CSE 421
Algorithms

Richard Anderson

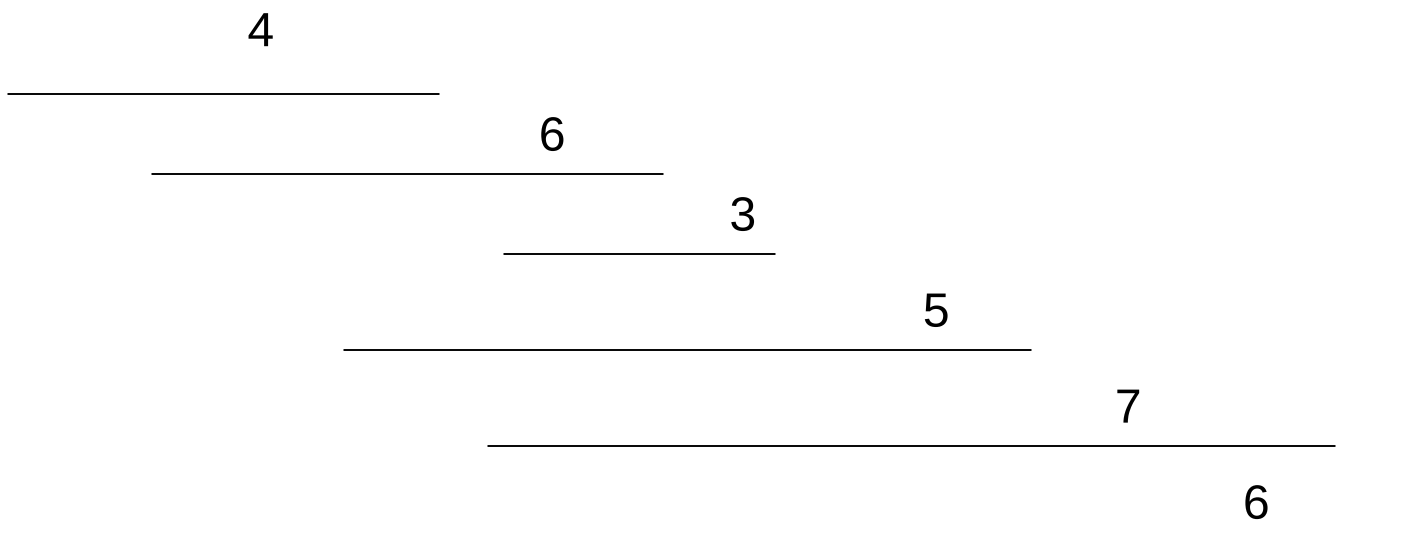
Lecture 16, Winter 2019

Dynamic Programming

Intervals sorted by end time

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



Intervals sorted by end time

Optimality Condition

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

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 ];
```

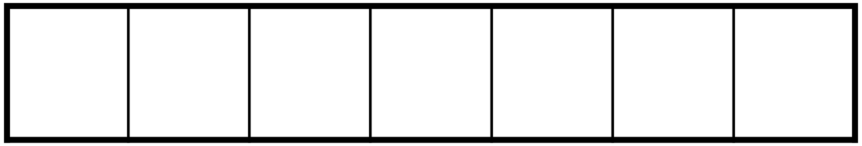
Iterative Algorithm

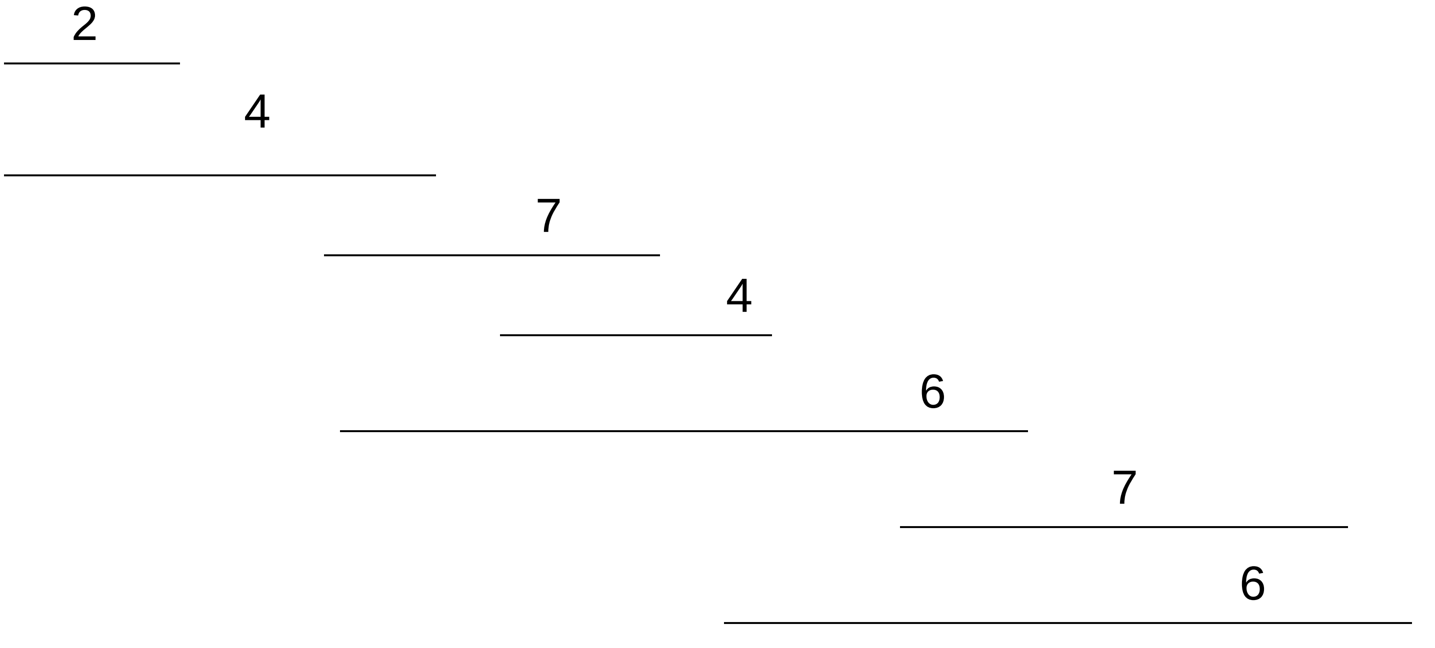
Express the MaxValue algorithm as an iterative algorithm

```
MaxValue {



}
```

Fill in the array with the Opt values

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

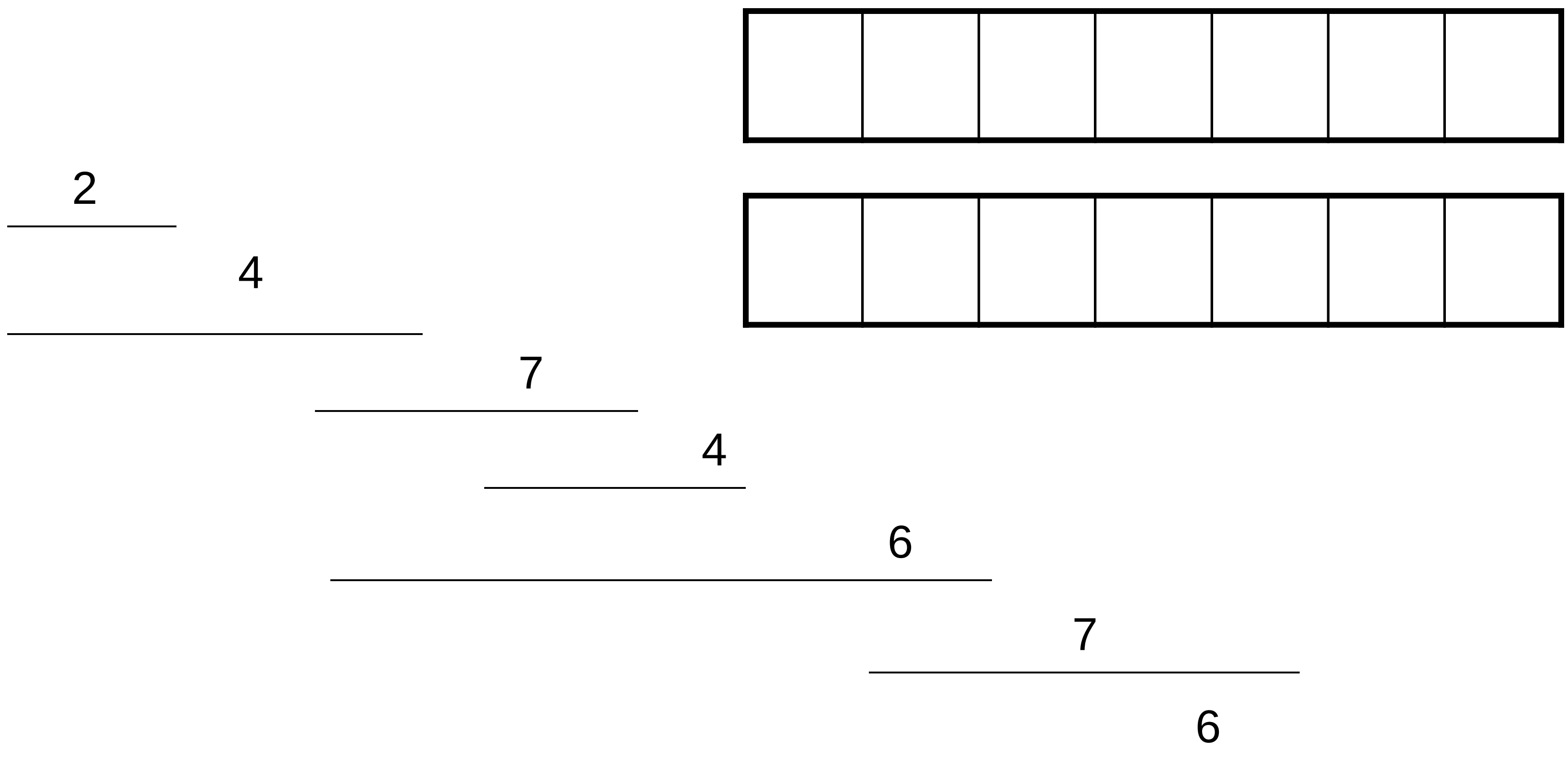




Computing the solution

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

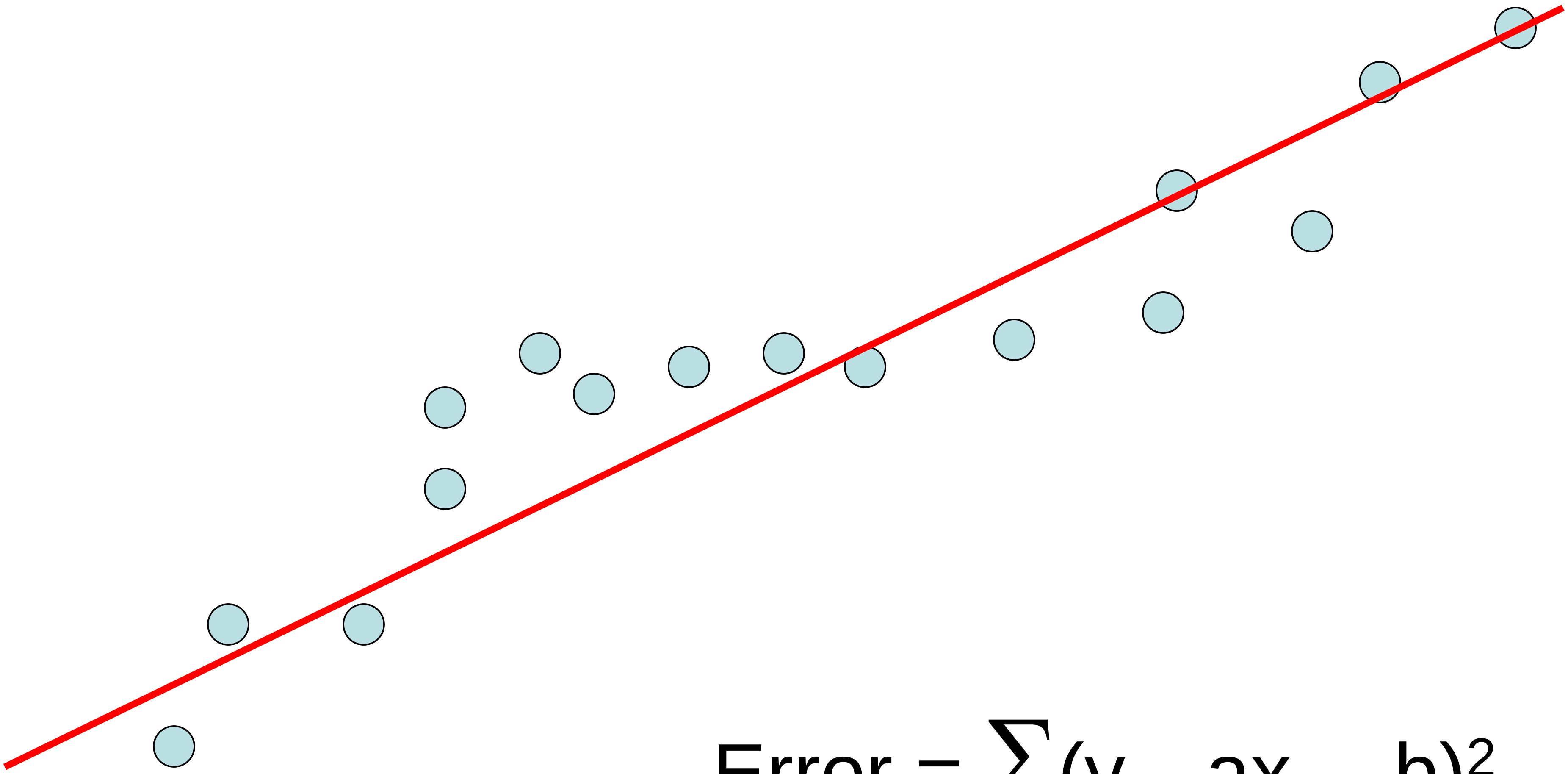
Record which case is used in Opt computation



Dynamic Programming

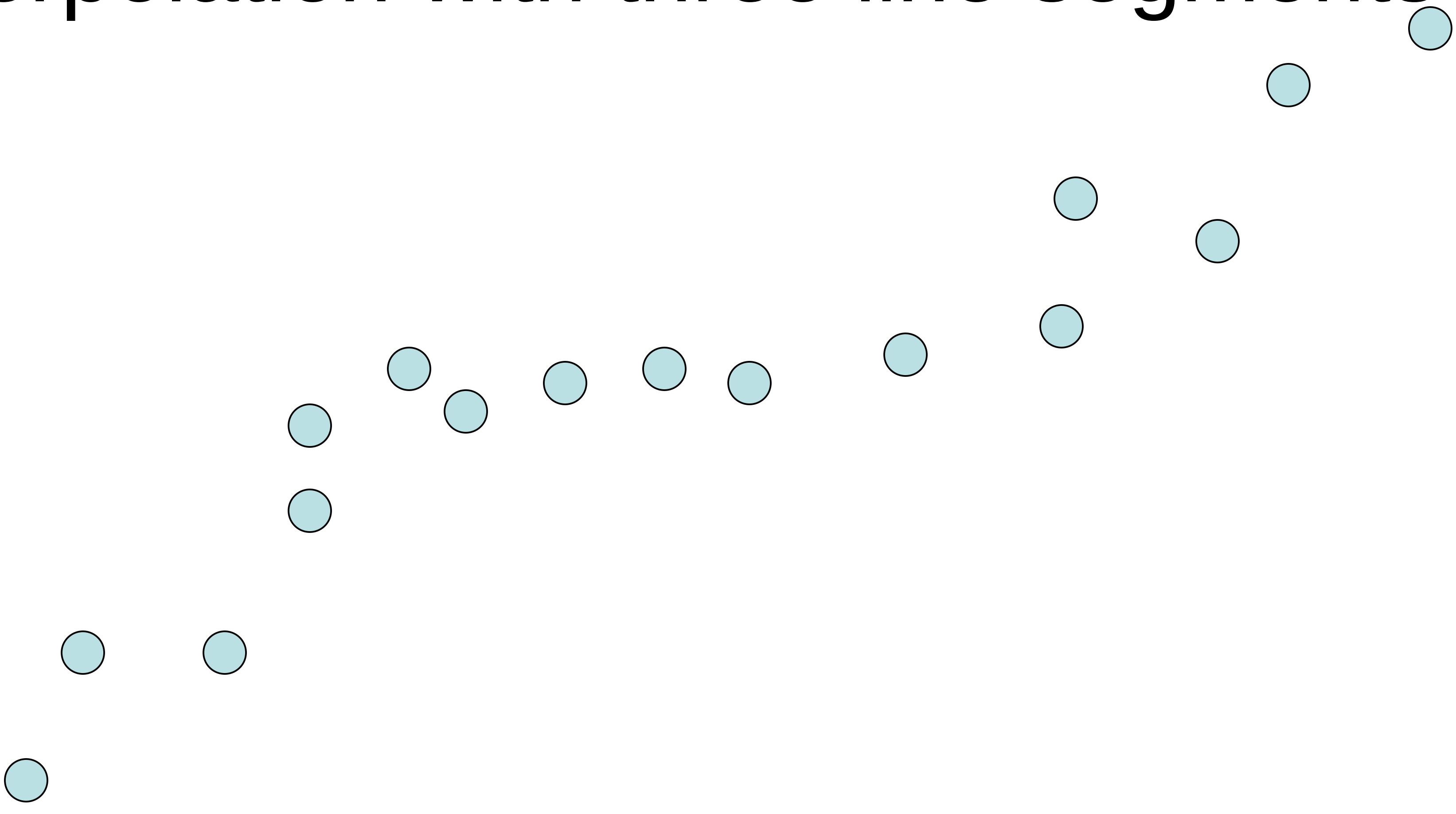
- The most important algorithmic technique covered in CSE 421
- Key ideas
  - Express solution in terms of a polynomial number of sub problems
  - Order sub problems to avoid recomputation

Optimal linear interpolation

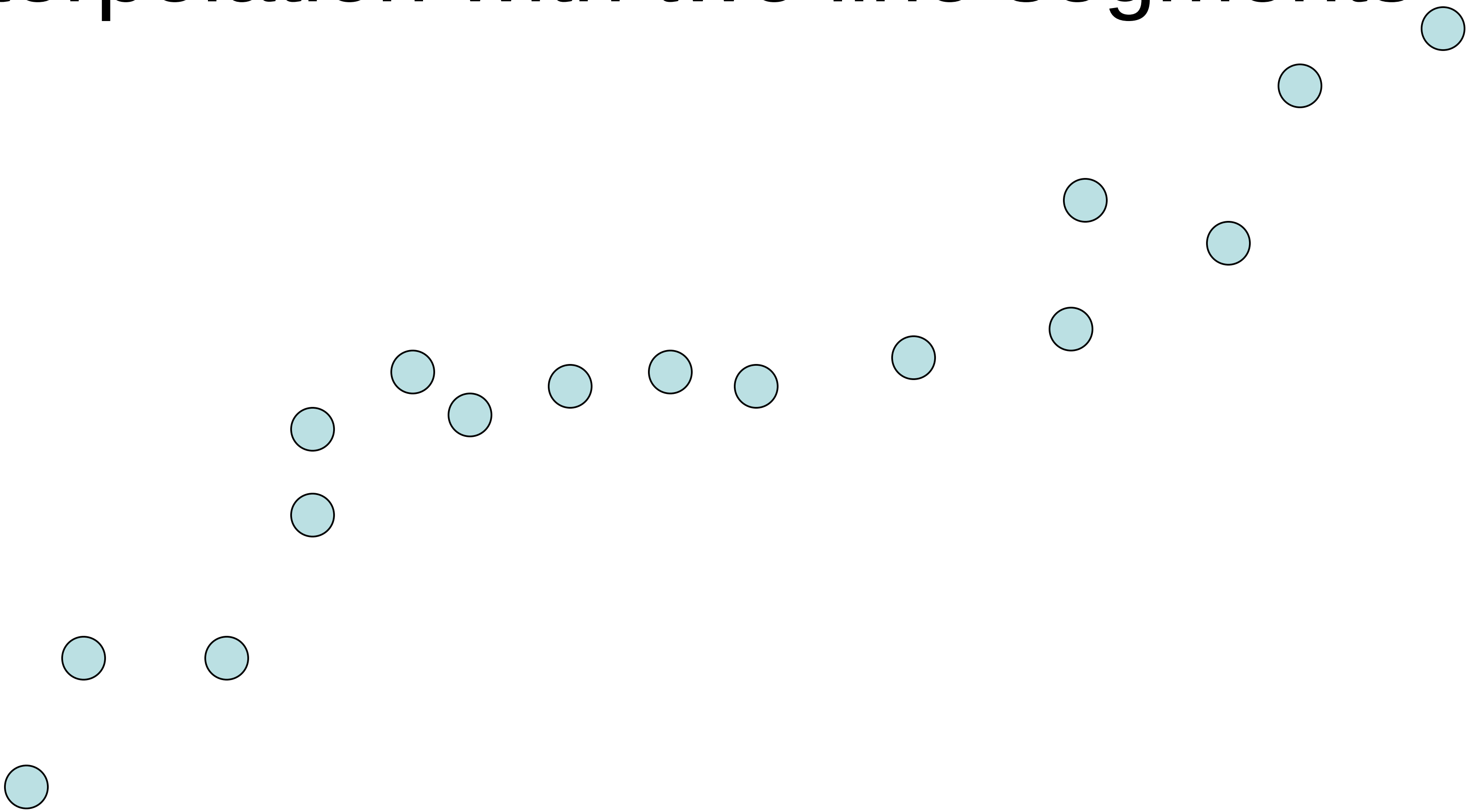


$$\text{Error} = \sum (y_i - ax_i - b)^2$$

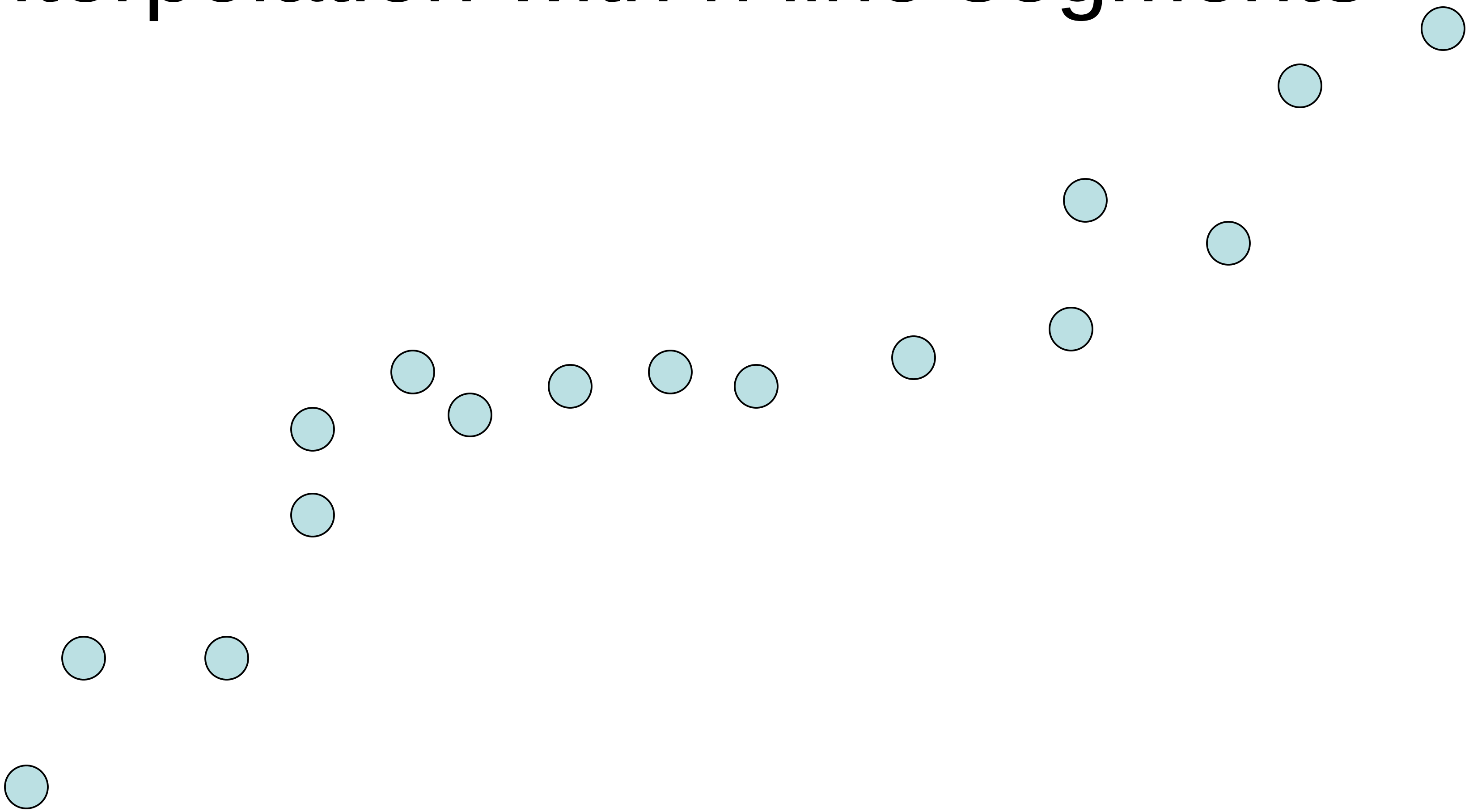
What is the optimal linear interpolation with three line segments



What is the optimal linear interpolation with two line segments

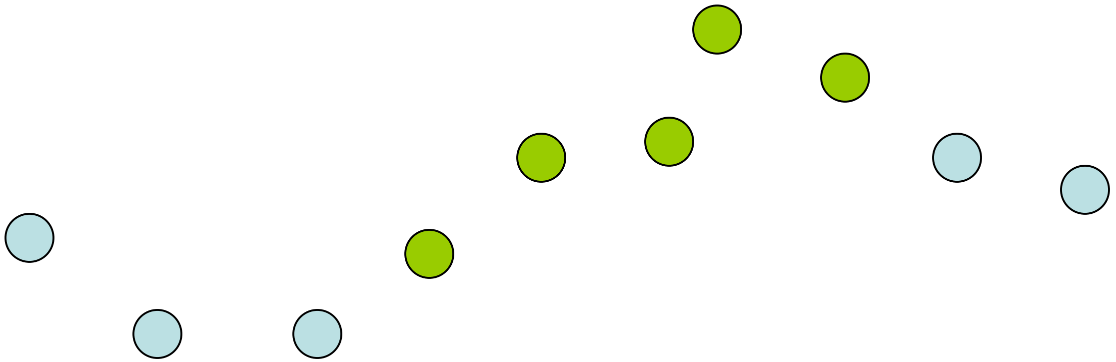


What is the optimal linear interpolation with n line segments



Notation

- Points p_1, p_2, . . ., 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, . . . p_j



Optimal interpolation with two segments

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

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

Optimal interpolation with k segments

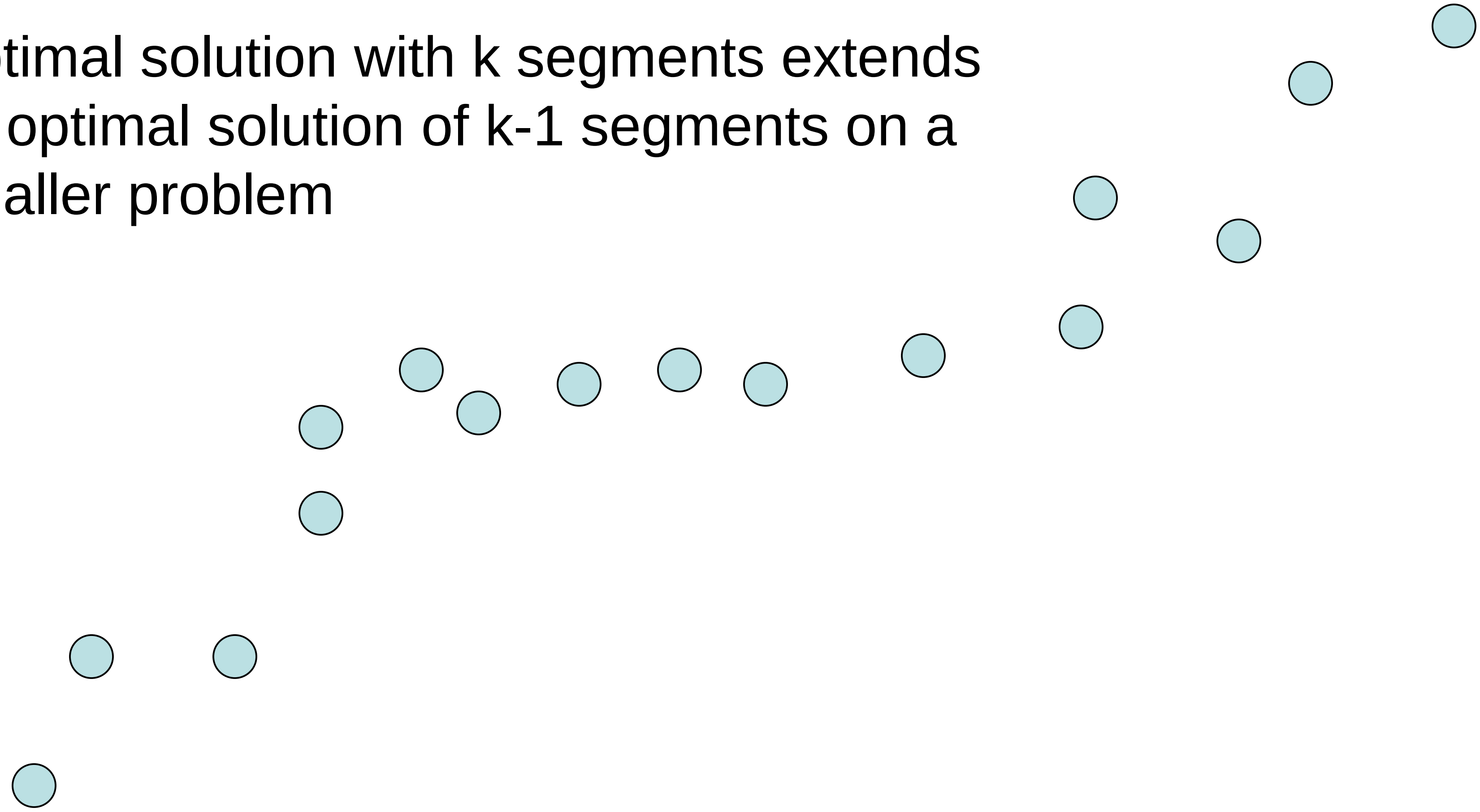
- Optimal segmentation with three segments
  - $\text{Min}_{i,j}\{E_{1,i} + E_{i,j} + E_{j,n}\}$
  - $O(n^2)$ combinations considered
- Generalization to k segments leads to considering $O(n^{k-1})$ combinations

$Opt_k[\ j\ ]$: 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 $0 < k < j < n$

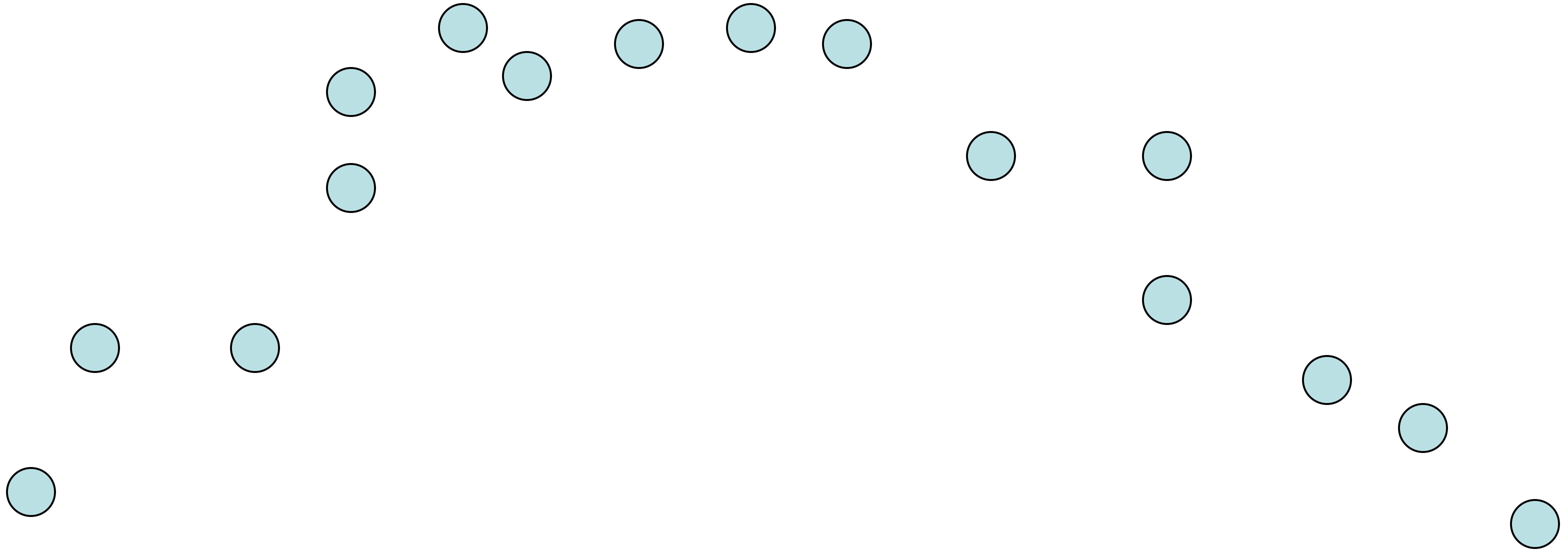
for j := 1 to n
    Opt[1, j] = $E_{1,j}$;
for k := 2 to n-1
    for j := 2 to n
        t := $E_{1,j}$
        for i := 1 to j -1
            t = min (t, Opt[k-1, i] + $E_{i,j}$)
        Opt[k, j] = t

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

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