n$) whose sum of elements is maximum over all subvectors.

1	-2	2	2
---	----	---	---

Review on Summations

```
MaxSubvector(A, n)
  maxsum ← 0;
  for i ← 1 to n
    do for j = i to n
      sum ← 0
      for k ← i to j
        do sum += A[k]
      maxsum ← max(sum, maxsum)
  return maxsum
```

$$\blacklozenge T(n) = \sum_{i=1}^n \sum_{j=i}^n \sum_{k=i}^j 1$$

- ◆ NOTE: This is not a simplified solution.
- ◆ What *is* the final answer?

Review on Summations

- **Constant Series:** For integers a and b , $a \leq b$,

$$\sum_{i=a}^b 1 = b - a + 1$$

- **Linear Series (Arithmetic Series):** For $n \geq 0$,

$$\sum_{i=1}^n i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

- **Quadratic Series:** For $n \geq 0$,

$$\sum_{i=1}^n i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Review on Summations

- **Cubic Series:** For $n \geq 0$,

$$\sum_{i=1}^n i^3 = 1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

- **Geometric Series:** For real $x \neq 1$,

$$\sum_{k=0}^n x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

For $|x| < 1$,

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

Review on Summations

- **Linear-Geometric Series:** For $n \geq 0$, real $c \neq 1$,

$$\sum_{i=1}^n ic^i = c + 2c^2 + \cdots + nc^n = \frac{-(n+1)c^{n+1} + nc^{n+2} + c}{(c-1)^2}$$

- **Harmonic Series:** n th harmonic number, $n \in \mathbb{I}^+$,

$$\begin{aligned} H_n &= 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \\ &= \sum_{k=1}^n \frac{1}{k} = \ln(n) + O(1) \end{aligned}$$

Review on Summations

- **Telescoping Series:**

$$\sum_{k=1}^n a_k - a_{k-1} = a_n - a_0$$

- **Differentiating Series:** For $|x| < 1$,

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$$

Problem 1 *Prove the following by induction.*

$$\sum_{i=1}^n i^3 = \left(\sum_{i=1}^n i\right)^2$$

Induction Hypothesis:

Induction Step: Assume it is true for some k

$$\sum_{i=1}^k i^3 = \left(\sum_{i=1}^k i\right)^2.$$

We need to show that

$$\sum_{i=1}^{k+1} i^3 = \left(\sum_{i=1}^{k+1} i\right)^2.$$

$$\left(\sum_{i=1}^{k+1} i\right)^2 = \left[\sum_{i=1}^k i + (k+1)\right]^2$$

$$\begin{aligned}
\left(\sum_{i=1}^{k+1} i\right)^2 &= \left[\sum_{i=1}^k i + (k+1)\right]^2 \\
&= \left(\sum_{i=1}^k i\right)^2 + 2(k+1) \sum_{i=1}^k i + (k+1)^2
\end{aligned}$$

$$\begin{aligned}
\left(\sum_{i=1}^{k+1} i\right)^2 &= \left[\sum_{i=1}^k i + (k+1)\right]^2 \\
&= \left(\sum_{i=1}^k i\right)^2 + 2(k+1) \sum_{i=1}^k i + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + 2(k+1) \frac{1}{2} k(k+1) + (k+1)^2
\end{aligned}$$

$$\begin{aligned}
\left(\sum_{i=1}^{k+1} i\right)^2 &= \left[\sum_{i=1}^k i + (k+1)\right]^2 \\
&= \left(\sum_{i=1}^k i\right)^2 + 2(k+1) \sum_{i=1}^k i + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + 2(k+1) \frac{1}{2} k(k+1) + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + (k+1)^2 k + (k+1)^2
\end{aligned}$$

$$\begin{aligned}
\left(\sum_{i=1}^{k+1} i\right)^2 &= \left[\sum_{i=1}^k i + (k+1)\right]^2 \\
&= \left(\sum_{i=1}^k i\right)^2 + 2(k+1) \sum_{i=1}^k i + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + 2(k+1) \frac{1}{2} k(k+1) + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + (k+1)^2 k + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + (k+1)^2 (k+1)
\end{aligned}$$

$$\begin{aligned}
\left(\sum_{i=1}^{k+1} i\right)^2 &= \left[\sum_{i=1}^k i + (k+1)\right]^2 \\
&= \left(\sum_{i=1}^k i\right)^2 + 2(k+1) \sum_{i=1}^k i + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + 2(k+1) \frac{1}{2} k(k+1) + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + (k+1)^2 k + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + (k+1)^2 (k+1) \\
&= \sum_{i=1}^k i^3 + (k+1)^3
\end{aligned}$$

$$\begin{aligned}
\left(\sum_{i=1}^{k+1} i\right)^2 &= \left[\sum_{i=1}^k i + (k+1)\right]^2 \\
&= \left(\sum_{i=1}^k i\right)^2 + 2(k+1) \sum_{i=1}^k i + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + 2(k+1) \frac{1}{2} k(k+1) + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + (k+1)^2 k + (k+1)^2 \\
&= \sum_{i=1}^k i^3 + (k+1)^2 (k+1) \\
&= \sum_{i=1}^k i^3 + (k+1)^3 \\
&= \sum_{i=1}^{k+1} i^3
\end{aligned}$$

Hence Proved

The following algorithm adds the positive integers from 1 to n.

```
sum = 0;  
for i=1 to n do  
    sum = sum + i;  
endfor
```

$$T(n) = \sum_{i=1}^n c_2 + c_1 \quad T(n) = nc_2 + c_1$$

$$T(n) = \Theta(n)$$

The following algorithm obtains the sum given below for an arbitrary n:

$$1 \times 1 + 1 \times 2 + \cdots + 1 \times n + 2 \times 1 + 2 \times 2 + \cdots + 2 \times n + \cdots + n \times 1 + \cdots + n \times n$$

```
sum = 0
```

```
for i = 1 to n do
```

```
    for j = 1 to n do
```

```
        sum = sum + i * j
```

```
    endfor
```

```
endfor
```


$$T(n) = \sum_{i=1}^n \sum_{j=1}^n c_2 + c_1$$

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n c_2 + c_1$$

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n c_2 + c_1$$

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n c_2 + c_1$$

$$\begin{aligned} T(n) &= \sum_{i=1}^n \sum_{j=1}^n c_2 + c_1 \\ &= \sum_{i=1}^n c_2 n + c_1 \end{aligned}$$

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n c_2 + c_1$$

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n c_2 + c_1$$

$$= \sum_{i=1}^n c_2 n + c_1$$

$$= c_2 \sum_{i=1}^n n + c_1$$

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n c_2 + c_1$$

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n c_2 + c_1$$

$$= \sum_{i=1}^n c_2 n + c_1$$

$$= c_2 \sum_{i=1}^n n + c_1$$

$$= c_2 n^2 + c_1$$

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n c_2 + c_1$$

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n c_2 + c_1$$

$$= \sum_{i=1}^n c_2 n + c_1$$

$$= c_2 \sum_{i=1}^n n + c_1$$

$$= c_2 n^2 + c_1$$

$$= \Theta(n^2)$$

The following algorithm computes the sum of the following series:

$$1 \times 1 \times 1 + \dots + 1 \times 1 \times n + 2 \times 1 \times 1 + 2 \times 2 \times 2 + \dots + n \times n \times n$$

```
sum = 0
for i = 1 to n do
    for j = 1 to i do
        for k = 1 to j do
            sum = sum + i * j * k
        endfor
    endfor
endfor
```

$$T(n) = \sum_{i=1}^n \sum_{j=1}^i \sum_{k=1}^j c_2 + c_1$$

$$\begin{aligned}
T(n) &= \sum_{i=1}^n \sum_{j=1}^i \sum_{k=1}^j c_2 + c_1 \\
&= \sum_{i=1}^n \sum_{j=1}^i c_2 j + c_1
\end{aligned}$$

$$\begin{aligned}
T(n) &= \sum_{i=1}^n \sum_{j=1}^i \sum_{k=1}^j c_2 + c_1 \\
&= \sum_{i=1}^n \sum_{j=1}^i c_2 j + c_1 \\
&= c_2 \sum_{i=1}^n \sum_{j=1}^i j + c_1
\end{aligned}$$

$$\begin{aligned}
T(n) &= \sum_{i=1}^n \sum_{j=1}^i \sum_{k=1}^j c_2 + c_1 \\
&= \sum_{i=1}^n \sum_{j=1}^i c_2 j + c_1 \\
&= c_2 \sum_{i=1}^n \sum_{j=1}^i j + c_1 \\
&= c_2 \sum_{i=1}^n \frac{1}{2} i(i+1) + c_1
\end{aligned}$$

$$\begin{aligned}
T(n) &= \sum_{i=1}^n \sum_{j=1}^i \sum_{k=1}^j c_2 + c_1 \\
&= \sum_{i=1}^n \sum_{j=1}^i c_2 j + c_1 \\
&= c_2 \sum_{i=1}^n \sum_{j=1}^i j + c_1 \\
&= c_2 \sum_{i=1}^n \frac{1}{2} i(i+1) + c_1 \\
&= \frac{1}{2} c_2 \sum_{i=1}^n i(i+1) + c_1
\end{aligned}$$

$$= \frac{1}{2}c_2 \sum_{i=1}^n (i^2 + i) + c_1$$

$$= \frac{1}{2}c_2 \sum_{i=1}^n (i^2 + i) + c_1$$

$$= \frac{1}{2} \left(\sum_{i=1}^n i^2 + \sum_{i=1}^n i \right) c_2 + c_1$$

$$= \frac{1}{2}c_2 \sum_{i=1}^n (i^2 + i) + c_1$$

$$= \frac{1}{2} \left(\sum_{i=1}^n i^2 + \sum_{i=1}^n i \right) c_2 + c_1$$

$$= \frac{1}{2} \left(\frac{1}{6}n(n+1)(2n+1) + \frac{1}{2}n(n+1) \right) c_2 + c_1$$

$$\begin{aligned}
&= \frac{1}{2} c_2 \sum_{i=1}^n (i^2 + i) + c_1 \\
&= \frac{1}{2} \left(\sum_{i=1}^n i^2 + \sum_{i=1}^n i \right) c_2 + c_1 \\
&= \frac{1}{2} \left(\frac{1}{6} n(n+1)(2n+1) + \frac{1}{2} n(n+1) \right) c_2 + c_1 \\
&= \frac{1}{4} n(n+1) \left(\frac{1}{3} (2n+1) + 1 \right) c_2 + c_1
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{2}c_2 \sum_{i=1}^n (i^2 + i) + c_1 \\
&= \frac{1}{2} \left(\sum_{i=1}^n i^2 + \sum_{i=1}^n i \right) c_2 + c_1 \\
&= \frac{1}{2} \left(\frac{1}{6}n(n+1)(2n+1) + \frac{1}{2}n(n+1) \right) c_2 + c_1 \\
&= \frac{1}{4}n(n+1) \left(\frac{1}{3}(2n+1) + 1 \right) c_2 + c_1 \\
&= \frac{1}{4}n(n+1) \left(\frac{2n+1+3}{3} \right) c_2 + c_1
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{2}c_2 \sum_{i=1}^n (i^2 + i) + c_1 \\
&= \frac{1}{2} \left(\sum_{i=1}^n i^2 + \sum_{i=1}^n i \right) c_2 + c_1 \\
&= \frac{1}{2} \left(\frac{1}{6}n(n+1)(2n+1) + \frac{1}{2}n(n+1) \right) c_2 + c_1 \\
&= \frac{1}{4}n(n+1) \left(\frac{1}{3}(2n+1) + 1 \right) c_2 + c_1 \\
&= \frac{1}{4}n(n+1) \left(\frac{2n+1+3}{3} \right) c_2 + c_1 \\
&= \frac{1}{12}n(n+1)(2n+4)c_2 + c_1
\end{aligned}$$

$$= \frac{1}{6}n(n+1)(n+2)c_2 + c_1$$

$$\begin{aligned} &= \frac{1}{6}n(n+1)(n+2)c_2 + c_1 \\ &= \frac{1}{6}n(n^2 + 3n + 2)c_2 + c_1 \end{aligned}$$

$$= \frac{1}{6}n(n+1)(n+2)c_2 + c_1$$

$$= \frac{1}{6}n(n^2 + 3n + 2)c_2 + c_1$$

$$= \frac{1}{6}c_2n^3 + \frac{1}{2}c_2n^2 + \frac{1}{3}c_2n + c_1$$

$$= \Theta(n^3)$$

Obtain the sum of the following series:

$$1 + 2 + 2^2 + \dots + 2^k$$

Here,

a = the initial value = 1

r = the ratio between the two terms = 2

n = the number of terms = k - 1

Therefore, S_n can be obtained as given below.

$$\begin{aligned} S_n &= a(1 - r^k)/1 - r = \frac{1(1 - 2^k)}{1 - 2} \\ &= \frac{1 - 2^k}{-1} \\ &= 2^k - 1 \end{aligned}$$

```
m := 1;  
for i:= 1 to n do begin  
    m:= m * 2;  
    for j:= 1 to m do  
        {do something that is  $O(1)$ }  
end;
```

```
m := 1;  
for i:= 1 to n do begin  
    m:= m * 2;  
    for j:= 1 to m do  
        print "hello";  
end;
```



```

m := 1;
for i:= 1 to n do begin
    m:= m * 2;
    for j:= 1 to m do
        print “hello”;
end;

```

$$2 + 2^2 + \dots + 2^n = 2(1 + 2^1 + \dots + 2^{n-1}) = 2 * \sum_{i=0}^{n-1} 2^i$$

We know that $\sum_{k=0}^{n-1} x^k = 1 + x + x^2 + \dots + x^{n-1} = \frac{x^n - 1}{x - 1}$



$$2 * \left(\frac{2^n - 1}{2 - 1} \right) = 2 * (2^n - 1) = O(2^n)$$

Examples

- Nested Loops
- Sequential statements
- Conditional statements
- More nested loops

Nested Loops

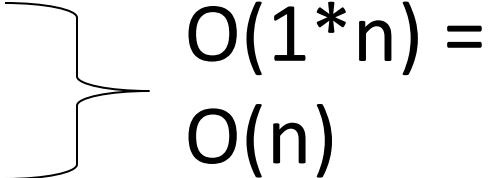
- Running time of a loop equals running time of code within the loop times the number of iterations
- Nested loops: analyze inside out

```
1 int k=0;
2 for ( int i=0; i<n; i++)
3     for ( int j=0; j<n; j++)  O(n)
4     k++;  O(1)
```

Nested Loops

- Running time of a loop equals running time of code within the loop times the number of iterations
- Nested loops: analyze inside out

```
1 int k=0;
2 for ( int i=0; i<n; i++)
3     for ( int j=0; j<n; j++)
4         k++;
```



$O(1*n) =$
 $O(n)$

Nested Loops

- Running time of a loop equals running time of code within the loop times the number of iterations
- Nested loops: analyze inside out

```
1 int k=0;
2 for ( int i=0; i<n; i++) _____ O(n)
3     for ( int j=0; j<n; j++) }
4         k++;                } O(n)
```

Nested Loops

- Running time of a loop equals running time of code within the loop times the number of iterations
- Nested loops: analyze inside out

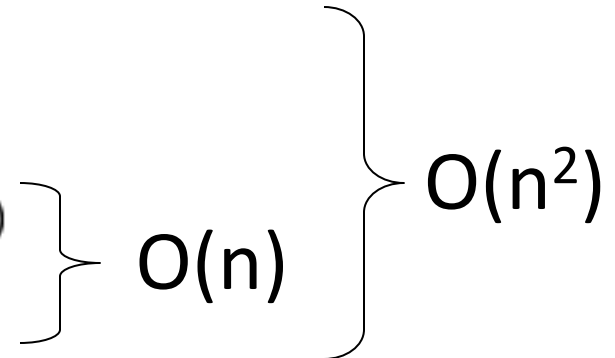
```
1 int k=0;  
2 for ( int i=0; i<n; i++)  
3     for ( int j=0; j<n; j++)  
4         k++;
```

} $O(n*n) =$
 $O(n^2)$

Nested Loops

- Running time of a loop equals running time of code within the loop times the number of iterations
- Nested loops: analyze inside out

```
1 int k=0;
2 for ( int i=0; i<n; i++)
3     for ( int j=0; j<n; j++)
4         k++;
```



$O(n)$

$O(n^2)$

- **Note:** Running time **grows** with **nesting** rather than the length of the code

Sequential Statements

- For a sequence $S_1; S_2; \dots; S_k$ of statements, running time is maximum of running times of individual statements

```
1 for ( int i=0; i<n; i++)  
2     X[i] = 0;  
3 for ( int i=0; i<n; i++)  
4     for ( int j=0; j<n; j++)  
5         X[i] += i+j;
```

$\left. \begin{array}{l} \text{Lines 1-2} \end{array} \right\} O(n)$

$\left. \begin{array}{l} \text{Lines 3-5} \end{array} \right\} O(n^2)$

Running time is: $\max(O(n), O(n^2)) = O(n^2)$

More Nested Loops

```
1 int k=0;  
2 for ( int i=0; i<n; i++)  
3     for ( int j=i; j<n; j++)  
4         k++;
```

$\left. \begin{array}{l} \text{for (int j=i; j<n; j++)} \\ \text{k++;} \end{array} \right\} n - i$?

$$\sum_{i=0}^{n-1} (n - i) = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = O(n^2)$$

What does the following algorithm do?

Analyze its worst-case running time, and express it using “Big-Oh” notation.

Algorithm Foo (a, n):

Input: two integers, a and n

Output: ?

$k \leftarrow 0$

$b \leftarrow 1$

while $k < n$ **do**

$k \leftarrow k + 1$

$b \leftarrow b * a$

return b

Algorithm Foo (a, n):

Input: two integers, a and n

Output: ?

$k \leftarrow 0$

$b \leftarrow 1$

while $k < n$ **do**

$k \leftarrow k + 1$

$b \leftarrow b * a$

return b

Solution

This algorithm computes a^n .

The running time
of this algorithm is **$O(n)$** because:

- the initial assignments take constant time
- each iteration of the **while** loop takes constant time
- there are exactly n iterations

Example : Bubblesort

```
Void Bubblesort( int[] A ) // A[1...n]
```

```
1. begin  
2. for i = 1 to n-1 do  
3.   for j=1 to n-i do  
4.     if A[j] > A[j+1] then  
5.       swap A[j] with A[j+1]  
6. end
```

Line 1,6: $O(1)$

Line 4,5: $O(1)$

Line 3-5: $O(n-i)$

$$\text{Line 2-5: } O\left(\sum_1^{n-1} (n-i)\right) = O\left(n(n-1) - \sum_1^{n-1} i\right) = O(n^2)$$

Example : Polynomial Growth

```
for k=1 to n do // pseudocode
  for j=1 to n do
    x = x + 1 // count this line or
               // count additions/assignments
```

$T(n)$ = “No. of additions for input size n ”

$$= k_{=1} + k_{=2} + \dots + k_{=n}$$

$$= n + n + \dots + n = nn = n^2$$

Example : Logarithmic Growth

```
1. k=n;  
2. while( k >= 1)    // top  
3.     x = x + 1;    // count this line  
4.     k = k / 2;    // k is halved  
5. end
```

Iteration#	value of k (at entry)	#line 3 exec'd
1	n	1
2	$n/2$	1
3	$n/2^2$	1
4	$n/2^3$	1
...
m-1	$n/2^{m-2}$	1
m	$n/2^{m-1} \geq 1$	1

Example : Logarithmic Growth (cont)

We are interested in what m is
(because that is the number of times line 3 is executed).
In other words,

$$T(n) = \underbrace{1 + 1 + 1 + \dots + 1}_m = (m * 1) \quad (\text{eq 1})$$

To derive m , we look at the last iteration,

$$\begin{aligned} 1 &\leq n/2^{m-1} < 2 \\ \Rightarrow 2^{m-1} &\leq n < 2^m \\ \Rightarrow (m-1) &\leq \log_2 n < m \\ \Rightarrow \lfloor \log_2 n \rfloor &= m-1 \\ \Rightarrow m &= \lfloor \log_2 n \rfloor + 1 \end{aligned}$$

From (eq 1),

$$T(n) = (\lfloor \log_2 n \rfloor + 1) * 1 = O(\lg n)$$

If $\log_2 n$ is between $m-1$ and m , then the **floor** of $\log_2 n$ is exactly $m-1$.

That's the definition of the floor function:

$\lfloor x \rfloor$ = the greatest integer less than or equal to x

Example : Insertion Sort

Unit Cost

(amount of work)

?

Times

?

$O(?)$

```
InsertionSort( int[] A )      // A is an n-element array
begin                          // Ignore function entry costs
    int i,j;                   // Ignore compile time costs1.
    for j=2 to length of A do
        key = A[j];
        i = j-1;
        while i>0 and A[i] > key do
            A[i+1] = A[i]
            i = i-1 ;
        endwhile              // ignore goto costs
        A[i+1] = key;
    endfor                     // ignore goto costs
end                             // ignore exit costs
```


Example : Portion of a selection sort which does the sorting

1	for (i= 0; i < n ; i++)	?
2	{	
3	m = i;	O(1)
4	for (j = i + 1; j <= n-1; j++)	?
5	{	
6	if (A[j] < A[m])	O(1)
7	m = j;	O(1)
8	}	
9	if (A[i] != A[m])	O(1)
10	{	
11	temp = A[i];	O(1)
12	A[i] = A[m];	O(1)
13	A[m] = temp;	O(1)
14	}	
15	}	

Basic Asymptotic Efficiency Classes

Class	Name	Comments
1	Constant	Algorithm ignores input (i.e., can't even scan input)

Basic Asymptotic Efficiency Classes

Class	Name	Comments
1	Constant	Algorithm ignores input (i.e., can't even scan input)
lgn	Logarithmic	Cuts problem size by constant fraction on each iteration

Basic Asymptotic Efficiency Classes

Class	Name	Comments
1	Constant	Algorithm ignores input (i.e., can't even scan input)
lgn	Logarithmic	Cuts problem size by constant fraction on each iteration
n	Linear	Algorithm scans its input (at least)

Basic Asymptotic Efficiency Classes

Class	Name	Comments
1	Constant	Algorithm ignores input (i.e., can't even scan input)
$\lg n$	Logarithmic	Cuts problem size by constant fraction on each iteration
n	Linear	Algorithm scans its input (at least)
$n \lg n$	"n-log-n"	Some divide and conquer

Basic Asymptotic Efficiency Classes

Class	Name	Comments
1	Constant	Algorithm ignores input (i.e., can't even scan input)
lgn	Logarithmic	Cuts problem size by constant fraction on each iteration
n	Linear	Algorithm scans its input (at least)
n lgn	"n-log-n"	Some divide and conquer
n ²	Quadratic	Loop inside loop = "nested loop"

Basic Asymptotic Efficiency Classes

Class	Name	Comments
1	Constant	Algorithm ignores input (i.e., can't even scan input)
$\lg n$	Logarithmic	Cuts problem size by constant fraction on each iteration
n	Linear	Algorithm scans its input (at least)
$n \lg n$	"n-log-n"	Some divide and conquer
n^2	Quadratic	Loop inside loop = "nested loop"
n^3	Cubic	Loop inside nested loop

Basic Asymptotic Efficiency Classes

Class	Name	Comments
1	Constant	Algorithm ignores input (i.e., can't even scan input)
$\lg n$	Logarithmic	Cuts problem size by constant fraction on each iteration
n	Linear	Algorithm scans its input (at least)
$n \lg n$	"n-log-n"	Some divide and conquer
n^2	Quadratic	Loop inside loop = "nested loop"
n^3	Cubic	Loop inside nested loop
2^n	Exponential	Algorithm generates all subsets of n-element set

Basic Asymptotic Efficiency Classes

Class	Name	Comments
1	Constant	Algorithm ignores input (i.e., can't even scan input)
$\lg n$	Logarithmic	Cuts problem size by constant fraction on each iteration
n	Linear	Algorithm scans its input (at least)
$n \lg n$	"n-log-n"	Some divide and conquer
n^2	Quadratic	Loop inside loop = "nested loop"
n^3	Cubic	Loop inside nested loop
2^n	Exponential	Algorithm generates all subsets of n-element set
$n!$	Factorial	Algorithm generates all permutations of n-element set

Recurrence Relation

The recurrence relation for an algorithm can be written as

$$T(n) = \begin{cases} T(n-1) + cn & n > 1 \\ d, & \text{otherwise} \end{cases}$$

The first equation says that the algorithm looks at all n elements in the input. c is a small positive constant. The $T(n-1)$ term on the right hand side says there is one fewer element to look at in the next round. Note that the coefficient of $T(n-1)$ is also 1. d is also a small positive constant.

$$T(n) = T(n - 1) + cn$$

$$T(n) = (T(n - 2) + c(n - 1)) + cn$$

$$T(n) = T(n - 2) + c((n - 1)) + n$$

$$T(n) = T(n - 3) + c((n - 1) + (n - 1) + n)$$

...

...

...

$$T(n) = T(n - k) + c((n - (k - 1)) + \cdots + (n - 1) + n)$$

$$= T(1) + c(2 + 3 + \cdots + (n - 1) + n)$$

$$= d + c(\textcolor{red}{1} + 2 + 3 + \cdots + (n - 1) + n - \textcolor{red}{1})$$

$$\begin{aligned}T(n) &= d + c\left(\frac{1}{2}n(n+1) - 1\right) \\T(n) &= \frac{1}{2}cn^2 + \frac{1}{2}cn + (d - c) \\&= O(n^2)\end{aligned}$$

$$T(n) = \begin{cases} T(n/2) + c & n > 1 \\ d, & \text{otherwise} \end{cases}$$

Here, c and d are small positive constants.

$$T(n) = T(n/2) + c$$

$$T(n) = \left[T\left(\frac{n}{2^2}\right) + c \right] + c$$

$$T(n) = \left[T\left(\frac{n}{2^3}\right) + c \right] + 2c$$

...

...

...

$$T(n) = \left[T\left(\frac{n}{2^k}\right) \right] + kc \quad \Rightarrow \quad = T(1) + kc$$

- We can assume that the number of elements we are dealing with is the next perfect power of 2.
- With this assumption, we will get an upper bound on the time consumed by the algorithm.
- With the assumption that n is a perfect power of 2, we can write $n = 2^k$, $k \geq 0$
- which leads us to the conclusion that $T(n) = \log_2 n$.

Solving the Recurrence

$$T(n) = \begin{cases} T(n/2) + cn & n > 1 \\ d, otherwise \end{cases}$$

$$T(n) = T\left(\frac{n}{2}\right) + c n$$

Solving the Recurrence

$$T(n) = \begin{cases} T(n/2) + cn & n > 1 \\ d, otherwise \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + c n \\ &= \left[T\left(\frac{n}{2^2}\right) + c \frac{n}{2} \right] + c n \end{aligned}$$

Solving the Recurrence

$$T(n) = \begin{cases} T(n/2) + cn & n > 1 \\ d, otherwise \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + c n \\ &= \left[T\left(\frac{n}{2^2}\right) + c \frac{n}{2} \right] + c n \\ &= T\left(\frac{n}{2^2}\right) + c n \left(\frac{1}{2} + 2 \right) \end{aligned}$$

Solving the Recurrence

$$T(n) = \begin{cases} T(n/2) + cn & n > 1 \\ d, otherwise \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + cn \\ &= \left[T\left(\frac{n}{2^2}\right) + c\frac{n}{2} \right] + cn \\ &= T\left(\frac{n}{2^2}\right) + cn\left(\frac{1}{2} + 2\right) \\ &= \left[T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2} \right] + cn\left(\frac{1}{2} + 1\right) \end{aligned}$$

Solving the Recurrence

$$T(n) = \begin{cases} T(n/2) + cn & n > 1 \\ d, otherwise \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + cn \\ &= \left[T\left(\frac{n}{2^2}\right) + c\frac{n}{2} \right] + cn \\ &= T\left(\frac{n}{2^2}\right) + cn \left(\frac{1}{2} + 2 \right) \\ &= \left[T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2} \right] + cn \left(\frac{1}{2} + 1 \right) \\ &= T\left(\frac{n}{2^3}\right) + cn \left(\frac{1}{2^2} + \frac{1}{2} + 1 \right) \end{aligned}$$

Solving the Recurrence

$$T(n) = \begin{cases} T(n/2) + cn & n > 1 \\ d, otherwise \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + cn \\ &= \left[T\left(\frac{n}{2^2}\right) + c\frac{n}{2} \right] + cn \\ &= T\left(\frac{n}{2^2}\right) + cn\left(\frac{1}{2} + 2\right) \\ &= \left[T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2} \right] + cn\left(\frac{1}{2} + 1\right) \\ &= T\left(\frac{n}{2^3}\right) + cn\left(\frac{1}{2^2} + \frac{1}{2} + 1\right) \\ &\vdots \end{aligned}$$

$$T(n) = T\left(\frac{n}{2^k}\right) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1\right)$$

$$T(n) = T\left(\frac{n}{2^k}\right) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right)$$

$$T(n) = T(1) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right)$$

$$T(n) = T\left(\frac{n}{2^k}\right) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right)$$

$$\begin{aligned} T(n) &= T(1) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right) \\ &= d + c n \frac{1 - \frac{1}{2^k}}{1 - \frac{1}{2}} \end{aligned}$$

$$T(n) = T\left(\frac{n}{2^k}\right) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right)$$

$$\begin{aligned} T(n) &= T(1) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right) \\ &= d + c n \frac{1 - \frac{1}{2^k}}{1 - \frac{1}{2}} \\ &= d + 2 c n \left(1 - \frac{1}{2^k} \right) \end{aligned}$$

$$T(n) = T\left(\frac{n}{2^k}\right) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right)$$

$$T(n) = T(1) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right)$$

$$= d + c n \frac{1 - \frac{1}{2^k}}{1 - \frac{1}{2}}$$

$$= d + 2 c n \left(1 - \frac{1}{2^k} \right)$$

$$= d + 2 c n \left(1 - \frac{1}{n} \right)$$

$$T(n) = T\left(\frac{n}{2^k}\right) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right)$$

$$T(n) = T(1) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right)$$

$$= d + c n \frac{1 - \frac{1}{2^k}}{1 - \frac{1}{2}}$$

$$= d + 2 c n \left(1 - \frac{1}{2^k} \right)$$

$$= d + 2 c n \left(1 - \frac{1}{n} \right)$$

$$= d + 2 c n - 2 c$$

$$T(n) = T\left(\frac{n}{2^k}\right) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right)$$

$$T(n) = T(1) + c n \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1 \right)$$

$$= d + c n \frac{1 - \frac{1}{2^k}}{1 - \frac{1}{2}}$$

$$= d + 2 c n \left(1 - \frac{1}{2^k} \right)$$

$$= d + 2 c n \left(1 - \frac{1}{n} \right)$$

$$= d + 2 c n - 2 c$$

$$= 2 c n + (d - 2c)$$

$$= O(n)$$

a = the initial value = 1

r = the ratio between the two terms = $\frac{1}{2}$

n = the number of terms = k - 1

Therefore,

$$S_n = a(1 - r^k)/1 - r$$

Solving the Recurrence

$$T(n) = 2 T\left(\frac{n}{2}\right) + c n \quad n > 1$$

$$T(1) = d$$

$$T(n) = 2 T \left(\frac{n}{2} \right) + c n \qquad n > 1$$

$$T(1) = d$$

$$T(n) = 2 T \left(\frac{n}{2} \right) + cn$$

$$T(n) = 2 T \left(\frac{n}{2} \right) + c n \qquad n > 1$$

$$T(1) = d$$

$$\begin{aligned} T(n) &= 2 T \left(\frac{n}{2} \right) + cn \\ &= 2 \left[2 T \left(\frac{n}{2^2} \right) + c \frac{n}{2} \right] + cn \end{aligned}$$

$$T(n) = 2 T \left(\frac{n}{2} \right) + c n \qquad n > 1$$

$$T(1) = d$$

$$\begin{aligned} T(n) &= 2 T \left(\frac{n}{2} \right) + cn \\ &= 2 \left[2 T \left(\frac{n}{2^2} \right) + c \frac{n}{2} \right] + cn \\ &= 2^2 T \left(\frac{n}{2^2} \right) + cn + cn \end{aligned}$$

$$T(n) = 2 T \left(\frac{n}{2} \right) + c n \quad n > 1$$

$$T(1) = d$$

$$\begin{aligned} T(n) &= 2 T \left(\frac{n}{2} \right) + cn \\ &= 2 \left[2 T \left(\frac{n}{2^2} \right) + c \frac{n}{2} \right] + cn \\ &= 2^2 T \left(\frac{n}{2^2} \right) + cn + cn \\ &= 2^2 T \left(\frac{n}{2^2} \right) + 2cn \end{aligned}$$

$$T(n) = 2 T \left(\frac{n}{2} \right) + c n \quad n > 1$$

$$T(1) = d$$

$$\begin{aligned} T(n) &= 2 T \left(\frac{n}{2} \right) + cn \\ &= 2 \left[2 T \left(\frac{n}{2^2} \right) + c \frac{n}{2} \right] + cn \\ &= 2^2 T \left(\frac{n}{2^2} \right) + cn + cn \\ &= 2^2 T \left(\frac{n}{2^2} \right) + 2cn \\ &= 2^2 \left[2 T \left(\frac{n}{2^3} \right) + c \frac{n}{2^2} \right] + 2cn \end{aligned}$$

$$T(n) = 2 T \left(\frac{n}{2} \right) + c n \quad n > 1$$

$$T(1) = d$$

$$\begin{aligned} T(n) &= 2 T \left(\frac{n}{2} \right) + cn \\ &= 2 \left[2 T \left(\frac{n}{2^2} \right) + c \frac{n}{2} \right] + cn \\ &= 2^2 T \left(\frac{n}{2^2} \right) + cn + cn \\ &= 2^2 T \left(\frac{n}{2^2} \right) + 2cn \\ &= 2^2 \left[2 T \left(\frac{n}{2^3} \right) + c \frac{n}{2^2} \right] + 2cn \\ &= 2^3 T \left(\frac{n}{2^3} \right) + cn + 2cn \end{aligned}$$

$$T(n) = 2 T \left(\frac{n}{2} \right) + c n \quad n > 1$$

$$T(1) = d$$

$$\begin{aligned} T(n) &= 2 T \left(\frac{n}{2} \right) + cn \\ &= 2 \left[2 T \left(\frac{n}{2^2} \right) + c \frac{n}{2} \right] + cn \\ &= 2^2 T \left(\frac{n}{2^2} \right) + cn + cn \\ &= 2^2 T \left(\frac{n}{2^2} \right) + 2cn \\ &= 2^2 \left[2 T \left(\frac{n}{2^3} \right) + c \frac{n}{2^2} \right] + 2cn \\ &= 2^3 T \left(\frac{n}{2^3} \right) + cn + 2cn \\ &= 2^3 T \left(\frac{n}{2^3} \right) + 3cn \end{aligned}$$

$$T(n) = 2 T \left(\frac{n}{2} \right) + c n \quad n > 1$$

$$T(1) = d$$

$$\begin{aligned}
 T(n) &= 2 T \left(\frac{n}{2} \right) + cn \\
 &= 2 \left[2 T \left(\frac{n}{2^2} \right) + c \frac{n}{2} \right] + cn \\
 &= 2^2 T \left(\frac{n}{2^2} \right) + cn + cn \\
 &= 2^2 T \left(\frac{n}{2^2} \right) + 2cn \\
 &= 2^2 \left[2 T \left(\frac{n}{2^3} \right) + c \frac{n}{2^2} \right] + 2cn \\
 &= 2^3 T \left(\frac{n}{2^3} \right) + cn + 2cn \\
 &= 2^3 T \left(\frac{n}{2^3} \right) + 3cn \\
 &\vdots
 \end{aligned}$$

$$= 2^k T\left(\frac{n}{2^k}\right) + k c n$$

$$= 2^k T\left(\frac{n}{2^k}\right) + k c n$$

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

$$= 2^k T\left(\frac{n}{2^k}\right) + k c n$$

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

$$= nd + c n \log_2 n$$

$$\begin{aligned}
&= 2^k T\left(\frac{n}{2^k}\right) + k c n \\
&= 2^k T(1) + k c n \\
&= nd + c n \log_2 n \\
&= \Theta(n \log_2 n)
\end{aligned}$$

Solving the Recurrence

$$T(n) = 3 T\left(\frac{n}{2}\right) + c$$

Solving the Recurrence

$$T(n) = 3 T \left(\frac{n}{2} \right) + c$$

$$T(n) = 3 T \left(\frac{n}{2} \right) + c \quad n > 1$$

$$T(1) = d$$

Solving the Recurrence $T(n) = 3 T \left(\frac{n}{2} \right) + c$

$$T(n) = 3 T \left(\frac{n}{2} \right) + c \quad n > 1$$

$$T(1) = d$$

$$T(n) = 3 T \left(\frac{n}{2} \right) + c$$

Solving the Recurrence $T(n) = 3 T \left(\frac{n}{2} \right) + c$

$$T(n) = 3 T \left(\frac{n}{2} \right) + c \quad n > 1$$

$$T(1) = d$$

$$\begin{aligned} T(n) &= 3 T \left(\frac{n}{2} \right) + c \\ &= 3 \left[3 T \left(\frac{n}{2^2} \right) + c \right] + c \end{aligned}$$

Solving the Recurrence

$$T(n) = 3 T \left(\frac{n}{2} \right) + c$$

$$T(n) = 3 T \left(\frac{n}{2} \right) + c \quad n > 1$$

$$T(1) = d$$

$$\begin{aligned} T(n) &= 3 T \left(\frac{n}{2} \right) + c \\ &= 3 \left[3 T \left(\frac{n}{2^2} \right) + c \right] + c \\ &= 3^2 T \left(\frac{n}{2^2} \right) + (3 + 1)c \end{aligned}$$

Solving the Recurrence $T(n) = 3 T \left(\frac{n}{2} \right) + c$

$$T(n) = 3 T \left(\frac{n}{2} \right) + c \quad n > 1$$

$$T(1) = d$$

$$\begin{aligned} T(n) &= 3 T \left(\frac{n}{2} \right) + c \\ &= 3 \left[3 T \left(\frac{n}{2^2} \right) + c \right] + c \\ &= 3^2 T \left(\frac{n}{2^2} \right) + (3 + 1)c \\ &= 3^2 \left[3 T \left(\frac{n}{2^3} \right) + c \right] + (3 + 1)c \end{aligned}$$

Solving the Recurrence $T(n) = 3 T\left(\frac{n}{2}\right) + c$

$$T(n) = 3 T\left(\frac{n}{2}\right) + c \quad n > 1$$

$$T(1) = d$$

$$\begin{aligned} T(n) &= 3 T\left(\frac{n}{2}\right) + c \\ &= 3 \left[3 T\left(\frac{n}{2^2}\right) + c \right] + c \\ &= 3^2 T\left(\frac{n}{2^2}\right) + (3 + 1)c \\ &= 3^2 \left[3 T\left(\frac{n}{2^3}\right) + c \right] + (3 + 1)c \\ &= 3^3 T\left(\frac{n}{2^3}\right) + (3^2 + 3 + 1)c \end{aligned}$$

Solving the Recurrence

$$T(n) = 3 T\left(\frac{n}{2}\right) + c$$

$$\begin{aligned} & \vdots \\ &= 3^k T\left(\frac{n}{2^k}\right) + \left(3^{k-1} + 3^{k-2} + \dots + 3 + 1\right) c \\ &= 3^k T(1) + c \frac{3^k - 1}{3 - 1} \\ &= 3^k d + \frac{1}{2} c \left(3^k - 1\right) \\ &= \left(d + \frac{1}{2} c\right) 3^k - \frac{1}{2} c \\ &= b 3^k - \frac{1}{2} c \end{aligned}$$

We made the assumption $n = 2^k$. This gives

$$k = \log_2 n$$

$$= \log_3 n \log_2 3$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$3^k = 3^{\log_3 n \log_2 3}$$

$$= \left(3^{\log_3 n}\right)^{\log_2 3}$$

$$= n^{\log_2 3}$$

$$= n^{1.58496}$$

$$T(n) = b n^{1.58496} - \frac{1}{2}c$$

Assignment: Solve the recurrence relation:

$$T(n) = a T\left(\frac{n}{b}\right) + c \quad n \geq 1$$

$$T(1) = d$$

where a and b are positive integers, and $a > b$,

Solving the Recurrence

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c & n > 2 \\ &= d & n = 1, 2 \end{aligned}$$

$$T(n) = T(\sqrt{n}) + c$$

Solving the Recurrence

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c & n > 2 \\ &= d & n = 1, 2 \end{aligned}$$

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c \\ &= T(n^{\frac{1}{2}}) + c \end{aligned}$$

Solving the Recurrence

$$\begin{aligned}T(n) &= T(\sqrt{n}) + c & n > 2 \\ &= d & n = 1, 2\end{aligned}$$

$$\begin{aligned}T(n) &= T(\sqrt{n}) + c \\ &= T\left(n^{\frac{1}{2}}\right) + c \\ &= \left[T\left(n^{\frac{1}{2^2}}\right) + c\right] + c\end{aligned}$$

Solving the Recurrence

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c & n > 2 \\ &= d & n = 1, 2 \end{aligned}$$

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c \\ &= T\left(n^{\frac{1}{2}}\right) + c \\ &= \left[T\left(n^{\frac{1}{2^2}}\right) + c\right] + c \\ &= T\left(n^{\frac{1}{2^2}}\right) + 2c \end{aligned}$$

Solving the Recurrence

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c & n > 2 \\ &= d & n = 1, 2 \end{aligned}$$

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c \\ &= T\left(n^{\frac{1}{2}}\right) + c \\ &= \left[T\left(n^{\frac{1}{2^2}}\right) + c\right] + c \\ &= T\left(n^{\frac{1}{2^2}}\right) + 2c \\ &= \left[T\left(n^{\frac{1}{2^3}}\right) + c\right] + 2c \end{aligned}$$

Solving the Recurrence

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c & n > 2 \\ &= d & n = 1, 2 \end{aligned}$$

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c \\ &= T\left(n^{\frac{1}{2}}\right) + c \\ &= \left[T\left(n^{\frac{1}{2^2}}\right) + c\right] + c \\ &= T\left(n^{\frac{1}{2^2}}\right) + 2c \\ &= \left[T\left(n^{\frac{1}{2^3}}\right) + c\right] + 2c \end{aligned}$$

Solving the Recurrence

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c & n > 2 \\ &= d & n = 1, 2 \end{aligned}$$

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c \\ &= T\left(n^{\frac{1}{2}}\right) + c \\ &= \left[T\left(n^{\frac{1}{2^2}}\right) + c\right] + c \\ &= T\left(n^{\frac{1}{2^2}}\right) + 2c \\ &= \left[T\left(n^{\frac{1}{2^3}}\right) + c\right] + 2c \\ &= T\left(n^{\frac{1}{2^3}}\right) + 3c \end{aligned}$$

Solving the Recurrence

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c & n > 2 \\ &= d & n = 1, 2 \end{aligned}$$

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c \\ &= T\left(n^{\frac{1}{2}}\right) + c \\ &= \left[T\left(n^{\frac{1}{2^2}}\right) + c\right] + c \\ &= T\left(n^{\frac{1}{2^2}}\right) + 2c \\ &= \left[T\left(n^{\frac{1}{2^3}}\right) + c\right] + 2c \\ &= T\left(n^{\frac{1}{2^3}}\right) + 3c \\ &\vdots \end{aligned}$$

$$= T\left(n^{\frac{1}{2^k}}\right) + kc$$

$$\begin{aligned}
&= T\left(n^{\frac{1}{2^k}}\right) + kc \\
&= T(2) + kc
\end{aligned}$$

$$\begin{aligned}
&= T\left(n^{\frac{1}{2^k}}\right) + kc \\
&= T(2) + kc \\
&= d + c \log_2 \log_2 n
\end{aligned}$$

$$\begin{aligned} &= T\left(n^{\frac{1}{2^k}}\right) + kc \\ &= T(2) + kc \\ &= d + c \log_2 \log_2 n \\ &= \Theta(\log_2 \log_2 n) \end{aligned}$$

$$\begin{aligned}
&= T\left(n^{\frac{1}{2^k}}\right) + kc \\
&= T(2) + kc \\
&= d + c \log_2 \log_2 n \\
&= \Theta(\log_2 \log_2 n)
\end{aligned}$$

$$n^{\frac{1}{2^k}} = 2$$

$$\begin{aligned}
&= T\left(n^{\frac{1}{2^k}}\right) + kc \\
&= T(2) + kc \\
&= d + c \log_2 \log_2 n \\
&= \Theta(\log_2 \log_2 n)
\end{aligned}$$

$$n^{\frac{1}{2^k}} = 2$$

$$n^{\frac{1}{2^k}} = 2$$

$$\begin{aligned}
&= T\left(n^{\frac{1}{2^k}}\right) + kc \\
&= T(2) + kc \\
&= d + c \log_2 \log_2 n \\
&= \Theta(\log_2 \log_2 n)
\end{aligned}$$

$$n^{\frac{1}{2^k}} = 2$$

$$n^{\frac{1}{2^k}} = 2$$

$$\Rightarrow \sqrt[2^k]{n} = 2$$

$$\begin{aligned}
&= T\left(n^{\frac{1}{2^k}}\right) + kc \\
&= T(2) + kc \\
&= d + c \log_2 \log_2 n \\
&= \Theta(\log_2 \log_2 n)
\end{aligned}$$

$$n^{\frac{1}{2^k}} = 2$$

$$n^{\frac{1}{2^k}} = 2$$

$$\Rightarrow \sqrt[2^k]{n} = 2$$

$$\Rightarrow n = 2^{2^k}$$

$$\begin{aligned}
&= T\left(n^{\frac{1}{2^k}}\right) + kc \\
&= T(2) + kc \\
&= d + c \log_2 \log_2 n \\
&= \Theta(\log_2 \log_2 n)
\end{aligned}$$

$$n^{\frac{1}{2^k}} = 2$$

$$n^{\frac{1}{2^k}} = 2$$

$$\Rightarrow \sqrt[2^k]{n} = 2$$

$$\Rightarrow n = 2^{2^k}$$

$$\Rightarrow 2^k = \log_2 n$$

$$\begin{aligned}
&= T\left(n^{\frac{1}{2^k}}\right) + kc \\
&= T(2) + kc \\
&= d + c \log_2 \log_2 n \\
&= \Theta(\log_2 \log_2 n)
\end{aligned}$$

$$n^{\frac{1}{2^k}} = 2$$

$$n^{\frac{1}{2^k}} = 2$$

$$\Rightarrow \sqrt[2^k]{n} = 2$$

$$\Rightarrow n = 2^{2^k}$$

$$\Rightarrow 2^k = \log_2 n$$

$$\Rightarrow k = \log_2 \log_2 n$$

Ackermann's Function

$$A(1, j) = 2^j \text{ for } j \geq 1$$

$$A(i, 1) = A(i - 1, 2) \text{ for } i \geq 2$$

$$A(i, j) = A(i - 1, A(i, j - 1)) \text{ for } i, j \geq 2$$

A(i,j)	j=1	j=2	j=3	j=4
i=1	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
i=2	$2^2 = 4$	$2^{2^2} = 16$	$2^{16} = 65536$	2^{65536}
i=3	$2^{2^2} = 16$	$2^{16} = 65536$	2^{65536}	$2^{2^{65536}} = \text{BIG}$

Master's theorem

- Let $T(n)$ be a monotonically increasing function that satisfies

- $T(n) = a T\left(\frac{n}{b}\right) + n^d$

- $T(1) = c$

- Where $a \geq 1$, $b \geq 2$, $c \geq 0$, $d \geq 1$ then

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

Master's theorem Ex. 1

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

$$- T(n) = 2 T\left(\frac{n}{2}\right) + n$$

- Here $a = 2, b = 2, d = 1$

- This is $a = b^d$ form

- $\Rightarrow T(n) = O(n \log n)$

Master's theorem Ex. 2

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

$$- T(n) = 3 T\left(\frac{n}{2}\right) + n$$

- Here $a = 3, b = 2, d = 1$
- This is $a > b^d$ form
- $\Rightarrow T(n) = O(n^{\log_2 3})$

Master's theorem Ex. 3

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

$$- T(n) = T\left(\frac{n}{2}\right) + n$$

- Here $a = 1, b = 2, d = 1$
- This is $a < b^d$ form
- $\Rightarrow T(n) = O(n)$

Master's theorem Ex. 4

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

$$- T(n) = 8T\left(\frac{n}{2}\right) + n^2$$

- Here $a = 8, b = 2, d = 2$
- This is $a > b^d$ form
- $\Rightarrow T(n) = O(n^{\log_2 8}) = O(n^3)$

Akra-Bazzi method

- Master's method is unable to handle certain cases.
- The Akra-Bazzi method is **very flexible** and allows us to solve recurrences that are **not neatly handled by the Master Theorem**, especially when
 - the recursive divisions are uneven or
 - when the additive term $g(n)$ is not polynomial.

Cont.

- The **Akra-Bazzi method** is a **powerful technique** used to determine the **asymptotic (time) complexity of divide-and-conquer recurrence relations** of the form:
- $T(x) = \sum_{i=1}^k a_i T(b_i x) + g(x)$
 - Where $a_i > 0$,
 - $0 < b_i < 1$,
 - $g(x)$ is a non-negative function that describes the cost outside the recursive calls
- This method **generalizes the Master Theorem** to more complex or irregular cases.

Cont.

- Suppose the recurrence is
- $T(x) = \sum_{i=1}^k a_i T(b_i x) + g(x), x > x_0$
- Then, under technical conditions (which are typically satisfied in practice), the solution is:
- $T(x) = \Theta \left(x^p \left(1 + \int_1^x \frac{g(u)}{u^{p+1}} du \right) \right)$
- Where: p is the **unique solution** of the equation:
- $\sum_{i=1}^k a_i b_i^p = 1$

Example: Use Akra-Bazzi to solve a recurrence

- Let's solve:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

- This recurrence is not solvable directly by Master's Theorem because of the non-polynomial $f(n)=n \log n$. So we apply Akra-Bazzi.
- $T(n) = \sum_{i=1}^n a_i T(b_i n) + f(n)$
- under certain conditions (smoothness of $f(n)$, and constants satisfying $0 < b_i < 1$, $a_i > 0$), the solution is:

Cont.

- $T(x) = \Theta \left(x^p \left(1 + \int_1^x \frac{g(u)}{u^{p+1}} du \right) \right)$
- Where: p is the **unique solution** of the equation:
- $\sum_{i=1}^k a_i b_i^p = 1$

Step 1: Identify parameters

- From $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$
- We have $a_1 = 2$, $b_1 = \frac{1}{2}$, $f(n) = n \log n$

Step 2: Solve for p

- $\sum_{i=1}^k a_i b_i^p = 1 \Rightarrow 2 \left(\frac{1}{2}\right)^p = 1$
- $\Rightarrow \left(\frac{1}{2}\right)^p = \frac{1}{2} \Rightarrow p = 1$

Step 3: Plug into Akra-Bazzi formula

- $T(x) = \Theta \left(n^1 \left(1 + \int_1^n \frac{u \log u}{u^{1+1}} du \right) \right)$
- $= \Theta \left(n \left(1 + \int_1^n \frac{\log u}{u} du \right) \right)$

Step 4: Compute the integral

- $= \Theta \left(n \left(1 + \int_1^n \frac{\log u}{u} du \right) \right)$
- $\int_1^n \frac{\log u}{u} du = \frac{1}{2} (\log n)^2$ (This is a standard integral result.)
- Let $t = \log u$, so $dt = \frac{1}{u} du$
- Change the limit, when $u = 1$, $t = \log 1 = 0$
- when $u = n$, $t = \log n$
- $I = \int_0^{\log n} t dt = \left[\frac{t^2}{2} \right]_0^{\log n} = \frac{1}{2} (\log n)^2$

Step 5: Final Result

- $T(n) = \Theta\left(n \left(1 + \frac{1}{2} (\log n)^2\right)\right)$
- $T(n) = \Theta(n (\log n)^2)$