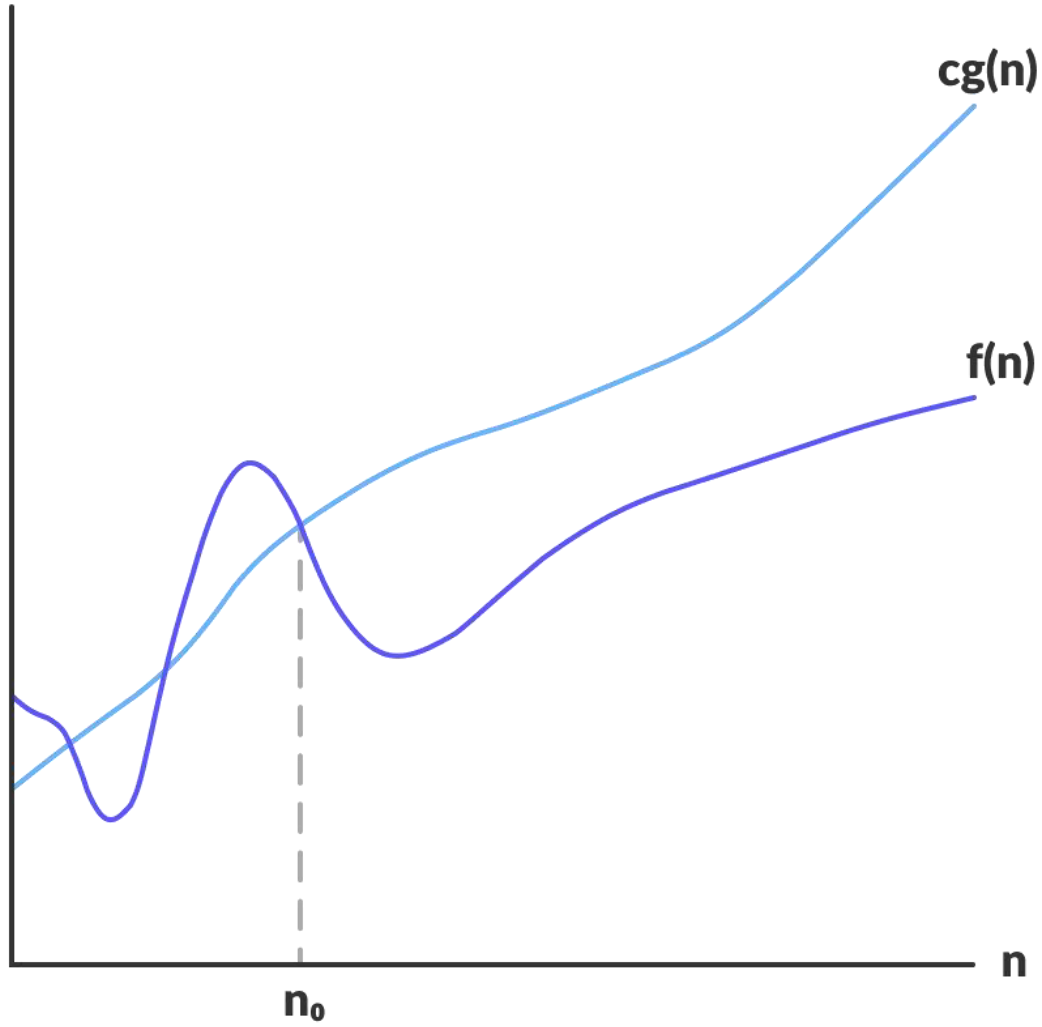
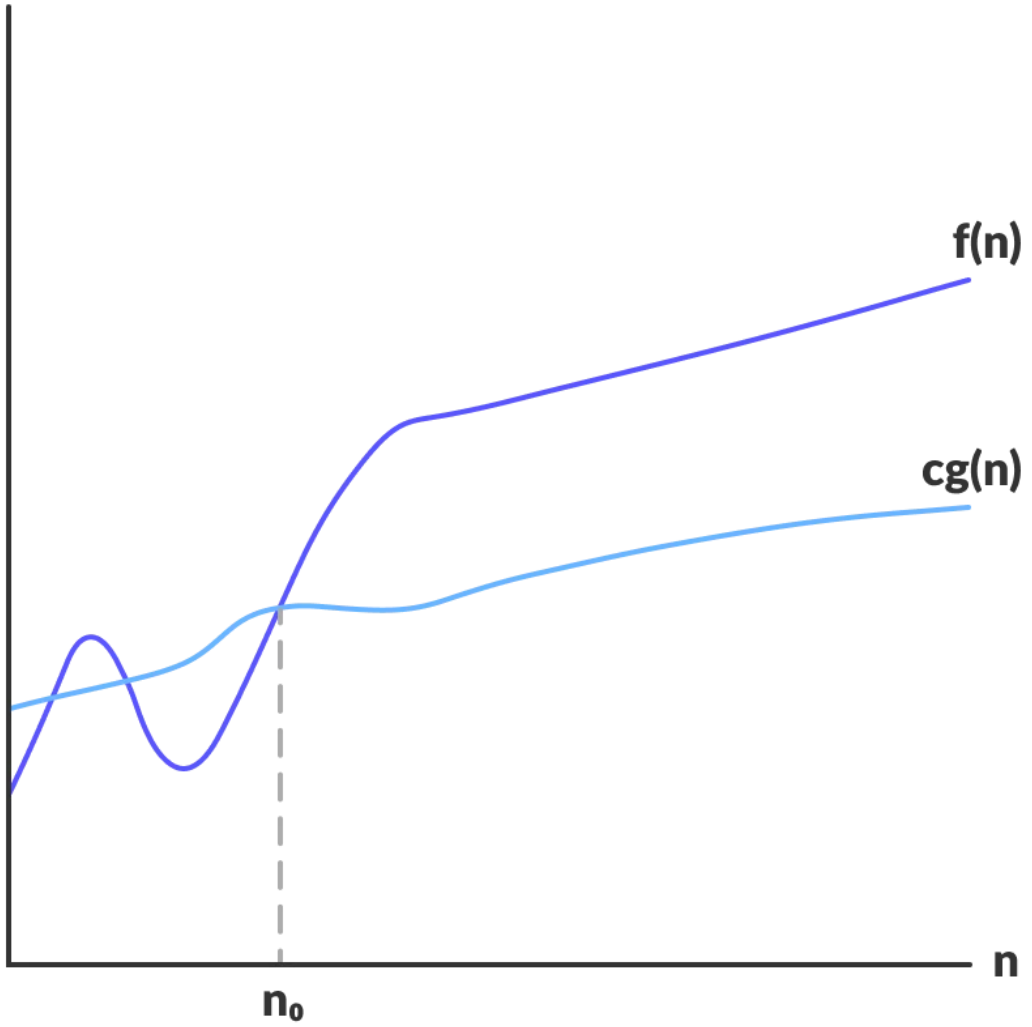
Design and Analysis of Algorithms

Asymptotic Notations: O-notation



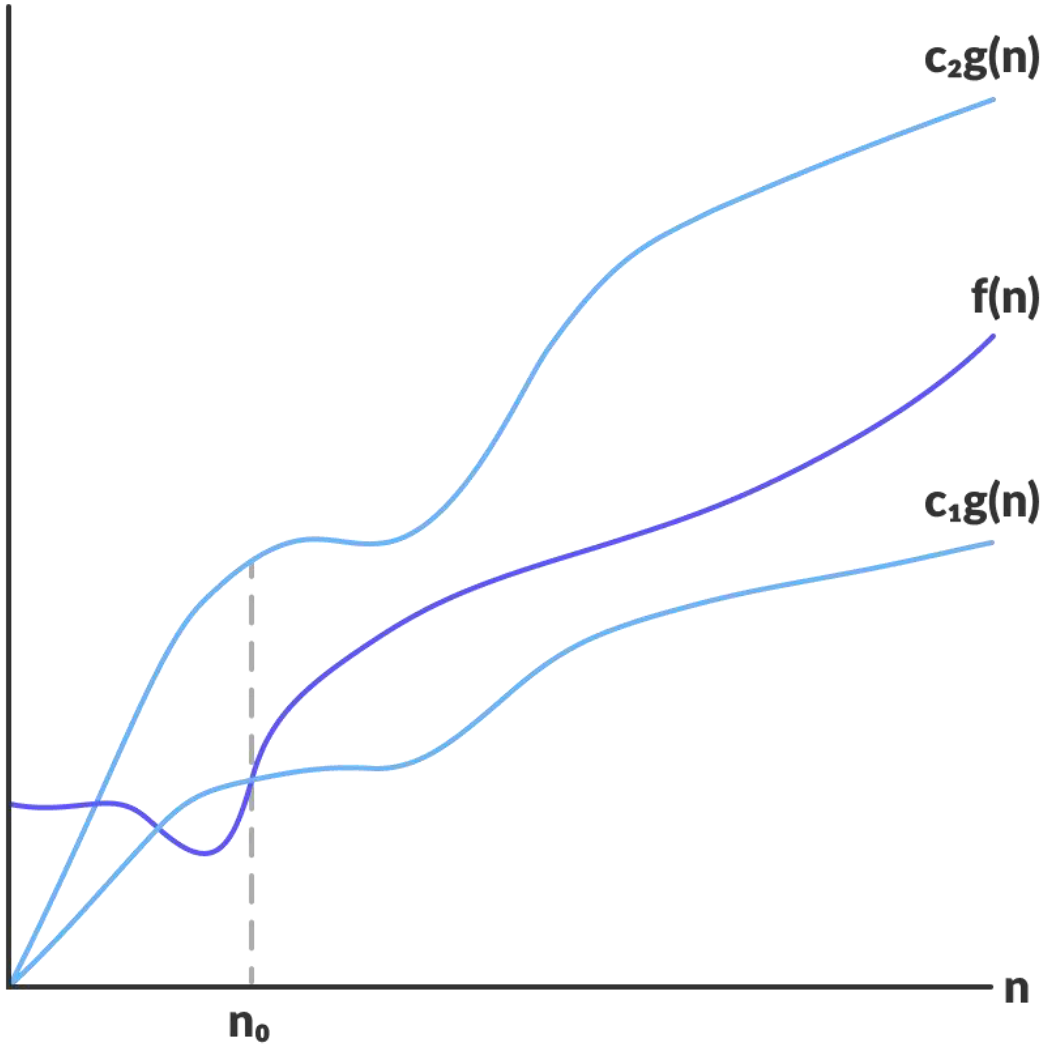
f(n) = O(g(n))

Asymptotic Notations: Ω-notation



$f(n) = \Omega(g(n))$

Asymptotic Notations: Θ-notation



$f(n) = \Theta(g(n))$

Asymptotic Notations

$$T(n) \in O(g(n)) \quad \text{is like} \quad b \geq a$$

$$T(n) \in \Omega(g(n)) \quad \text{is like} \quad b \leq a$$

$$T(n) \in \Theta(g(n)) \quad \text{is like} \quad b = a$$

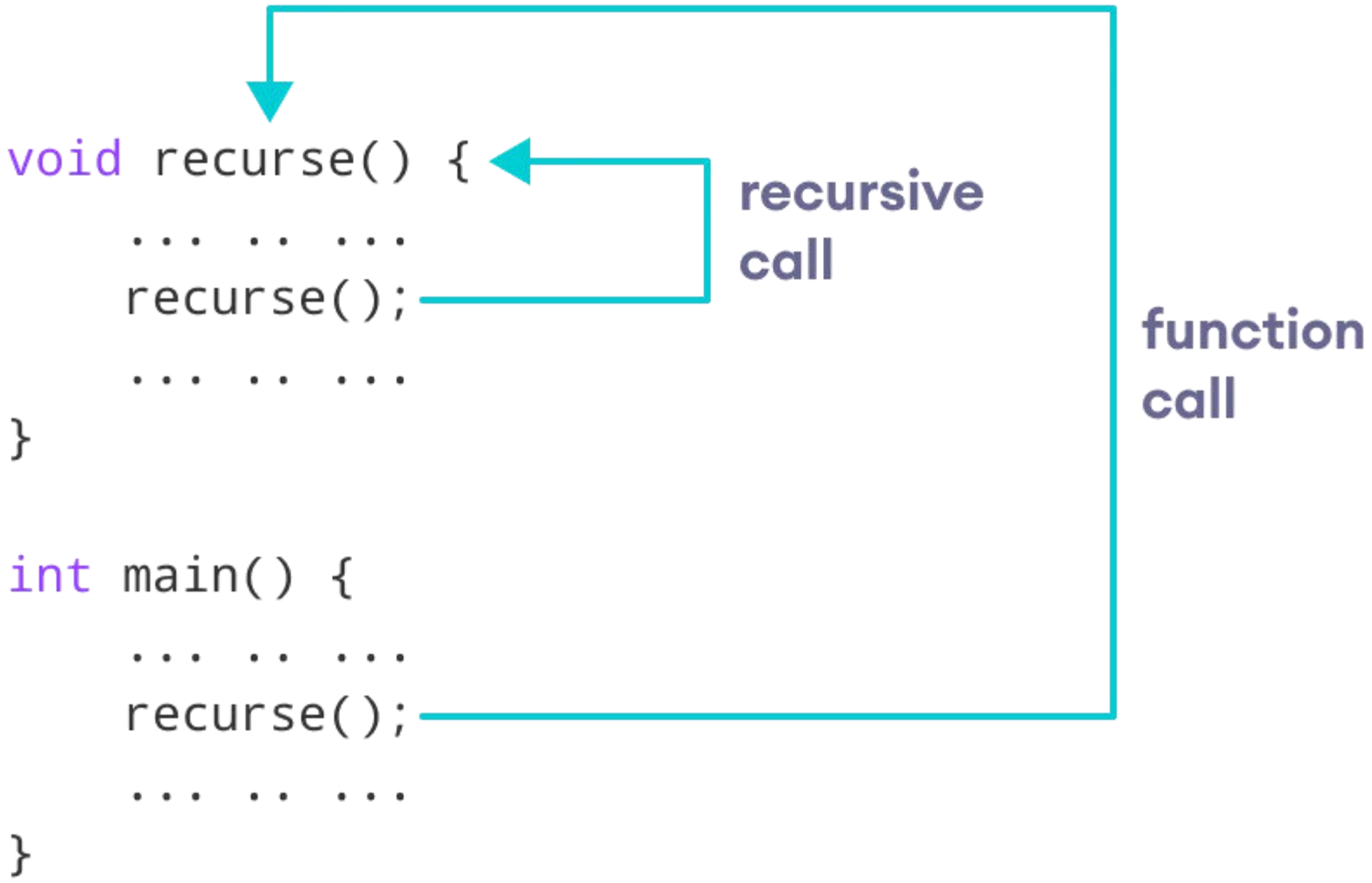
$$T(n) \in o(g(n)) \quad \text{is like} \quad b > a$$

$$T(n) \in \omega(g(n)) \quad \text{is like} \quad b < a$$

Recursion

- Recursion is a coding technique used in many algorithms.
- A recursive function is a function that calls itself.
- A problem can be solved with recursion if it can be broken down into successive smaller problems that are identical to the overall problem.

Recursion


void recurse() {
... .
recurse();
... .
}
int main() {
... .
recurse();
... .
}
recursive call
function call

Base Case and Recursive Case

- Because a recursive function calls itself, it's easy to write a function incorrectly that ends up in an infinite loop.
- When you write a recursive function, you have to tell it when to stop recursing.
- That's why every recursive function has two parts: the base case, and the recursive case.
- The recursive case is when the function calls itself.
- The base case is when the function doesn't call itself again, so it doesn't go into an infinite loop.

Sum of Integers

ALGORITHM *Sum*(n)

//Computes the sum of integers from 1 to n recursively

//Input: A nonnegative integer n

//Output: The sum of integers from 1 to n

if $n = 0$

    return 0

else

    return *Sum*($n - 1$) + n

Factorial

- In mathematics, the notation $n!$ represents the factorial of the nonnegative integer n.
- The factorial of n is the product of all the integers from 1 to n.

$$n! = n \times (n - 1) \times (n - 2) \times (n - 3) \times \ldots \times 2 \times 1$$

$$n! = n \times (n - 1)!$$

- For example,

$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$

$4! = 4 \times 3 \times 2 \times 1 = 24$

$0! = 1$

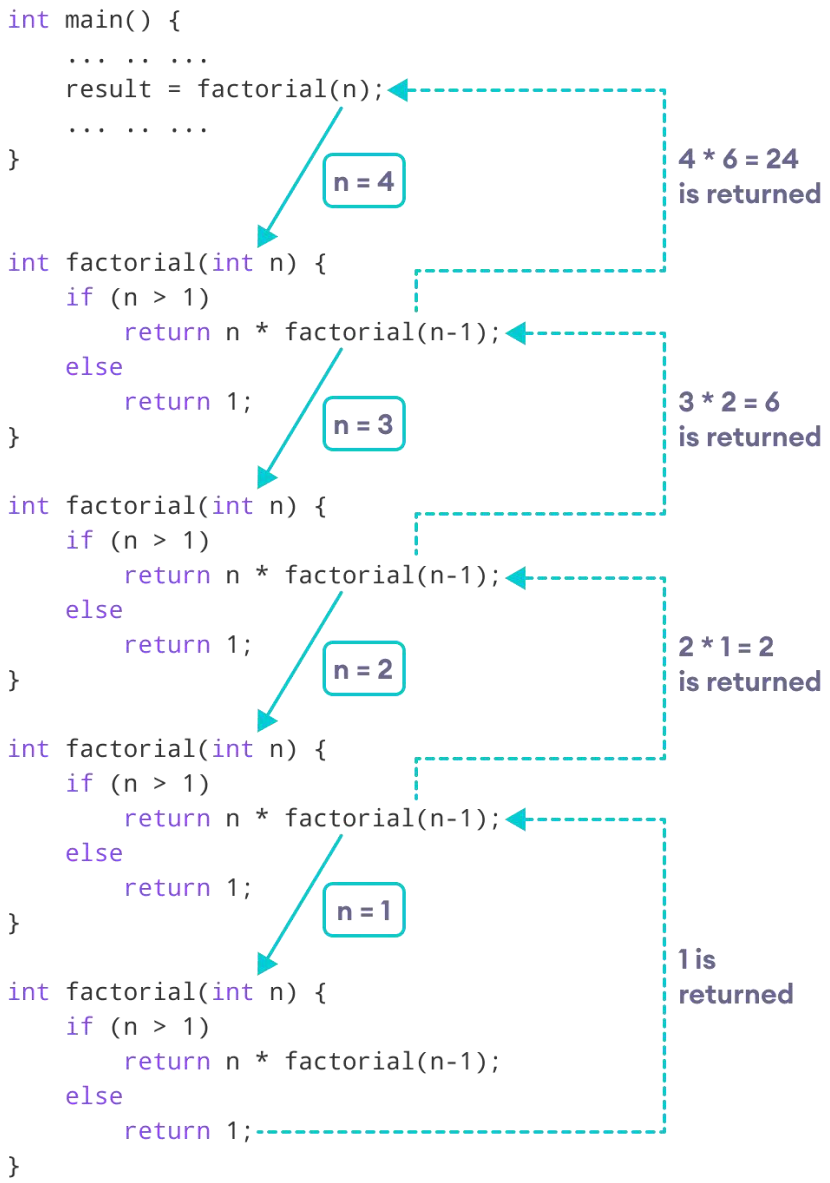
Factorial

By definition, we can compute $F(n) = F(n-1) \cdot n$ with the following recursive algorithm.

ALGORITHM F(n)

```
    //Computes n! recursively
    //Input: A nonnegative integer n
    //Output: The value of n!
    if n = 0
        return 1                          // 0! = 1
    else
        return F(n − 1) ∗ n               // n! = n × (n – 1)!
```

Factorial


int main() {
... .
result = factorial(n);
... .
}
n = 4
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
n = 3
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
n = 2
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
n = 1
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
4 * 6 = 24
is returned
3 * 2 = 6
is returned
2 * 1 = 2
is returned
1 is
returned

Factorial: Analysis

$$T(n) = T(n-1) + 1, \qquad T(0) = 0$$

$$T(n-1) = T(n-2) + 1$$
$$T(n) = \left[T(n-2) + 1\right] + 1$$
$$T(n) = T(n-2) + 2$$

$$T(n-2) = T(n-3) + 1$$
$$T(n) = \left[T(n-3) + 1\right] + 2$$
$$T(n) = T(n-3) + 3$$

$$T(n) = T(n-i) + i$$
$$T(n) = T(n-n) + n$$
$$T(n) = T(0) + n$$
$$T(n) = n \qquad \in \Theta(n)$$

Advantages of Recursion

- It makes our code shorter and cleaner.
- Recursion is required in problems concerning data structures and advanced algorithms, such as Graph and Tree Traversal.

Disadvantages of Recursion

- It takes a lot of stack space compared to an iterative program.
- It uses more processor time.
- It can be more difficult to debug compared to an equivalent iterative program.

Summations

$$\sum_{i=m}^{n} 1 = n - m + 1$$

$$\sum_{i=1}^{n} 1 = n$$

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

$$\sum_{i=0}^{n} i^2 = \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}$$

Algebraic Properties of Logarithms

Definition

$$2^y = x \Leftrightarrow y = \log_2(x)$$

Properties

$$\log_b(1) = 0$$

$$\log_b(b) = 1$$

$$\log_b(x \cdot y) = \log_b(x) + \log_b(y)$$

$$\log_b\left(x / y\right) = \log_b(x) - \log_b(y)$$

$$\log_b\left(1 / y\right) = -\log_b(y)$$

$$\log_b(x^a) = a \cdot \log_b(x)$$

$$\log_b(x) = \frac{\ln(x)}{\ln(b)} = \frac{\log_2(x)}{\log_2(b)} = \frac{\log_a(x)}{\log_a(b)}$$

Backward Substitution Method: Example 1

$$T(n) = T(n-1) + 5, \qquad T(1) = 0$$

$$T(n-1) = T(n-2) + 5$$

$$T(n) = \left[T(n-2) + 5\right] + 5$$

$$T(n) = T(n-2) + 2 \times 5$$

$$T(n-2) = T(n-3) + 5$$

$$T(n) = \left[T(n-3) + 5\right] + 2 \times 5$$

$$T(n) = T(n-3) + 3 \times 5$$

$$T(n) = T(n-i) + 5i$$

$$T(n) = T(n-n+1) + 5(n-1)$$

$$T(n) = T(1) + 5(n-1)$$

$$T(n) = 5(n-1) \qquad \in \Theta(n)$$

Backward Substitution Method: Example 2

$$T(n) = T(n-1) + n, \qquad T(0) = 0$$

$$T(n-1) = T(n-2) + (n-1)$$
$$T(n) = \left[T(n-2) + (n-1)\right] + n$$

$$T(n-2) = T(n-3) + (n-2)$$
$$T(n) = \left[T(n-3) + (n-2)\right] + (n-1) + n$$

$$T(n) = T(n-3) + (n-3+1) + (n-3+2) + n$$
$$T(n) = T(n-i) + (n-i+1) + (n-i+2) + n$$
$$T(n) = T(n-n) + (n-n+1) + (n-n+2) + n$$
$$T(n) = T(0) + 1 + 2 + \ldots + n$$
$$T(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad \in \Theta(n^2)$$

Backward Substitution Method: Example 3

$$T(n) = T(n-1) + 4n, \quad T(0) = 1$$

$$T(n-1) = T(n-2) + 4(n-1)$$

$$T(n) = \left[T(n-2) + 4(n-1)\right] + 4n$$

$$T(n-2) = T(n-3) + 4(n-2)$$

$$T(n) = \left[T(n-3) + 4(n-2)\right] + 4(n-1) + 4n$$

$$T(n) = T(n-3) + 4(n-3+1) + 4(n-3+2) + 4n$$

$$T(n) = T(n-i) + 4(n-i+1) + 4(n-i+2) + 4n$$

$$T(n) = T(n-n) + 4(n-n+1) + 4(n-n+2) + 4n$$

$$T(n) = T(0) + 4(1+2+\ldots+n) = 1 + 4\sum_{i=1}^{n} i = 1 + 4.\frac{n(n+1)}{2}$$

$$T(n) = 1 + 2n(n+1) = 2n^2 + 2n + 1 \qquad \in \Theta(n^2)$$

Backward Substitution Method: Example 4

$$T(n) = T(n-1) + n^2, \quad T(0) = 0$$

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

$$T(n) = \left[T(n-2) + (n-1)^2 \right] + n^2$$

$$T(n-2) = T(n-3) + (n-2)^2$$

$$T(n) = \left[T(n-3) + (n-2)^2 \right] + (n-1)^2 + n^2$$

$$T(n) = T(n-3) + (n-3+1)^2 + (n-3+2)^2 + n^2$$

$$T(n) = T(n-i) + (n-i+1)^2 + (n-i+2)^2 + n^2$$

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

$$T(n) = T(0) + 1^2 + 2^2 + \ldots + n^2$$

$$T(n) = \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \quad \in \Theta(n^3)$$

Backward Substitution Method: Example 5

$$T(n) = 3T(n-1), \qquad T(1) = 4$$

$$T(n-1) = 3T(n-2)$$

$$T(n) = 3\left[3T(n-2)\right]$$

$$T(n) = 3^2 T(n-2)$$

$$T(n-2) = 3T(n-2)$$

$$T(n) = 3^2\left[3T(n-3)\right]$$

$$T(n) = 3^3 T(n-3)$$

$$T(n) = 3^i T(n-i)$$

$$T(n) = 3^{n-1} T(n-n+1)$$

$$T(n) = 3^{n-1} T(1)$$

$$T(n) = 4.3^{n-1} \qquad \in \Theta(3^n)$$

Backward Substitution Method: Example 6

Algorithm Power(n)

//Computes 2^n recursively by the formula $2^n = 2^{n-1} + 2^{n-1}$

//Input: A nonnegative integer n

//Output: Returns 2^n

if $n = 0$

return 1

else

return Power($n - 1$) + Power($n - 1$)

Backward Substitution Method: Example 6

$$T(n) = 2T(n-1) + 1, \quad T(0) = 0$$

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

$$T(n) = 2\left[2T(n-1) + 1\right] + 1$$

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

$$T(n-2) = 2T(n-3) + 1$$

$$T(n) = 2^2\left[2T(n-3) + 1\right] + 2 + 1$$

$$T(n) = 2^3 T(n-3) + 2^2 + 2 + 1 = 2^3 T(n-3) + 7$$

$$T(n) = 2^3 T(n-3) + 2^3 - 1$$

$$T(n) = 2^i T(n-i) + 2^i - 1$$

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

$$T(n) = 2^n T(0) + 2^n - 1$$

$$T(n) = 2^n - 1 \quad \in \Theta(2^n)$$

Backward Substitution Method: Example 7

$$T(n) = T(n/3) + 1, \qquad T(1) = 1$$

Let $n = 3^k \Leftrightarrow k = \log_3(n)$

$$T(3^k) = T(3^{k-1}) + 1$$

$$T(3^{k-1}) = T(3^{k-2}) + 1$$

$$T(3^k) = \left[T(3^{k-2}) + 1\right] + 1 = T(3^{k-2}) + 2$$

$$T(3^{k-2}) = T(3^{k-3}) + 1$$

$$T(3^k) = \left[T(3^{k-3}) + 1\right] + 2 = T(3^{k-3}) + 3$$

$$T(3^k) = T(3^{k-i}) + i$$

$$T(3^k) = T(3^{k-k}) + k$$

$$T(3^k) = T(1) + k = 1 + k$$

$$T(n) = 1 + \log_3(n) \qquad \in \Theta(\log_3(n))$$

Backward Substitution Method: Example 8

$$T(n) = T(n/2) + n, \qquad T(1) = 1$$

Let $n = 2^k \iff k = \log_2(n)$

$$T(2^k) = T(2^{k-1}) + 2^k$$

$$T(2^{k-1}) = T(2^{k-2}) + 2^{k-1}$$

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

$$T(2^{k-2}) = T(2^{k-3}) + 2^{k-2}$$

$$T(2^k) = \left[T(2^{k-3}) + 2^{k-2}\right] + 2^{k-1} + 2^k$$

$$T(2^k) = T(2^{k-3}) + 2^{k-3+1} + 2^{k-3+2} + 2^k$$

$$T(2^k) = T(2^{k-i}) + 2^{k-i+1} + 2^{k-i+2} + 2^k$$

$$T(2^k) = T(2^{k-k}) + 2^{k-k+1} + 2^{k-k+2} + 2^k = T(1) + 2^1 + 2^2 + \ldots + 2^k$$

$$T(2^k) = 2^0 + 2^1 + 2^2 + \ldots + 2^k = 2^{k+1} - 1 = 2.2^k - 1 = 2n - 1 \qquad \in \Theta(n)$$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}$$

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$$

Backward Substitution Method: Example 9

$$T(n) = 7T(n/2), \qquad T(1) = 1$$

Let $n = 2^k \Leftrightarrow k = \log_2(n)$

$$T(2^k) = 7T(2^{k-1})$$

$$T(2^{k-1}) = 7T(2^{k-2})$$

$$T(2^k) = 7^2 T(2^{k-2})$$

$$T(2^{k-2}) = 7T(2^{k-3})$$

$$T(2^k) = 7^3 T(2^{k-3})$$

$$T(2^k) = 7^i T(2^{k-i})$$

$$T(2^k) = 7^k T(2^{k-k})$$

$$T(2^k) = 7^k T(0) = 7^k$$

$$T(n) = 7^{\log_2(n)} = 7^{\frac{\log_7(n)}{\log_7(2)}} = 7^{2.807\log_7(n)} = 7^{\log_7(n^{2.807})} = n^{2.807} \qquad \in \Theta(n^{2.807})$$

Backward Substitution Method: Example 10

$$T(n) = 2T(n/2) + n, \quad T(1) = 0$$

Let $n = 2^k \Leftrightarrow k = \log_2(n)$

$$T(2^k) = 2T(2^{k-1}) + 2^k$$

$$T(2^{k-1}) = 2T(2^{k-2}) + 2^k$$

$$T(2^k) = 2\left[2T(2^{k-2}) + 2^k\right] + 2^k = 2^2 T(2^{k-2}) + 2.2^k$$

$$T(2^{k-2}) = 2T(2^{k-3}) + 2^{k-2}$$

$$T(2^k) = 2^2\left[2T(2^{k-3}) + 2^{k-2}\right] + 2.2^k = 2^3 T(2^{k-3}) + 3.2^k$$

$$T(2^k) = 2^i T(2^{k-i}) + i.2^k$$

$$T(2^k) = 2^k T(2^{k-k}) + k.2^k$$

$$T(2^k) = 2^k T(1) + k.2^k = nT(1) + n\log_2(n)$$

$$T(n) = n\log_2(n) \qquad \in \Theta(n\log_2(n))$$