- Graphs
    - Breadth-First Search(BFS)
        - Bipartite Graph
    - Depth-First Search (DFS)
        - Articulation Points
    - Node
- Greedy Algorithm
    - Huffman Algorithm
- Divide and conquer
    - Merge sort
    - Closest pair algorithm
    - Recurrence
        - Karatsuba multiplication
        - Master Recurrence
- Dynamic Programmin
    - Intro
    - Schedule
        - Weighted Interval Scheduling
        - Knapsack Algorithm
    - RNA

> Week 1 Class 2

# Runtime

**Upper Bounds**:
$f(n)$ is $O(g(n))$ if there is a constant $c > 0$ so that $f(n)$ is eventually always $\leq cg(n)$

**Lower Bounds**:
$f(n)$ is $\Omega(g(n))$ if there is a constant $c > 0$ so that $f(n)$ is eventually always $\geq cg(n)$

**Both:**
$f(n)$ is $\Theta(g(n))$ if there is are constants $c1, c2 > 0$ so that eventually always $c_1 g(n) \leq f(n) \leq c_2 g(n)$

> Week 1 Class 3

**Logarithms**

Change-of=base formula:

$$log_a(x) = \frac{log_b(x)}{log_b(a)}$$

$$log_b(a) * log_a(x) = log_b(x)$$

For all $x > 0$, *(no mater how small)* $log\ n = O(n^x)$

```
+ Week 2 Class 1
```

# Graphs

Undirected Graph:

Directed Graph:

Sparse graphs: $m \ll n^2$

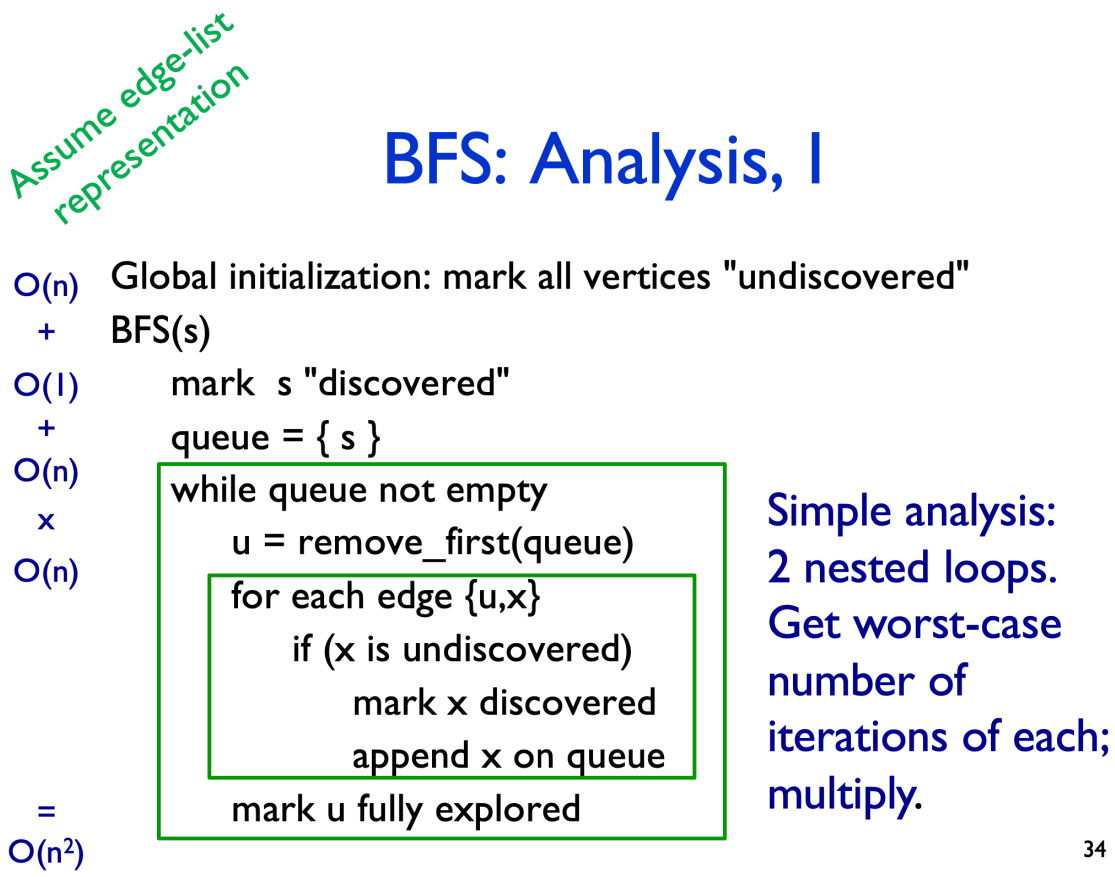
```
+ Week 2 Class 2
```

## Breadth-First Search(BFS)

level by level search

runtime:

- $O(n^2)$ in genal.




- $(n+m) if sparse graphs

  Application: Shortest Paths

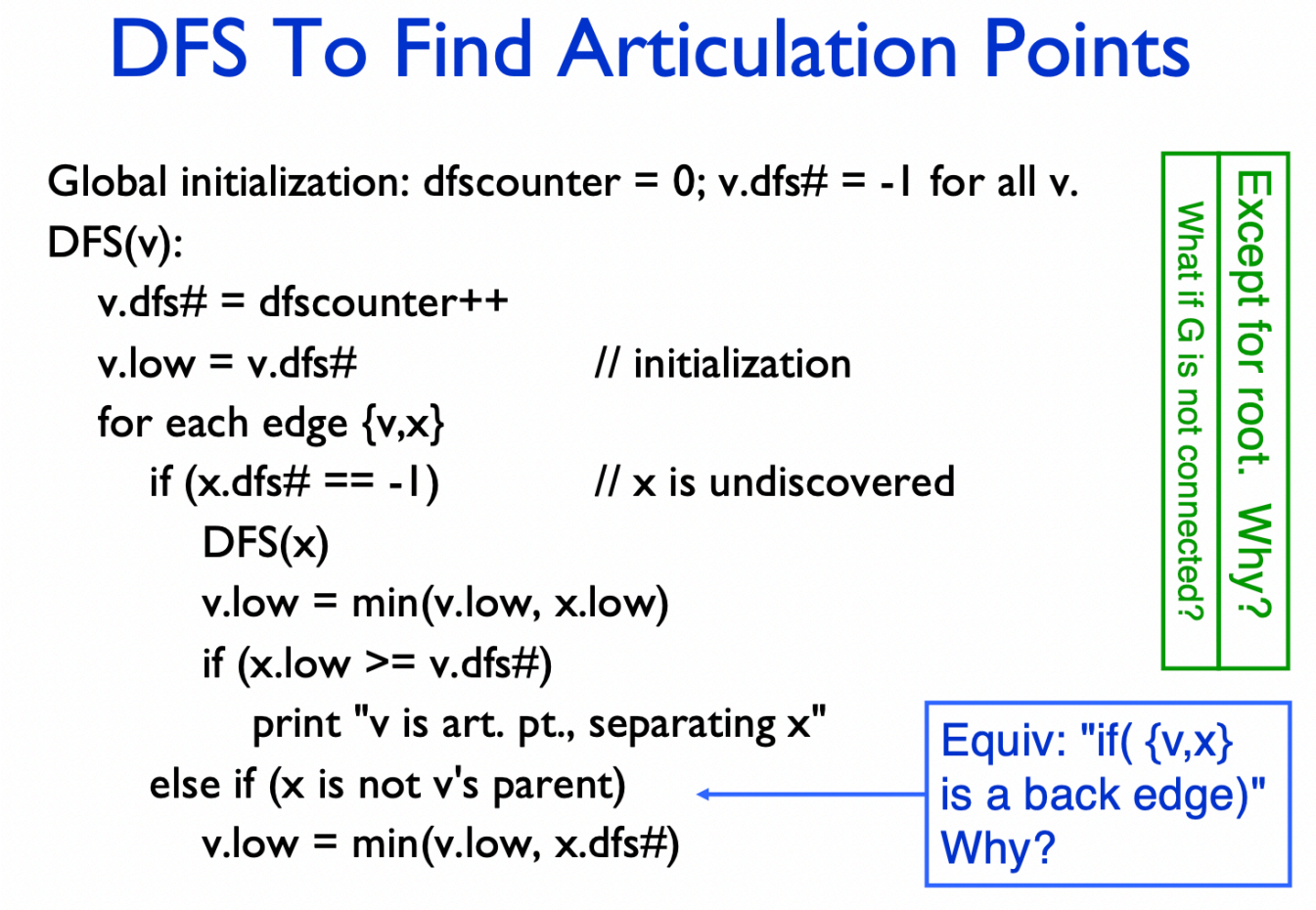
### Bipartite Graph

> odd cycle is,

## Depth-First Search (DFS)

go one deep frist and go back

**Articulation Points**




## Node

- Leaf nodes(Leaves): The botten nodes in every way of the tree. Node without children
- Internal nodes: Node have at least one child
- Root: Node without parent

```
+ Class 4.1
```

# Greedy Algorithm

```
+ Class 4.3
```

## Huffman Algorithm

use the less bits to represent the every elements (the data has probobility)

```
+ Class 5.2
```

# Divide and conquer

The basic idea is divide one full size probelem to two half

## Merge sort

sort the array

# Closest pair algorithm

```
Closest-Pair(p1, …, pn) {
   if(n <= ??) return ??

   Compute separation line L such that half the points
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Delete all points further than δ from separation line L

   Sort remaining points p[1]…p[m] by y-coordinate.

   for i = 1..m
      k = 1
      while i+k <= m && p[i+k].y < p[i].y + δ
        δ = min(δ, distance between p[i] and p[i+k]);
        k++;

   return δ.
}
```

+ Class 5.3

$$C(n) \le \left\{ \begin{array}{ll} 0 & n = 1 \\ 2C(n/2) + kn \log n & n > 1 \end{array} \right\} \Rightarrow C(n) = O(n \log^2 n)$$

for some constant $k$

Sort center strip

$C(n)$ is number of comparisons

# Recurrence

Merge sort

```
+ Class 6.1 2/7
```

## Karatsuba multiplication

To multiply two *n*-bit integers:

Add two pairs of ½*n* bit integers.

Multiply *three* pairs of ½*n*-bit integers.

Add, subtract, and shift *n*-bit integers to obtain result.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot \big( (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0 \big) + x_0 y_0 \\
&\qquad\quad A \qquad\qquad\quad B \qquad\qquad A \quad C \qquad C
\end{aligned}
$$

Theorem. [Karatsuba-Ofman, 1962] Can multiply two *n*-digit integers in $O(n^{1.585})$ bit operations.

$$
T(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}}
$$

*Sloppy version :* $T(n) \le 3T(n/2) + O(n)$

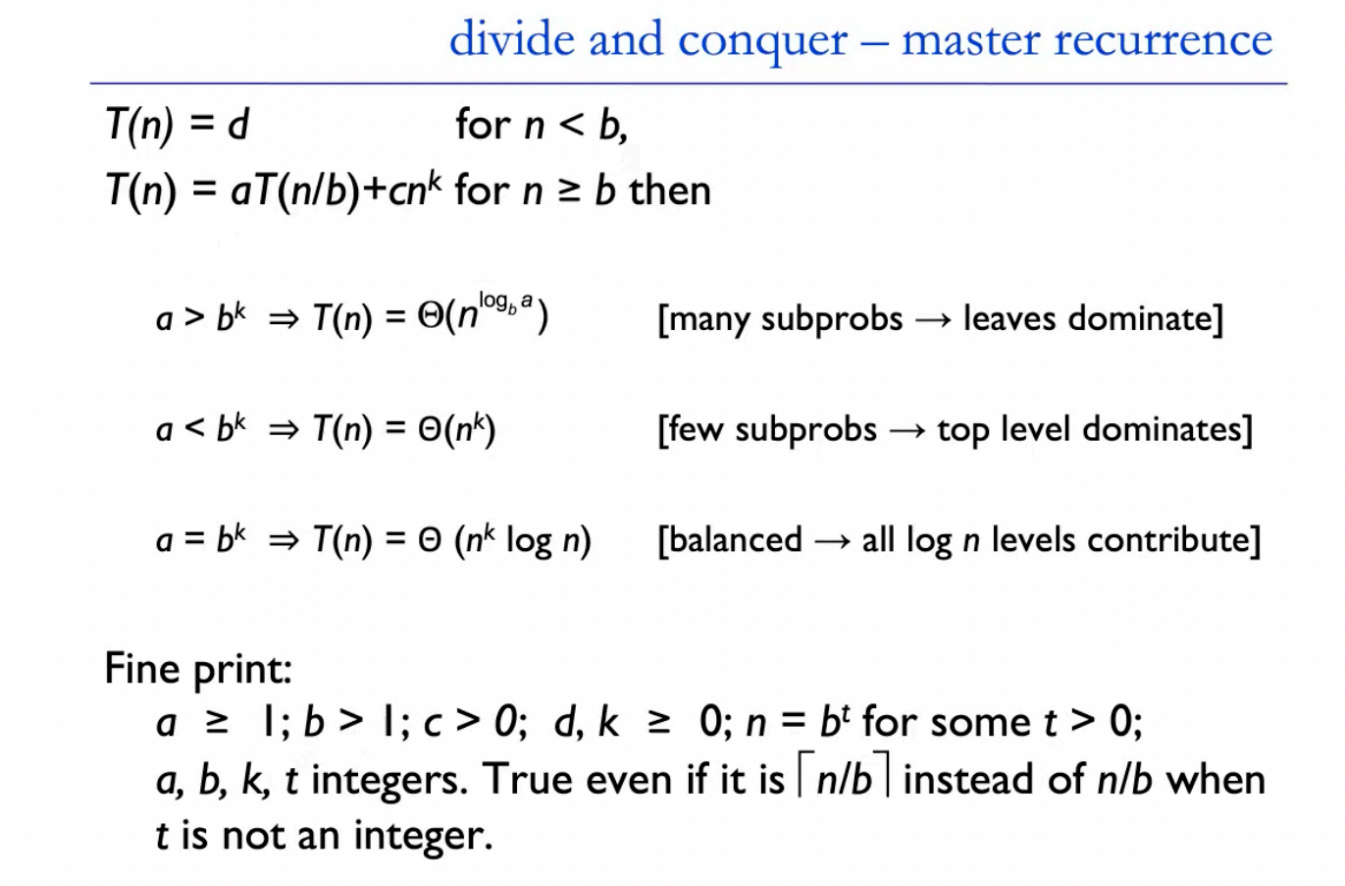
$$
\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585})
$$

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

```
+ Class 6.2 2/9
```

## Master Recurrence

---




```
+ Class 6.3 2/11
```

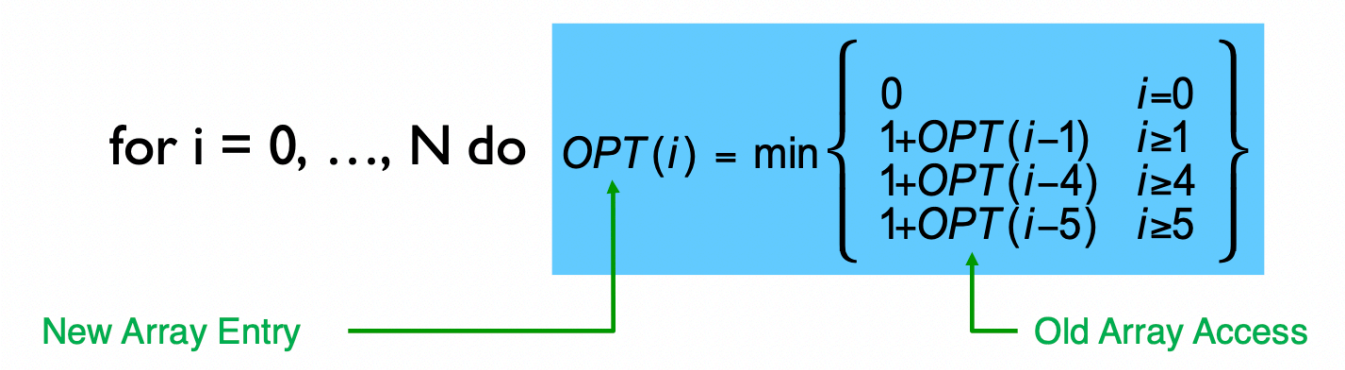
# Dynamic Programmin

## Intro

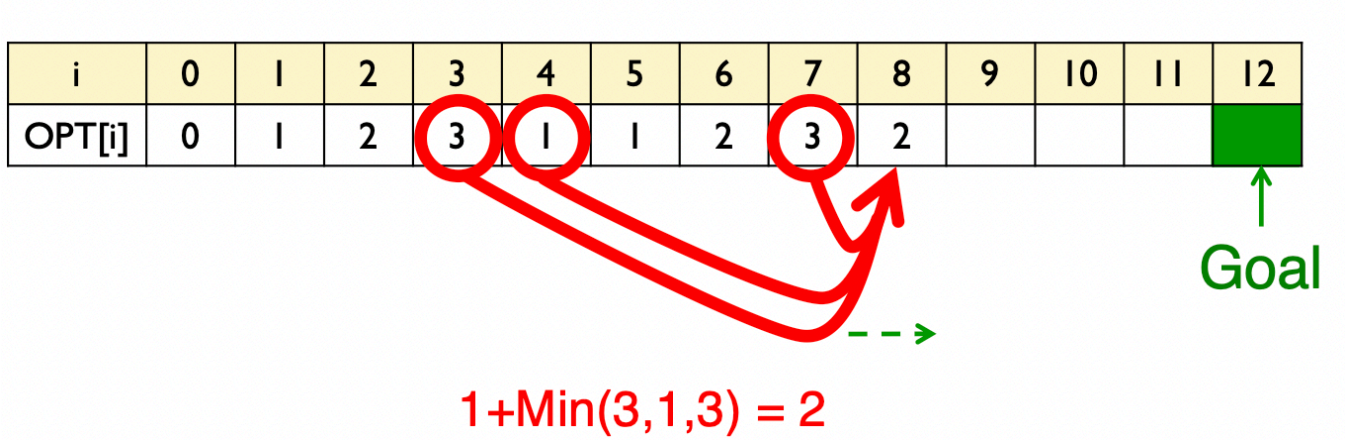
```
+ Class 7.1 2/14
```

Eg. Give the elements 1, 4, 5, Find the min number of element to get 27.

| 5 | 4 | 1 | total number |
|---|---|---|---|
| 5 | 0 | 2 | 7 |
| 4 | 1 | 3 | 8 |
| 3 | 3 | 0 | 6 |



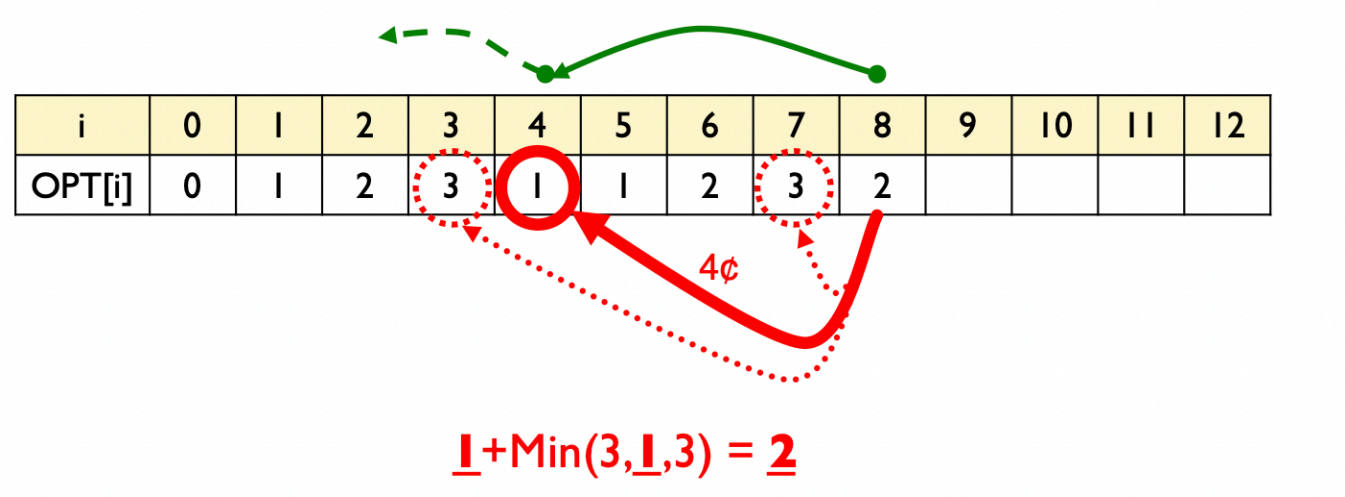

OPT($i$) is the min number of stamps totaling $i$




Run time: $O(n)$

In this way, we do not know which number is the elements, but we can use trace back

**Trace-Back**



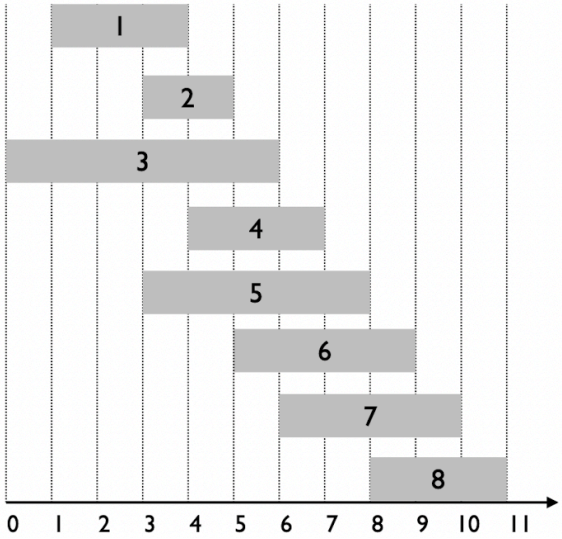

# Schedule

## Weighted Interval Scheduling

Jobs with start time, end time, and value. Time may be confilict, find most valueable sequency.

```
Input: n, s1,…,sn , f1,…,fn , v1,…,vn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

Compute p(1), p(2), …, p(n)

Iterative-Compute-Opt {
   OPT[0] = 0
   for j = 1 to n
      OPT[j] = max(vj + OPT[p(j)], OPT[j-1])
}

Output OPT[n]
```

Exercise: try other concrete examples:
If all vj=1: greedy by finish time ➛ 1,4,8
what if v2 > v1?, but < v1+v4?
v2>v1+v4, but v2+v6 < v1+v7, say? etc.

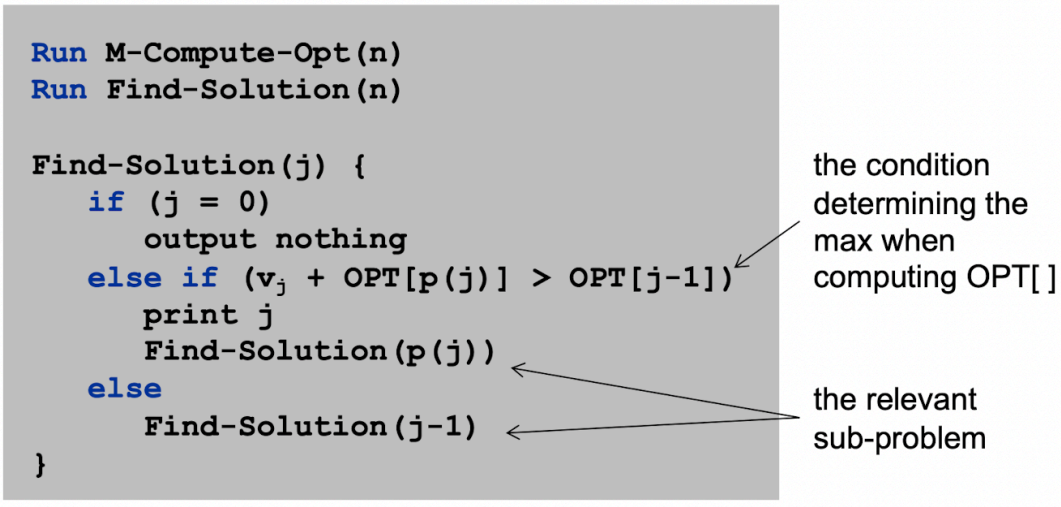


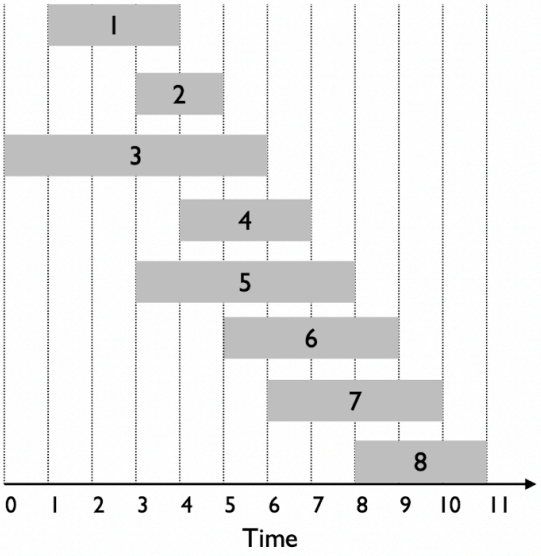

| j | pj | vj | max($v_j$+opt[p(j)], opt[j-1]) = | opt[j] |
|---|---|---|---|---|
| 0 | - | - | | 0 |
| 1 | 0 | 2 | max(2+0, 0) = | 2 |
| 2 | 0 | 3 | max(3+0, 2) = | 3 |
| 3 | 0 | 1 | max(1+0, 3) = | 3 |
| 4 | 1 | 6 | max(6+2, 3) = | 8 |
| 5 | 0 | 9 | max(9+0, 8) = | 9 |
| 6 | 2 | 7 | max(7+3, 9) = | 10 |
| 7 | 3 | 2 | max(2+3, 10) = | 10 |
| 8 | 5 | ? | max(?+9, 10) = | ? |

The job is sorted by the end time. $p_j$ is last possible job. $v_j$ is the value.

```
+ Class 7.2 2/16
```

That can only find the optimal value, so we need use traceback to find the solution which job has been counted.

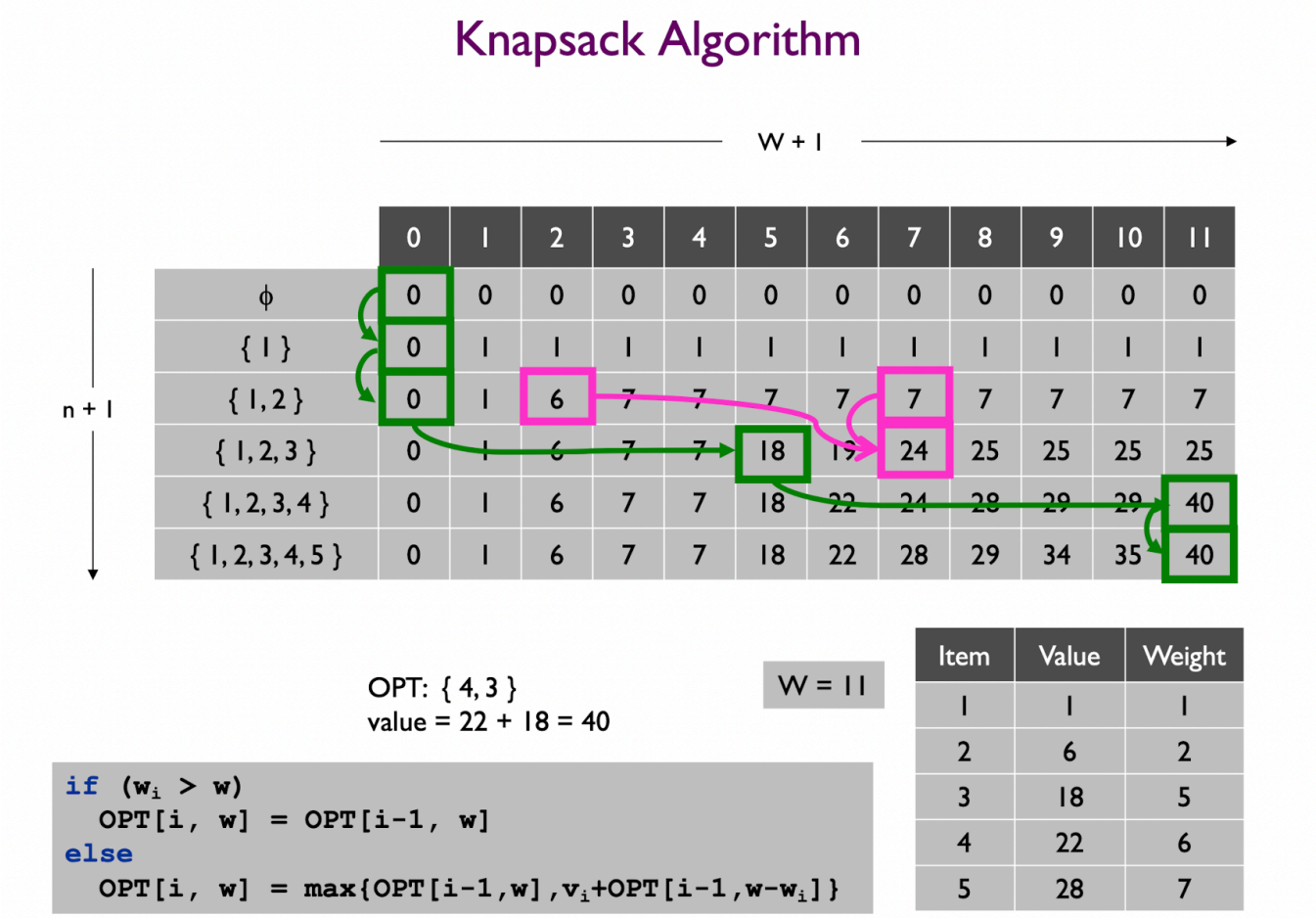





| j | pj | vj | max($v_j$+opt[p(j)], opt[j-1]) = | opt[j] |
|---|---|---|---|---|
| 0 | - | - | - | 0 |
| 1 | 0 | 2 | max(2+0, 0) = | 2 |
| 2 | 0 | 3 | max(3+0, 2) = | 3 |
| 3 | 0 | 1 | max(1+0, 3) = | 3 |
| 4 | 1 | 6 | max(6+2, 3) = | 8 |
| 5 | 0 | 9 | max(9+0, 8) = | 9 |
| 6 | 2 | 7 | max(7+3, 9) = | 10 |
| 7 | 3 | 2 | max(2+3, 10) = | 10 |
| 8 | 5 | .1 | max(0.1+9, 10) = | 10 |

V8 = 0.1 is *ex*cluded; opt solution is v6+v2

# Knapsack Algorithm




Last number in line 5: 40 = max(40, 28+7)

Red line: 24 = max(6+18, 7) =24 *where 6: 2{1,2} 18: $v_3$

```
+ Class 7.3 2/18
```

# RNA

## Dynamic Programming Over Intervals: (R. Nussinov's algorithm)

Notation. OPT[i, j] = maximum number of base pairs in a secondary structure of the substring $b_i b_{i+1} \ldots b_j$.

- Case 1a. If $i \geq j - 4$ (and base $b_j$ is not paired):

  OPT[i, j] = 0 by no-sharp turns condition.

- Case 1b. If $i < j - 4$, but base $b_j$ is not paired:

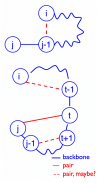
  OPT[i, j] = OPT[i, j-1]

Key point: Either last base is unpaired (case 1a,b) or paired (case 2)

- Case 2. Base $b_j$ pairs with $b_t$ for some $i \leq t < j - 4$.
  non-crossing constraint decouples resulting sub-problems

  $OPT[i, j] = 1 + \max_t \{ OPT[i, t-1] + OPT[t+1, j-1] \}$

take max over t such that $i \leq t < j-4$ and $b_t$ and $b_j$ are Watson-Crick complements




```
+ Class 8.1 holiday
+ Class 8.2 2/23
```