1.1. Recursive Dynamic Programming

```
function Fib(n)
    if n<2 then
        return 1
    return Fib(n-1) + Fib(n-2)
```

$T(n) = T(n-1) + T(n-2) + c \qquad O(1.618^n)$

```
function Fib(n)
    if mem[n] is not empty then
        return mem[n]
    if n<2 then
        mem[n] <- 1
    else
        mem[n] <- Fib(n-1) + Fib(n-2)
    return mem[n]
```

(memoisation)

compute mem[1-n] once $\to O(n)$

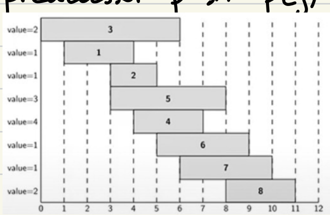
1.2. Weighted interval scheduling

Problem:

Job j, starts at s_j, finishes at f_j, has value v_j.
Two jobs compatible if no overlap.
Goal: Find max value subset of compatible jobs.

Solution:

label jobs by f_i s.t. $f_1 \le f_2 \le \ldots \le f_n$
predecessor p s.t. $p(j)$ = largest i, $i<j$, jobs i,j are compatible



$p(8) = 5, \; p(7) = 3, \; p(2) = 0$
$Opt(j)$ = value of optimal solution of jobs 1-j
$Opt(5), Opt(7)$, use v_8 if $Opt(5) + v_8 \ge Opt(7)$
$Opt(8) = \max(v_8 + Opt(5), \; Opt(7))$

$$Opt(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max(v_j + Opt(p(j)), \; Opt(j-1)) & \text{else} \end{cases}$$

```
function Opt(j)
    if j=0 then return 0
    else return max(v(j)+Opt(p(j)), Opt(j-1))
```

p(j) could be different

$T(0) = 0, \; T(n) = T(n-1) + T(n-2) + c$ so exponential

```
Label jobs by ascending finish times, so f1 <= f2 <= … <= fn        O(n log n)
Compute all predecessors p(1), p(2), …, p(n):        O(n log n)
    Store job indexes by ascending start times s(j) in array t        O(n log n)
    k <- n; t[0] <- 0        O(1)
    for l <- n…1 do        O(n)
        while f(k) > s(t[l]) do        O(log n)        } O(n log n)
            k <- k-1        O(1)
        p[l]<- t[k]        O(1)
for j <- 1…n do        O(n)
    M[j] <-empty        O(1)        } O(n)
M[0] <- 0        O(1)
function Opt(j)
    if M[j] = empty then
        M[j] <- max(v(j)+Opt(p(j)), Opt(j-1))        O(1)
    return M[j]        -> total: O(n log n)
```

Claim: $\forall i \in [0,n]$: $M[i]$ contains max weight of subset of jobs $0-i$

Proof by induction:

Base: $M[0]=0$, no job, no value

IH: $\forall i \in [0,j]$, claim holds

Inductive step: consider job j

1) include j: besides j, $M[p(j)]$ - max value by IH
2) exclude j: $M[j-1]$ - max value by IH

Algorithm: $M[j] = \max(v_j + M[p(j)], M[j-1]) \Rightarrow$ optimal

```
function Find-Solution(j)
    if j=0 then
        return 0
    else if v(j)+M[p(j)] > M[j-1] then
        Find-Solution(p(j))
        print j
    else
        Find-Solution(j-1)
```

2.1. Word Segmentation

Problem:

string x of letters $x_1 x_2 \dots x_n$

quality function $q(i,j)$ gives value of $x_i x_{i+1} \dots x_j$ in $O(1)$

Goal: split x into words with max value

word	quality
lawy	1
ersar	1
eawe	1
so	2
me	3
lawyers	5
are	4
awesome	10

Solution:

ex. "lawyersareawesome"

$q(\text{lawy}) + q(\text{ersar}) + q(\text{eawe}) + q(\text{so}) + q(\text{me}) = 9$

$q(\text{lawyers}) + q(\text{are}) + q(\text{awesome}) = \underline{19}$

$$OPT(j) = \begin{cases} 0 & \text{if } j=0 \\ \max\limits_{1 \le i \le j} \left(q(i,j) + Opt(i-1) \right) & \text{else} \end{cases}$$

```
function Word-Segmentation
    M[0] <- 0
    for j <- 1 to n do    O(n)
        M[j] <- max{1 <= i <= j}(q(i,j) + M[i-1])  O(n)
    return M[n]  -> total: O(n^2)
function Find-Solution(j)
    if j=0 then
        return 0
    else
        i <- j
        while i>=1 and q(i,j) + M[i-1] != M[j] do
            i <- i-1
        Find-Solution(i-1)
        print i
```

2.2. Segmented Least Squares

Problem:

set of 2D points, point i with (x_i, y_i)

Goal: sequence of line segments L with min size and min sum of squared errors e, balance: $e + c|L|$, $c > 0$

Solution:

$Opt(j)$ – min cost for points $p_1, p_2, \dots, p_j$ (sorted on x). $Opt(0)=0$

$e(i,j)$ – min sum of squared errors for $p_i, p_{i+1}, \dots, p_j$

c – cost of extra line

$$Opt(j) = \begin{cases} 0 & \text{if } j=0 \\ \min\limits_{1 \le i \le j} \left(e(i,j) + c + OPT(i-1) \right) & \text{else} \end{cases}$$

```
M[0] <- 0
for j <- 1 to n do
    for i <- 1 to j do
        compute e(i,j)
for j <- 1 to n do    O(n)
    M[j] = e(1,j) + c
    for i <- 2 to j do    O(n)
        if M[j] > e(i,j) + c + M[i-1] then
            M[j] <- e(i,j) + c + M[i-1]
return M[n]   -> total: O(n^2)
```

3.1. Knapsack

Problem:

n items, item i with weight $w_i > 0$ kg, value $v_i > 0$

knapsack with capacity W kg

Goal: Find max value to carry

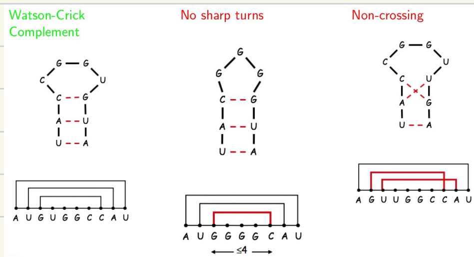
Solution:

$$Opt(i,w) = \begin{cases} 0 & \text{if } i=0 \\ Opt(i-1,w) & \text{if } w_i > w \\ \max\left(Opt(i-1,w),\ v_i + Opt(i-1, w-w_i)\right) & \text{else} \end{cases}$$

```
for w <- 0 to W do M[0,w] <- 0
for i <- 1 to n do
    for w <- 0 to W do
        if w(i) > w then M[i,w] <- M[i-1,w]
        else M[i,w] <- max(M[i-1,w], v(i)+M[i-1,w-w(i)])
return M[n,W]   -> total O(nW)   pseudo-polynomial
```

```
function Find-Solution(i,w)
    if i=0 or w=0 then
        return 0
    else if M[i,w] = M[i-1,w] then
        Find-Solution(i-1, w)
    else
        Find-Solution(i-1, w-w(i))
        print i
```

3.2. RNA



Problem:

string $B = b_1 b_2 .. b_n$ over $\{A, C, G, U\}$

set od pairs $S: \{(b_i, b_j) \mid b_i, b_j \in B\}$ s.t.

$A+U$ or $C+G$, $i<j-4$,

if $(b_i, b_j), (b_k, b_l) \in S$ then $i<k<l<j$

Solution:

$$Opt(i,j) = \begin{cases} 0 & \text{if } i \geq j-4 \\ \max\left(1 + \max\limits_{i \leq t \leq j-4} \left(Opt(i,t-1) + Opt(t+1,j-1)\right), Opt(i,j-1)\right) \end{cases}$$

```
function RNA(b1,…,by)
    for k <- 1 to n-1 do
        for i <- 1 to n-k do
            j <- i + k
            Compute M[i,j]
    return M[1,n]
```

4.1. Sequence Alignment

Problem:

two strings $X = x_1 x_2 .. x_m$ and $Y = y_1 y_2 ... y_n$, find min cost alignment M

alignment M - set of ordered pairs $x_i - y_j$ s.t. ∀item occurs in most 1
two crossings

$x_i - y_j$, $x_{i'} - y_{j'}$ cross iff $i<i'$, but $j>j'$

$$cost(M) = \underbrace{\sum_{(x_i,y_j)\in M} \alpha_{x_i,y_j}}_{\text{missmatches}} + \underbrace{\sum_{i: x_i \text{ unmatched}} \delta + \sum_{j: y_j \text{ unmatched}} \delta}_{\text{gaps}}$$

Solution:

$Opt(i,j)$ – min cost of aligning $x_1 x_2 \dots x_i$ and $y_1 y_2 \dots y_j$

1) Opt match x_i, y_j: pay for mismatch + $Opt(i-1, j-1)$

2) Opt unmatch x_i: pay for gap + $Opt(i-1, j)$

3) Opt unmatch y_j: pay for gap + $Opt(i, j-1)$

$$Opt(i,j) = \begin{cases} j\delta & \text{if } i=0 \\ \min\begin{pmatrix} \alpha_{x_i y_j} + Opt(i-1, j-1) \\ \delta + Opt(i-1, j) \\ \delta + Opt(i, j-1) \end{pmatrix} & \text{if } i>0 \text{ and } j>0 \\ i\delta & \text{if } j=0 \end{cases}$$

```
function Sequence-Alignment(m,n,x1x2…xm,y1y2…yn,δ,α)
    for i <- 0 to m do
        M[i,0] <- iδ
    for j <- 0 to n do
        M[0,j] <- jδ
    for j <- 1 to n do
        for i <- 1 to m do
            M[i,j] <- min{α[xi,yj] + M[i-1,j-1], δ + M[i-1,j], δ + M[i,j-1]}
    return M[m,n]
```

→ total (time and space): $\Theta(nm)$

Proof by induction:

Base: if $m=0$, n gap penalties; if $n=0$, m gap penalties

IH: Algorithm provides optimal solution up to $i, j-1$ and $i-1, j$ $(k = i+j-1)$

Inductive step: x, y of lengths i, j $(k+1 = i+j)$

3 options for aligning last characters:

1) match x_i, y_j → pay for mismatch + aligning $i-1, j-1$

2) x_i unmatched → pay for gap + aligning $i-1, j$

3) y_j unmatched → pay for gap + aligning $i, j-1$

No other option, otherwise crossing.

Algorithm: take min of 3 options ⇒ optimal.

```
function Sequence-Alignment-LS(m,n,x1x2…xm,y1y2…yn,δ,α)
    for i <- 0 to m do
        M[i] <- iδ
    for j <- 1 to n do
        left <- (j-1)δ
        for i <- 1 to m do
            lb <- left
            left <- M[i]
            M[i] <- min{α[xi,yj] + lb, δ + M[i-1], δ + left}
    return M[m]
```
→ total: space $O(m)$

4.2. Bellman-Ford

Problem: Shortest path

directed graph $G=(V,E)$, edge weights $c_{v,w}$ (can be <0)

Goal: find shortest path from s to t

Solution:

$$Opt(v) = \begin{cases} 0 & \text{if } v=s \\ \min_{(v,w)\in E} \left(Opt(w) + c_{w,v}\right) & \text{otherwise} \end{cases}$$

most i edges

$$Opt(i,v) = \begin{cases} 0 & \text{if } v=s \\ \infty & \text{if } i=0 \\ \min_{(v,w)\in E} \left(Opt(i-1,w) + c_{w,v}\right) & \text{otherwise} \end{cases}$$

```
function Shortest-Path(G,s,t)
    for v in V do
        M[0,v] <- inf
    for i <- 0 to n-1 do
        M[i,s] <- 0
    for i <- 1 to n-1 do
        for v in V do
            M[i,v] <- min{(w,v) in E}(M[i-1,w] + c(w,v))
    return min(M[_,t])
```
→ total: $\Theta(nm)$ time, $\Theta(n^2)$ space

Reduce space:

$\left.\begin{array}{l} M[v] \text{ - shortest path } s\text{-}v \\ predecessor[v] \text{ - best step found} \end{array}\right\}$ $O(n+m)$ space

```
function Push-Based-Shortest-Path(G,s,t) (aka Bellman-Ford Algorithm)
    for v in V do
        M[v] <- inf
        predecessor[v] <- emp
    M[s] <- 0
    for i <- 1 to n-1 do
        for w in V do
            if M[w] updated prev iteration then
                for v s.t. (w,v) in E do
                    if M[v] > M[w] + c(w,v) then
                        M[v] <- M[w] + c(w,v)
                        predecessor[v] <- w
        if no M[w] value changed in iteration i then
            return M[t]
```