Complexity Analysis

To measure and compare growth of resource usage of the algorithm with respect to size of input N. We focus our measurement on long time trend (large input) by counting major primitive instruction as a function of N.

Well-known summations

- $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=0}^{n} a^i = \frac{1}{1-a}$ when $a < 1$
- $\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}$
- $\sum_{i=0}^{n} log(i) = log(n!)$

Asymptotic Notation

- $O(g(n))$ Set of all functions $T(n)$ that grows not faster than $g(n)$ Upper bound
- $\Omega(g(n))$ Set of all functions $T(n)$ that grows not slower than $g(n)$ Lower bound
- $\Theta(g(n))$ Set of all functions $T(n)$ that grows equal to than $g(n)$ Tight bound
- $o(g(n))$ Set of all functions $T(n)$ that grows faster than $g(n)$
- $\omega(g(n))$ Set of all functions $T(n)$ that grows slower than $g(n)$

Recursive Program

> *Function that call itself, which has 2 parts:*

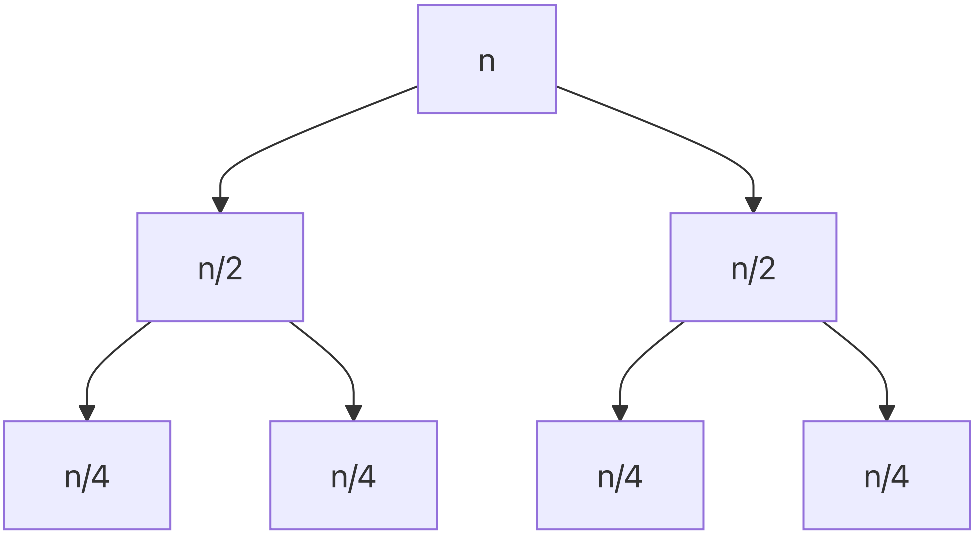
1. Terminal case (trivial case) the case that has no recursion.
2. Recursive case the case that call itself.

Example:

```
int add(int n) {
    if (n == 0) return n; // terminal case
    return n + add(n-1); // recursive case
}
```

Recursion Tree

> *A tree to help understanding recursive program*



- A node is each function call
  - Describe a parameter of a function in the node
- An edge is a function call
  - Children of each node is a function that were called by this node

Master Theorem

> *Shortcut in solving some recurrent relation in this form:*

$T(n) = aT(n/b) + f(n)$ with following conditions

1. $a \geq 1$
2. $b > 1$
3. $T(0) = 1$

$$T(n) = \begin{cases} \Theta(n^c) & ; f(n) = O(n^{c-\epsilon}) \\ \Theta(n^c log^{k+1} n) & ; f(n) = \Theta(n^c log^k n) \\ \Theta(n^d) & ; f(n) = \Omega(n^{c+\epsilon}) \end{cases}$$

where $af(a/b) \leq kf(n), k < 1, n > n_0$ and $c = log_b a$

Example:

```
void merge_sort(int l, int r) {
    if (l==r) return;
    int m = (l+r) / 2;

    merge_sort(l, m);
    merge_sort(m+1, r);
    merge(l, r);
}
```

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

Case: $\Theta(n^c log^{k+1} n)$ then $T(n) = \Theta(n^1 log^{(0+1)} n) = \Theta(n\ log\ n)$