CSEP 521
Applied Algorithms

Richard Anderson

Lecture 6

Dynamic Programming

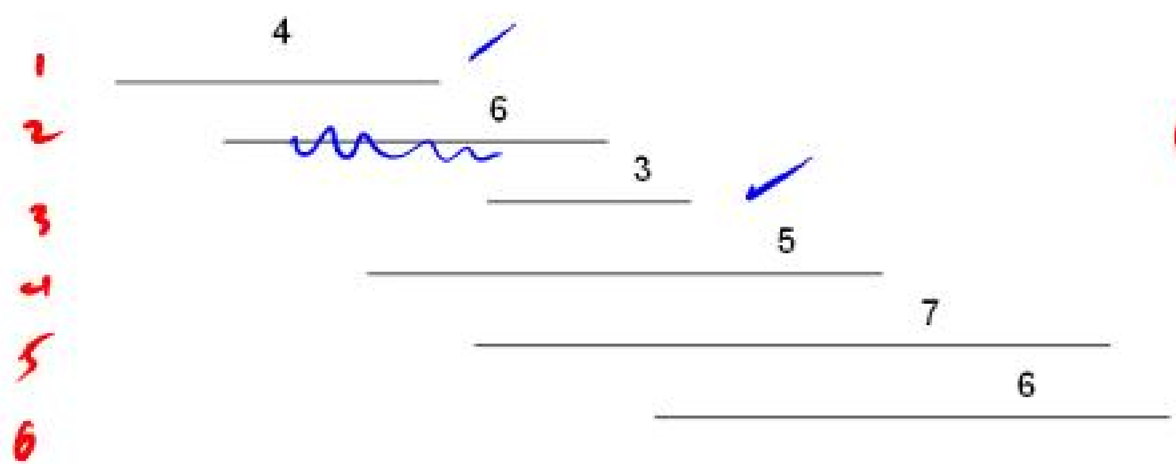
Announcements

- ~~Midterm today!~~
  - ~~60 minutes, start of class, closed book~~
- Reading for this week
  - 6.1, 6.2, 6.3., 6.4
- Makeup lecture
  - February 19, 6:30 pm.
    - ~~Still waiting on confirmation on MS room.~~

Tuesday
UW, Microsoft

Dynamic Programming

- Weighted Interval Scheduling
- Given a collection of intervals $I_1,\ldots,I_n$ with weights $w_1,\ldots,w_n$, choose a maximum weight set of non-overlapping intervals



Opt

$Opt_j \quad I_1, I_2, \ldots I_j$

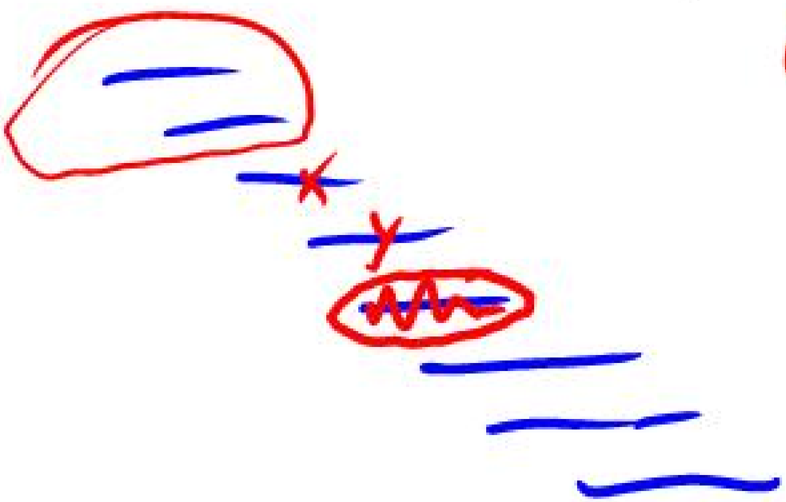
Opt_j – function($Opt_{j-1}, Opt_{j-2}, \ldots Opt_1$)

P[1] = 0
P[2] = 0
P[3] = 1

Optimality Condition

- Opt[j] is the maximum weight independent set of intervals $I_1, I_2, \ldots, I_j$
- $\boxed{\text{Opt[j]}}$ = max(Opt[j – 1], w_j + Opt[p[j]])
  - Where p[j] is the index of the last interval which finishes before I_j starts

Opt[5] – Include I_5
– Exclude I_5



Algorithm

```
MaxValue(j) =
  if j = 0 return 0
  else
      return max( MaxValue(j-1),
                  w_j + MaxValue(p[ j ]))
```

Worst case run time: 2^n

A better algorithm

M[j] initialized to -1 before the first recursive call for all j

```
MaxValue(j) =
    if j = 0 return 0;
    else if M[ j ] != -1 return M[ j ];
    else
        M[ j ] = max(MaxValue(j-1), w_j + MaxValue(p[ j ]));
        return M[ j ];
```

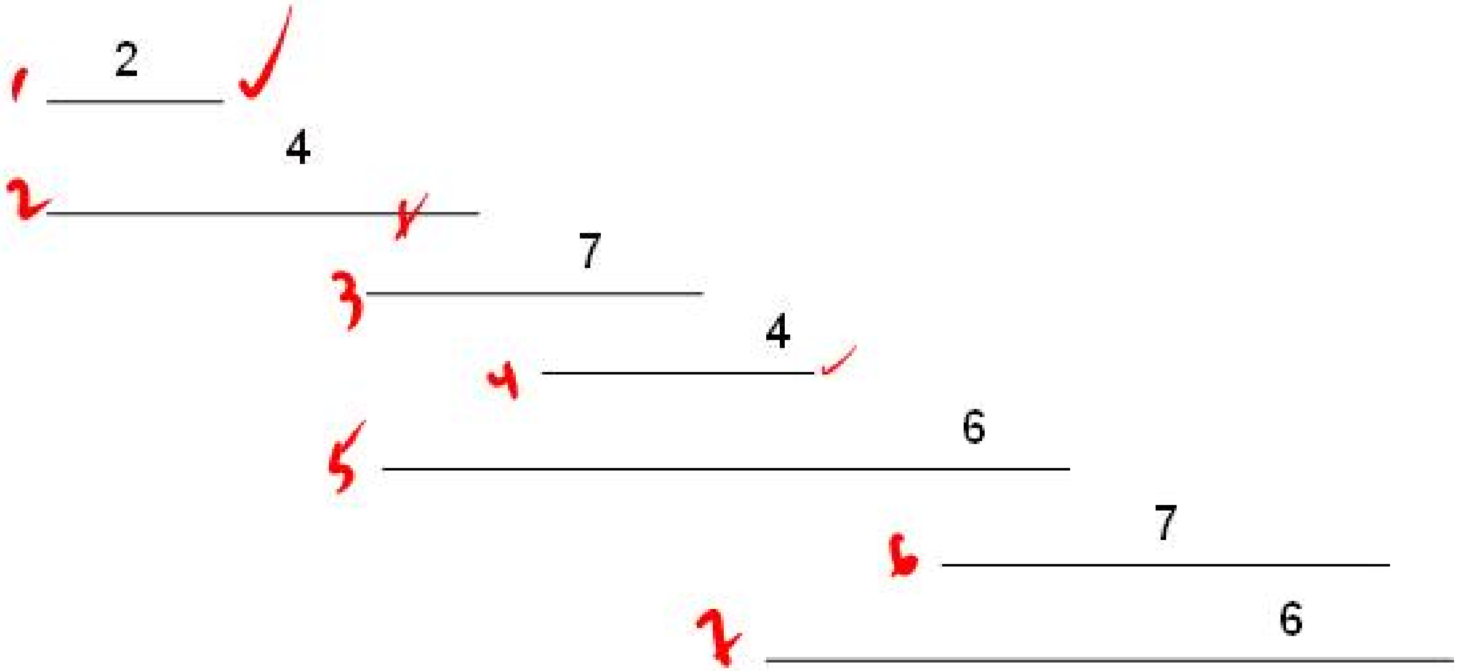
Iterative version

```
MaxValue (j)  {
        M[ 0 ] = 0;
        for (k = 1; k <= j; k++){
                M[ k ] = max(M[ k-1 ], w_k + M[ P[ k ] ]);
        return M[ j ];
}
```

Fill in the array with the Opt values

$$Opt[\,j\,] = \max\,(Opt[\,j-1], w_j + Opt[\,p[\,j\,]\,])$$

1	2	3	4	5	6	7
2	4	9	9	9	16	16



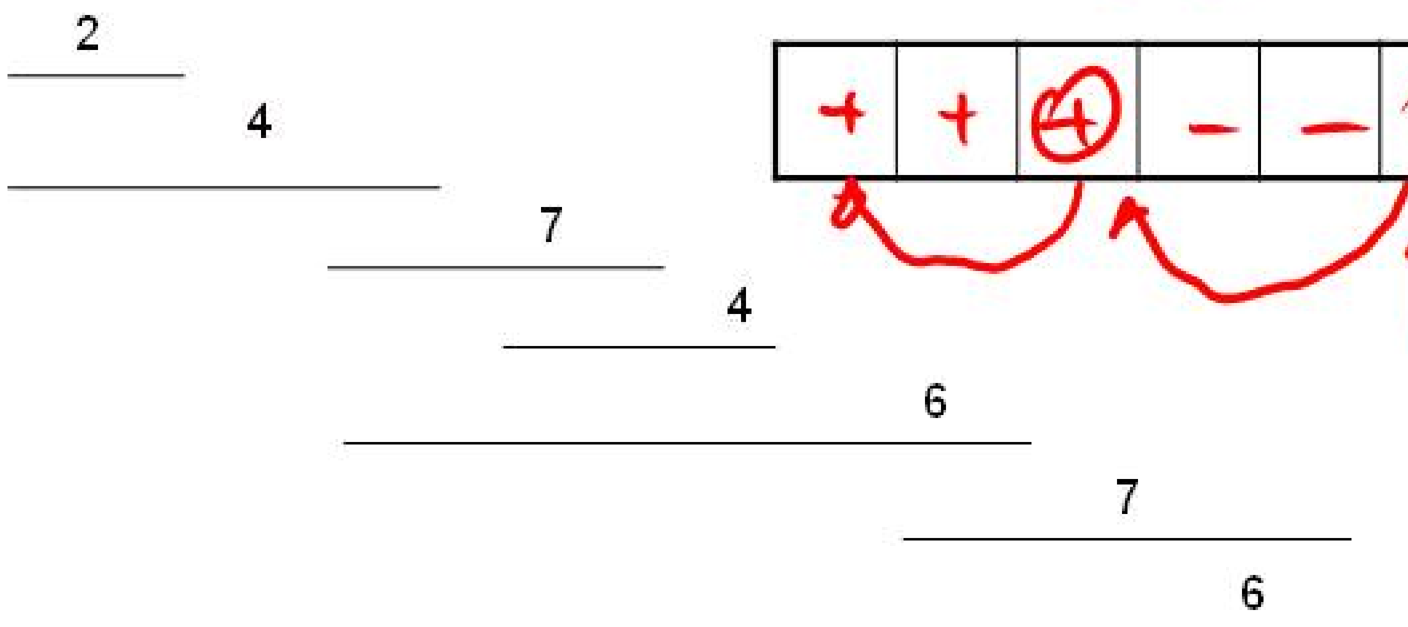
Computing the solution

Opt[j] = max (Opt[j – 1], w_j + Opt[p[j]])

Record which case is used in Opt computation

2	4	7	9	9	16	16

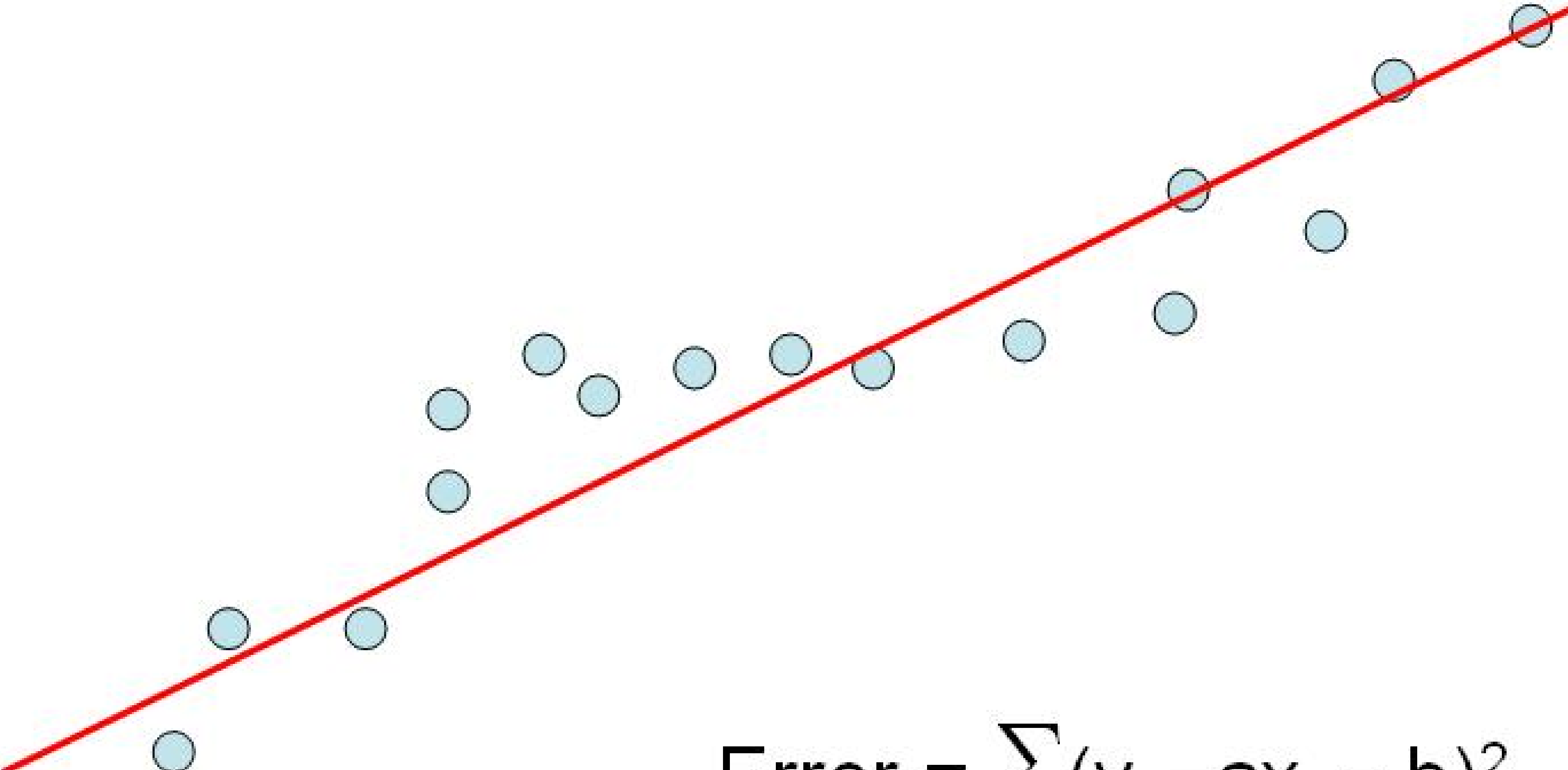
+	+	+	–	–	+	–



Dynamic Programming

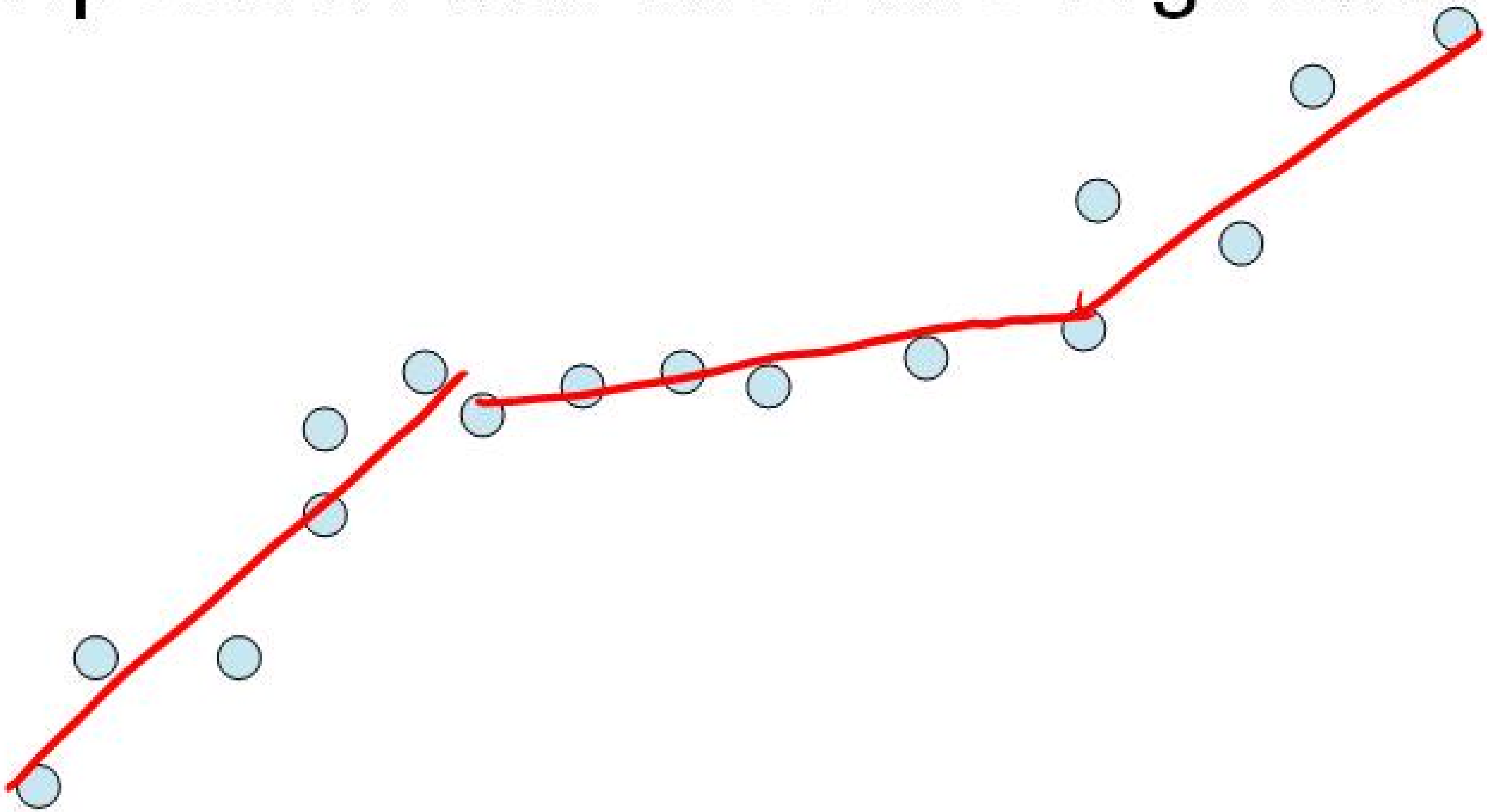
- The most important algorithmic technique covered in CSEP 521
- Key ideas
  - Express solution in terms of a polynomial number of sub problems
  - Order sub problems to avoid recomputation

Optimal linear interpolation

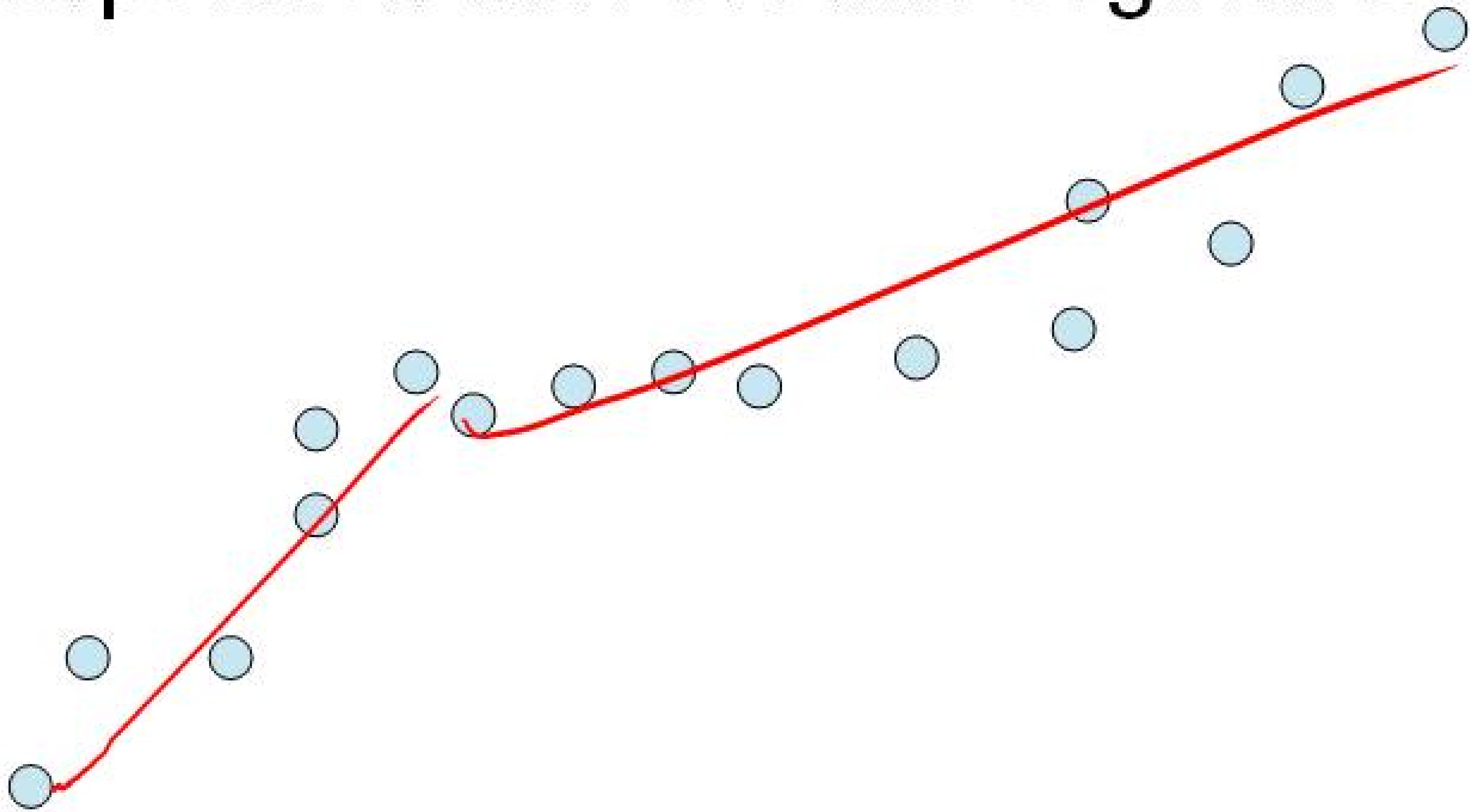


$$\text{Error} = \sum (y_i - ax_i - b)^2$$

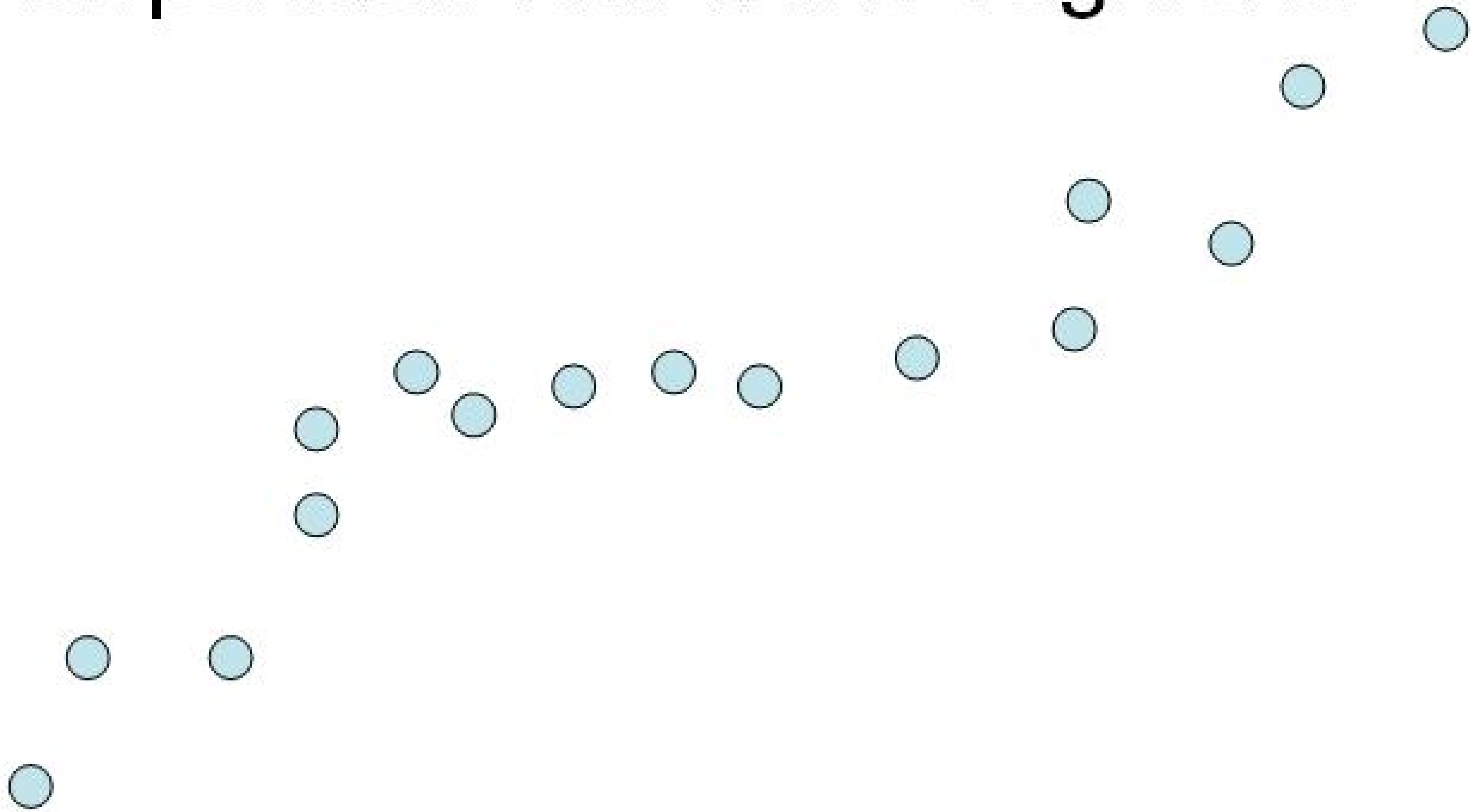
What is the optimal linear interpolation with three line segments



What is the optimal linear interpolation with two line segments



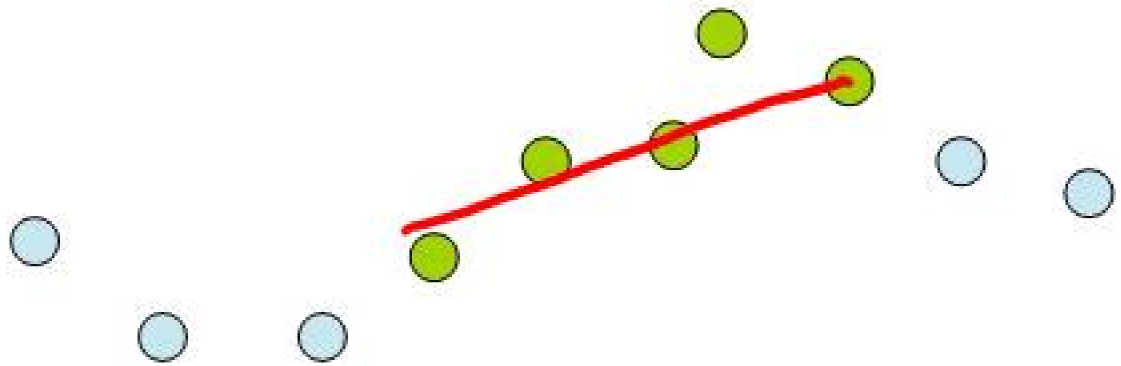
What is the optimal linear interpolation with n line segments



Notation

E_{ij}

- Points $p_1, p_2, \ldots, p_n$ ordered by x-coordinate ($p_i = (x_i, y_i)$)
- $E_{i,j}$ is the least squares error for the optimal line interpolating $p_i, \ldots p_j$



Optimal interpolation with two segments

- Give an equation for the optimal interpolation of $p_1, \ldots, p_n$ with two line segments

$$Opt_2 = \min_{K} \left\{ E_{1K} + E_{Kn} \right\}$$

- $E_{i,j}$ is the least squares error for the optimal line interpolating $p_i, \ldots p_j$

Optimal interpolation with k segments

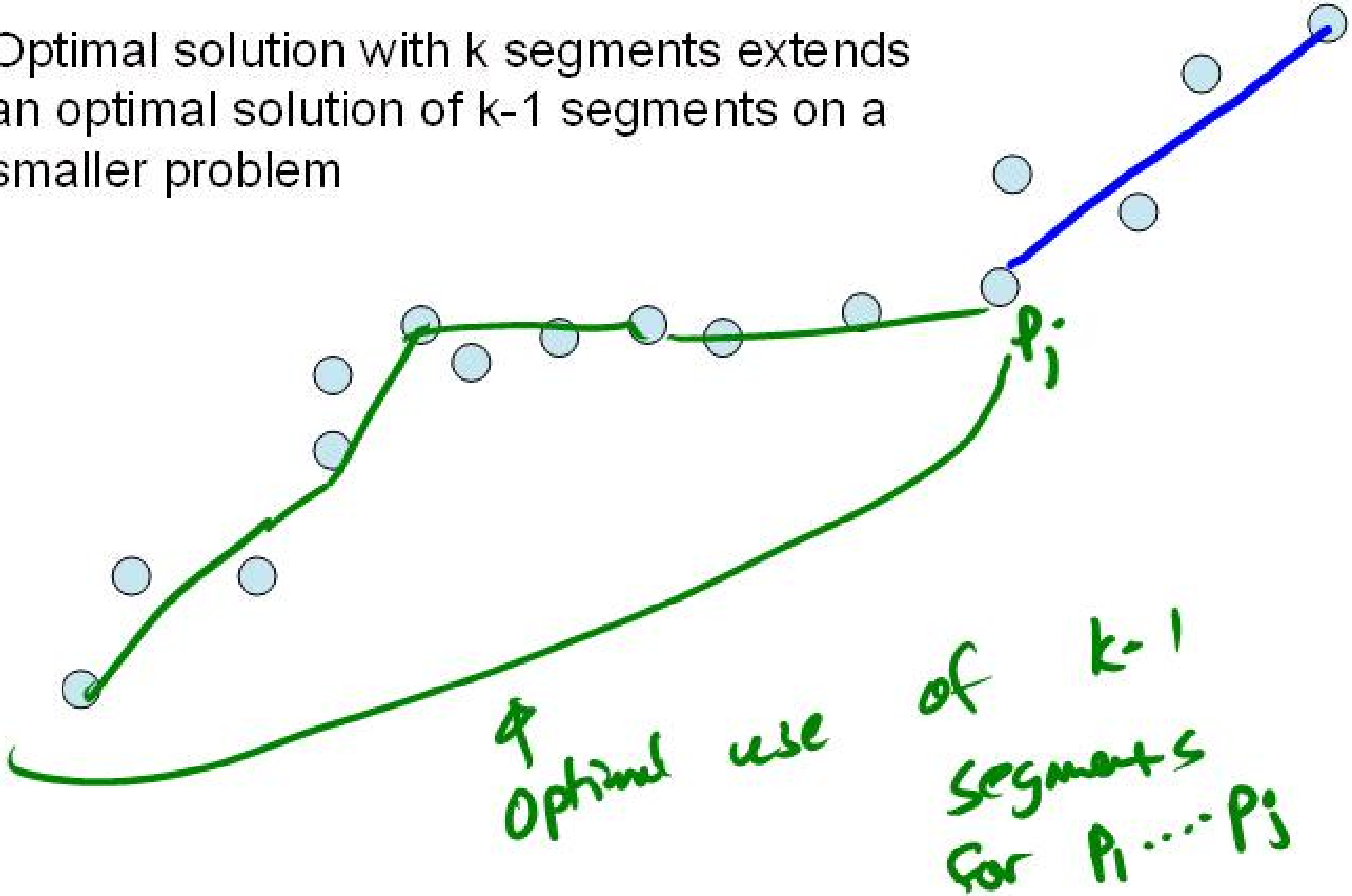
- Optimal segmentation with three segments
  - $\text{Min}_{i,j}\{E_{1,i} + E_{i,j} + E_{j,n}\}$
  - $O(n^2)$ combinations considered
- Generalization to k segments leads to considering $O(n^{k-1})$ combinations

$Opt_k[\, j\,]$: Minimum error approximating $p_1 \ldots p_j$ with k segments

How do you express $Opt_k[\, j\,]$ in terms of $Opt_{k-1}[1], \ldots, Opt_{k-1}[\, j\,]$?

Optimal sub-solution property

Optimal solution with k segments extends an optimal solution of k-1 segments on a smaller problem



Optimal multi-segment interpolation

Compute Opt[k, j] for $0 < k < j < n$

```
for j := 1 to n
    Opt[ 1, j] = E_{1,j};
for k := 2 to n-1
    for j := 2 to n
        t := E_{1,j}
        for i := 1 to j -1
            t = min (t, Opt[k-1, i ] + E_{i,j})
        Opt[k, j] = t
```

$P_1 \; P_2$

$P_1 \quad P_3$

$P_1 \cdots P_4$

$\vdots$

$P_1 \cdots P_n$

$\uparrow$

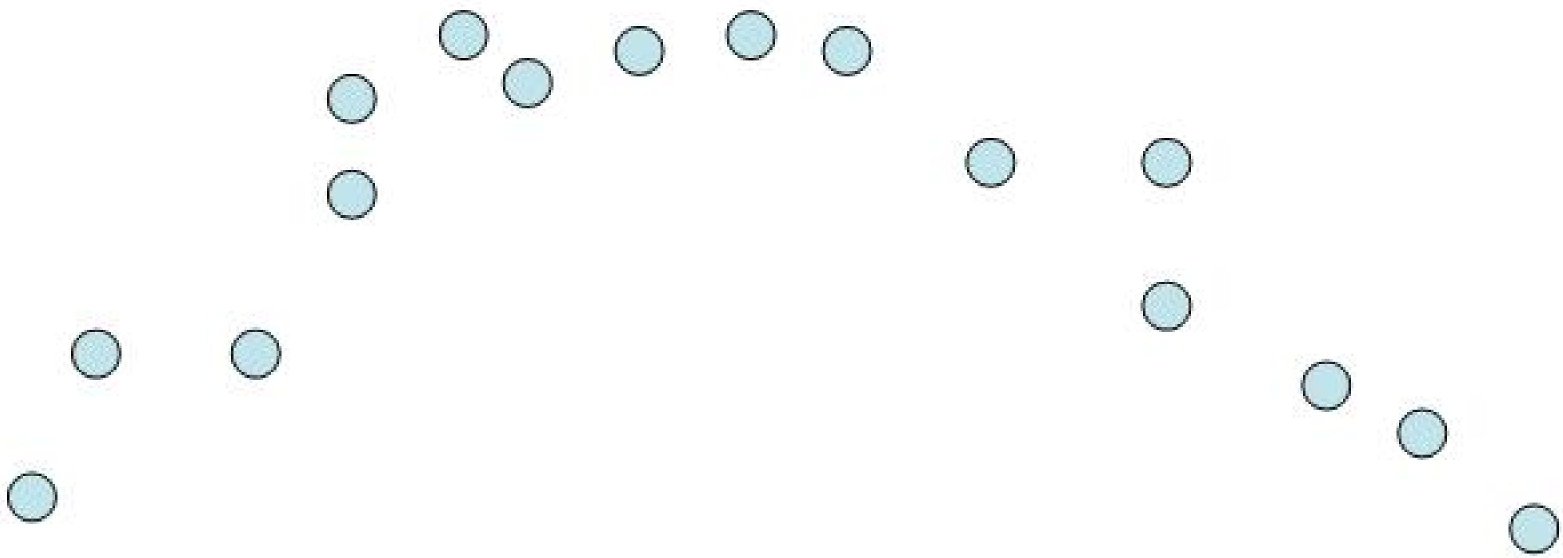
$K-1$

Determining the solution

- When Opt[k,j] is computed, record the value of i that minimized the sum
- Store this value in a auxiliary array
- Use to reconstruct solution

Variable number of segments

- Segments not specified in advance
- Penalty function associated with segments
- Cost = Interpolation error + C x #Segments



Penalty cost measure

- $Opt[\ j\ ] = \min(E_{1,j}, \min_i(Opt[\ i\ ] + E_{i,j} + P))$

Subset Sum Problem

- Let $w_1, \ldots, w_n = \{6, 8, 9, 11, 13, 16, 18, 24\}$
- Find a subset that has as large a sum as possible, without exceeding 50

Adding a variable for Weight

- Opt[j, K] the largest subset of $\{w_1, \ldots, w_j\}$ that sums to at most K
- {2, 4, 7, 10}
  - Opt[2, 7] = 6 {2, 4}
  - Opt[3, 7] = 7 {7}
  - Opt[3,12] = 11 {4, 7}
  - Opt[4,12] = 12 {2, 10}

Subset Sum Recurrence

- Opt[j, K] the largest subset of $\{w_1, \ldots, w_j\}$ that sums to at most K

$$Opt[j, K] = \max[\, Opt[j-1, K],\ w_j + Opt[j-1, K - w_j] \,]$$

Runtime $O(nK)$

Subset Sum Grid

$$Opt[j, K] = \max(Opt[j-1, K], Opt[j-1, K-w_j] + w_j)$$

4	0	2	2	4	4	6	7	7	9	10	11	12	13	14	14	16	17
3	0	2	2	4	4	6	7	7	9	9	11	11	13	13	13	13	13
Start → 2	0	2	2	4	4	6	6	6	6	6	6	6	6	6	6	6	6
1	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

{2, 4, 7, 10}

Knapsack Problem

- Items have weights and values
- The problem is to maximize total value subject to a bound on weght
- Items $\{I_1, I_2, \ldots I_n\}$
  - Weights $\{w_1, w_2, \ldots, w_n\}$
  - Values $\{v_1, v_2, \ldots, v_n\}$
  - Bound K
- Find set S of indices to:
  - Maximize $\sum_{i \in S} v_i$ such that $\sum_{i \in S} w_i <= K$

$v_1, v_2, \ldots v_K$

Knapsack Recurrence

Subset Sum Recurrence:

$$Opt[\, j, K] = \max(Opt[\, j - 1, K], Opt[\, j - 1, K - w_j] + w_j)$$

Knapsack Recurrence:

$$Opt[i, K] = \max(Opt[j-1, K], Opt[j-1, K-w_j] + v_j)$$

$K = \Sigma w_i$

Knapsack Grid

$$Opt[j, K] = \max(\underbrace{Opt[j-1, K]}, \underbrace{Opt[j-1, K-w_j] + v_j})$$

4																	
3	0	3	3	5	5	8	9	9	12	.	.	.	.		-		
2	0	3	3	5	5	8	8	8	8	8	8	6	8	8	8	8	8
1	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Weights {2, 4, 7, 10} Values: {3, 5, 9, 16}