Master Theorem

- If $af(n/b) = \kappa f(n)$ for some constant $\kappa < 1$, then $T(n) = \Theta(f(n))$.
- If $af(n/b) = Kf(n)$ for some constant $K > 1$, then $T(n) = \Theta(n^{\log_b a})$.
- If $af(n/b) = f(n)$, then $T(n) = \Theta(f(n)\log_b n)$.

Annihilators

Operator	Functions annihilated
$\mathbf{E} - 1$	α
$\mathbf{E} - a$	αa^n
$(\mathbf{E} - a)(\mathbf{E} - b)$	$\alpha a^n + \beta b^n$
$(\mathbf{E} - a_0)(\mathbf{E} - a_1)\cdots(\mathbf{E} - a_k)$	$\sum_{i=0}^{k} \alpha_i a_i^n$
$(\mathbf{E} - 1)^2$	$\alpha n + \beta$
$(\mathbf{E} - a)^2$	$(\alpha n + \beta)a^n$
$(\mathbf{E} - a)^2(\mathbf{E} - b)$	$(\alpha n + \beta)a^b + \gamma b^n$
$(\mathbf{E} - a)^d$	$(\sum_{i=0}^{d-1} \alpha_i n^i)a^n$

If X annihilates f, then X also annihilates $\mathbf{E}f$.
If X annihilates both f and g, then X also annihilates $f \pm g$.
If X annihilates f, then X also annihilates αf, for any constant α.

If X annihilates f and Y annihilates g, then XY annihilates $f \pm g$.

Tower of Hanoi - Vanilla, Variant 0

```
function Hanoi($n, src, dst, tmp$)
    if $n > 0$ then
        Hanoi($n - 1, src, tmp, dst$)
        move disk $n$ from $src$ to $dst$
        Hanoi($n - 1, tmp, dst, src$)
    end if
end function
```

Variant 1 - Moves Must Involve Temp/0

```
function HanoiVariant1($n, src, dst, tmp$)
    if $n > 0$ then
        HanoiVariant1($n - 1, src, dst, tmp$)
        move disk $n$ from $src$ to $tmp$
        HanoiVariant1($n - 1, dst, src, tmp$)
        move disk $n$ from $tmp$ to $dst$
        HanoiVariant1($n - 1, src, dst, tmp$)
    end if
end function
```

Variant 2 - Counterclock: only Src->Tmp->Dest->Src

```
function HanoiVariant2($n, src, dst, tmp$)
    if $n = 1$ then
        move disk $n$ from $src$ to $tmp$
        move disk $n$ from $tmp$ to $dst$
    else if $n > 0$ then
        HanoiVariant2($n - 1, src, dst, tmp$)
        move disk $n$ from $src$ to $tmp$
        HanoiVariant2($n - 2, dst, tmp, src$)
        move disk $n - 1$ from $dst$ to $src$
        HanoiVariant2($n - 2, tmp, src, dst$)
        move disk $n$ from $tmp$ to $dst$
        HanoiVariant2($n - 1, src, dst, tmp$)
    end if
end function
```

Merge Sort

```
function MergeSort($A[1..n]$)
    if $n > 1$ then
        $m \leftarrow \lfloor n/2 \rfloor$
        MergeSort($A[1..m]$)
        MergeSort($A[m + 1..n]$)
        Merge($A[1..n], m$)
    end if
end function
function Merge($A[1..n], m$)
    $i \leftarrow 1; j \leftarrow m + 1$
    for $k \leftarrow 1, n$ do
        if $j > n$ then
            $B[k] \leftarrow A[i]; i \leftarrow i + 1$
        else if $i > m$ then
            $B[k] \leftarrow A[j]; j \leftarrow j + 1$
        else if $A[i] < A[j]$ then
            $B[k] \leftarrow A[i]; i \leftarrow i + 1$
        else
            $B[k] \leftarrow A[j]; j \leftarrow j + 1$
        end if
    end for
    for $k \leftarrow 1, n$ do
        $A[k] \leftarrow B[k]$
    end for
end function
```

Quicksort

```
function QuickSort($A[1..n]$)
    if $n > 1$ then
        Choose a pivot element $A[p]$
        $r \leftarrow$ Partition($A, p$)
        QuickSort($A[1..r - 1]$)
        QuickSort($A[r + 1..n]$)
    end if
end function
function Partition($A[1..n], p$)
    swap $A[p] \leftrightarrow A[n]$
    $i \leftarrow 0; j \leftarrow n$
    while $i < j$ do
        repeat $i \leftarrow i + 1$ until $i \geq j$ or $A[i] \geq A[n]$
        repeat $j \leftarrow j - 1$ until $i \geq j$ or $A[j] \leq A[n]$
        if $i < j$ then
            swap $A[i] \leftrightarrow A[j]$
        end if
    end while
    swap $A[i] \leftrightarrow A[n]$
    return $i$
end function
```

Longest Common Subsequence

```
function LCS($A[1..m], B[1..n]$)
    if $m = 0$ or $n = 0$ then
        return 0
    else
        $a \leftarrow$ LCS($A[2..m], B[1..n]$)
        $b \leftarrow$ LCS($A[1..m], B[2..n]$)
        $c \leftarrow 0$
        if $A[1] = B[1]$ then
            $c \leftarrow 1+$ LCS($A[2..m], B[2..n]$)
        end if
        $r \leftarrow$ the maximum of $a$, $b$, and $c$
    end if
end function
```

Longest Oscillating Subsequence

```
function LOS($X[1..n]$)
    return LOS2($X[1..n], null, false$)
end function
function LOS2($X[1..n], prevValue, isEven$)
    if $n = 0$ then
        return 0
    else
        if $prevValue = null$ then
            $result \leftarrow true$
        else if $isEven$ then
```

```
            $result \leftarrow prevValue > X[1]$
        else
            $result \leftarrow prevValue < X[1]$
        end if
        $a \leftarrow$ LOS2($X[2..n], prevValue, isEven$)
        if $result$ then
            $b \leftarrow 1+$ LOS2($X[2..n], X[1]$, not $isEven$)
        else
            $b \leftarrow 0$
        end if
        return the maximum of $a$ and $b$
    end if
end function
```

Longest Accelerating Subsequence

```
function LXS($X[1..n]$)
    return LXS2($X[1..n], null, null$)
end function
function LXS2($X[1..n], prevValue, prevDiff$)
    if $n = 0$ then
        return 0
    else
        if $prevValue = null$ or $prevDiff = null$ then
            $result \leftarrow true$
        else
            $result \leftarrow X[1] - prevValue > prevDiff$
        end if
        $a \leftarrow$ LXS2($X[2..n], prevValue, prevDiff$)
        if $result$ then
            if $prevValue = null$ then
                $b \leftarrow 1+$ LXS2($X[2..n], X[1], null$)
            else
                $b \leftarrow 1+$ LXS2($X[2..n], X[1], X[1] - prevValue$)
            end if
        else
            $b \leftarrow 0$
        end if
        return the maximum of $a$ and $b$
    end if
end function
```

Subset Sum - Basic

```
function SubsetSum($X[1..n], T$)
    if $T = 0$ then
        return True
    else if $T < 0$ or $n = 0$ then
        return False
    else
        return SubsetSum($X[1..n-1], T$) $\vee$ SubsetSum($X[1..n-1], T - X[n]$)
    end if
end function
```

Subset Sum - Memoized

```
function SubsetSum($X[1..n], T$)
    $S[n+1, 0] \leftarrow$ True
    for $t \leftarrow 1, T$ do
        $S[n+1, t] \leftarrow$ False
    end for
    for $i \leftarrow n, 1$ do
        $S[i, 0] \leftarrow$ True
        for $t \leftarrow 1, X[i] - 1$ do
            $S[i, t] \leftarrow S[i+1, t]$ ▷ Avoid the case $t < 0$
        end for
        for $t \leftarrow X[i], T$ do
            $S[i, t] \leftarrow S[i+1, t] \vee S[i+1, t - X[i]]$
        end for
    end for
    return $S[1, T]$
end function
```

Dynamic Programming

1. Formulate the problem recursively
2. Build solutions to recurrence from the bottom up
   (a) Identify the subproblems
   (b) Analyze space and running time
   (c) Choose a data structure to memoize intermediate results
   (d) Identify dependencies between subproblems
   (e) Find a good evaluation order
   (f) Write down the algorithm

Longest Palindrome Subsequence

```
function LPS($text[1..n]$)
    $memoized \leftarrow$ empty hash map
    $memoized$[the empty list] $\leftarrow 0$
    for $i \leftarrow 1, n$ do
        for $j \leftarrow 1, n - i$ do
            $subproblem \leftarrow text[j..j+i]$
            if $i = 1$ then
                $memoized[subproblem] \leftarrow 1$ ▷ A single letter is always a palindrome of length one
            else
                $r \leftarrow$ the maximum of $memoized[subproblem[2..i]]$ and $memoized[subproblem[1..i-1]]$
                if $subproblem[1] = subproblem[i]$ then
                    $r \leftarrow$ the maximum of $r$ and $2 + memoized[subproblem[2..i-1]]$
                end if
                $memoized[subproblem] \leftarrow r$
            end if
        end for
    end for
    return $memoized[text]$
end function
```

Median Selection / kth largest

```
function QuickSelect($A[1..n], k$)
    if $n = 1$ then
        return $A[1]$
    else
        Choose pivot element $A[p]$ or MedOfFive
        $r \leftarrow PARTITION(A[1...n], p)$
        if $k < r$ then
            return QUICKSELECT(A[1...r-1], k)
        else if k > r then
            return QUICKSELECT(A[r+1...n], k-r)
        else
            return $A[r]$
        end if
    end if
end function
```

EditDistances(A[1..m], B[1..n])

```
for j <- 1 to n { Edit[0,j] <- j }
for i <- 1 to m
....Edit[i,0] <- i
....for j <- 1 to n
........if A[i] = B[j]
............Edit[i,j] <- min(edit[i-1, j] + 1, edit[i, j-1] + 1, Edit[i-1, j-1])
........else
...........Edit[i,j]<-min(edit[i-1, j]+1, edit[i, j-1]+1, Edit[i-1, j-1]+1)
....return Edit[m,n]
```