Dynamic Programming

Today:

- Weighted Interval Scheduling
- Segmented Least Squares

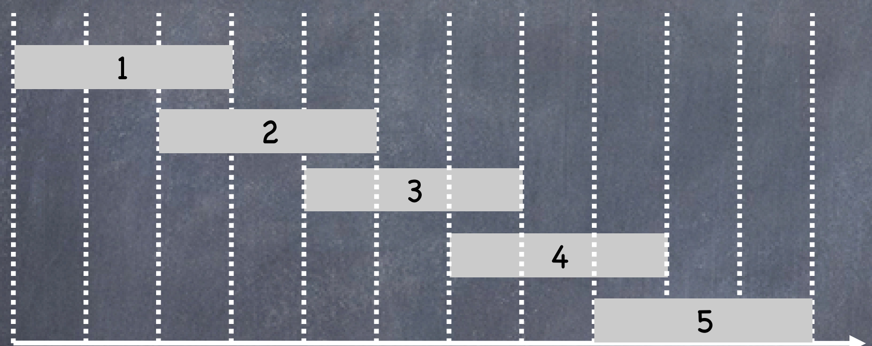
Weighted Interval Scheduling

Recursive Algorithm

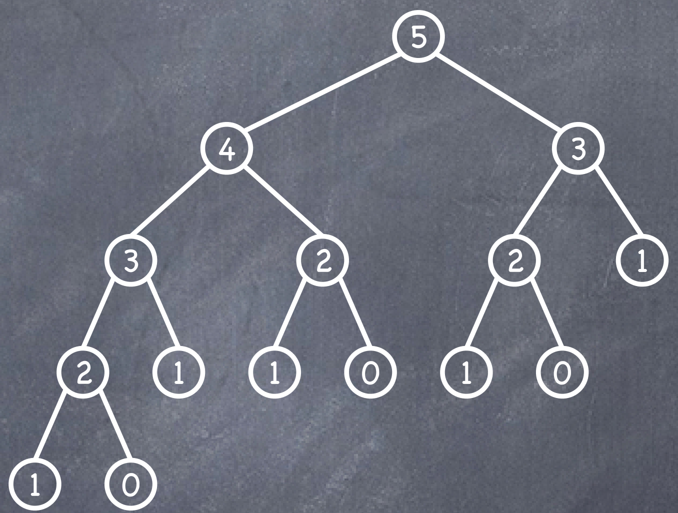
```
Compute-Opt(j) {
   if j == 0 then
      return 0
   else
      return max(v_j + Compute-Opt(p(j)), Compute-Opt(j-1))
   end
}
```

Running time?

Worst Case Running Time



p(1) = 0, p(j) = j-2



Worst-case running time is exponential.

Memoization

Store results of each sub-problem in an array.

```
Initialize M[j] to be "empty" for j=1,...,n
M-Compute-Opt(j) {
   if j == 0 then
      return 0
   else if M[j] is empty then
      M[j] = max(w_j + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
   end if
   return M[j]
}
```

This gives O(n) running time!
...but we'll see an even easier approach

Iterative Solution

Solve subproblems in ascending order

```
Iterative-Compute-Opt {
   M[0] = 0
   for j = 1 to n
      M[j] = max(v_j + M[p(j)], M[j-1])
   end
}
```

Running time is obviously $O(n)$

Finding the Solution (Not just value)

- Exercise: suppose you are given the array M, so that M[j] = OPT(j). How can you produce the optimal set of jobs?
- Hint: first decide whether job n is part of optimal solution

Find-Solution

Use the recurrence a second time to "backtrack" through M array

```
Find-Solution(M, j) {
    if j == 0
        return {}
    else if v_j + M[p(j)] > M[j-1] then
        return {j} ∪ Find-Solution(M, p(j))   // case 1
    else
        return Find-Solution(M, j-1)          // case 2
    end
}
```

Call Find-Solution(M, n)

Dynamic Programming "Recipe"

- Recursive formulation of optimal solution in terms of subproblems
- Only polynomially many different subproblems
- Iterate through subproblems in order

Interval scheduling: n subproblems

Segmented Least Squares

A Second Example of Dynamic Programming

Two important questions: (1) how many subproblems? and (2) what does recurrence look like? (how many cases?)

Weighted Interval scheduling

- n subproblems
- Two cases: include j or don't include j

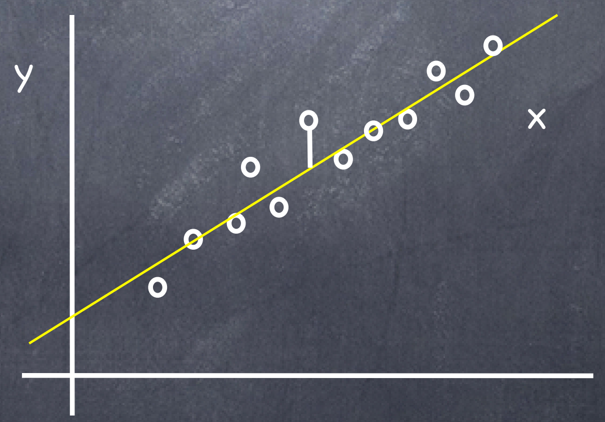
Segmented Least Squares

- n subproblems
- Many cases...

Ordinary Least Squares (OLS)

- Foundational problem in statistics and numerical analysis.
  - Given n points in the plane: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$.
  - Find a line $y = ax + b$ that minimizes the sum of the squared error:

$$SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2$$



Least Squares Solution

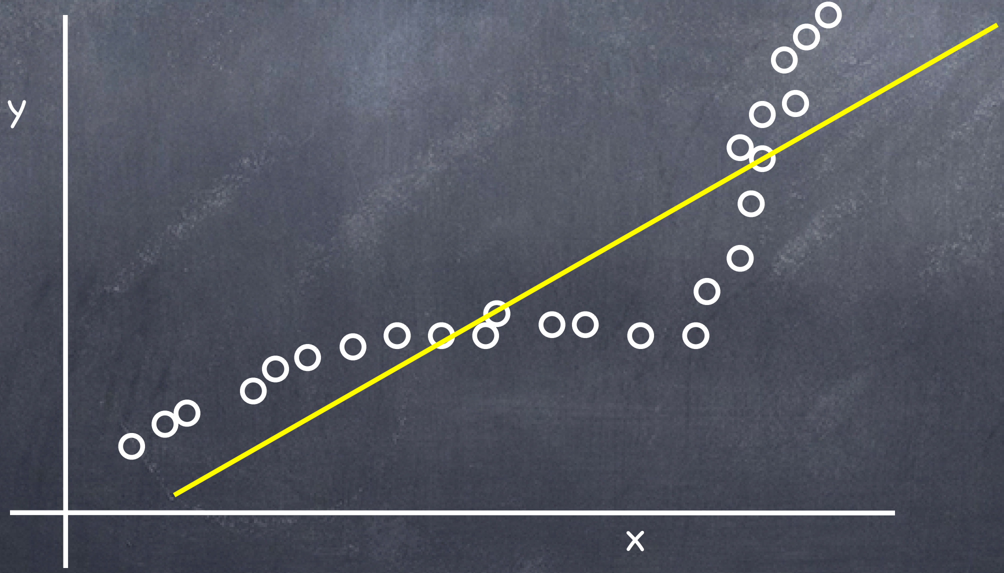
- Result from calculus, least squares achieved when:

$$a = \frac{n \sum_i x_i y_i \; - (\sum_i x_i)\,(\sum_i y_i)}{n \sum_i x_i^{\,2} - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i \; - a \sum_i x_i}{n}$$

We will use this as a subroutine
(running time O(n))

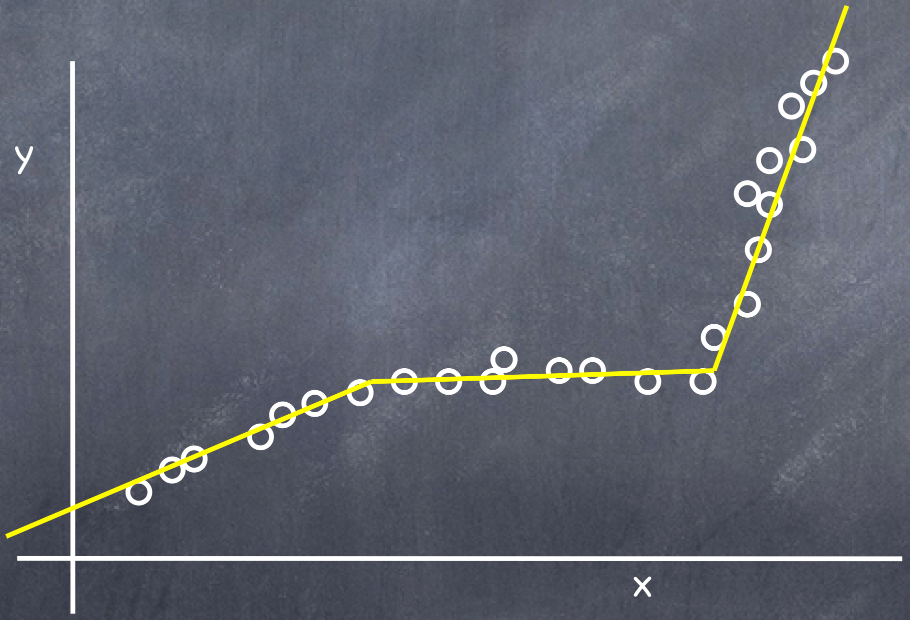
Least Squares

- Sometimes a single line does not work very well.



Segmented Least Squares

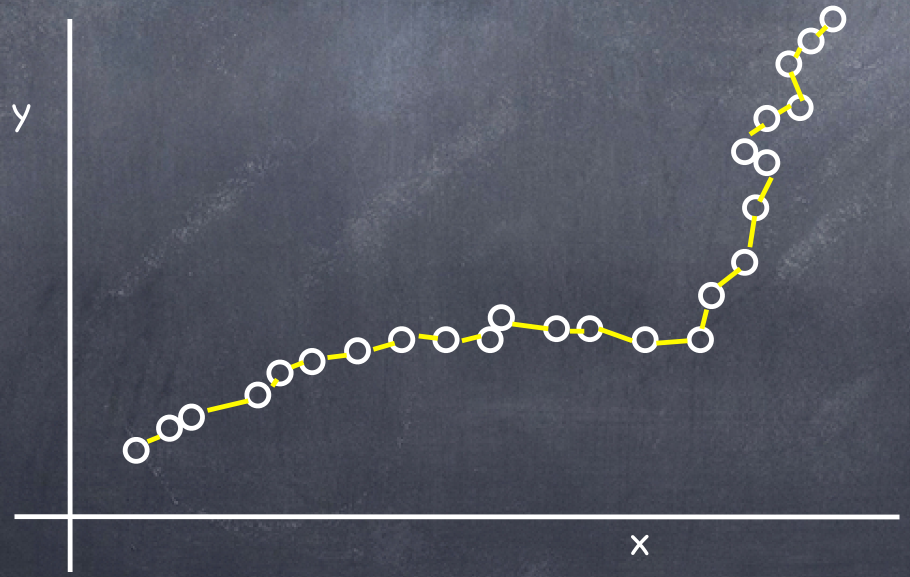
- Given n points in the plane $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_1 < x_2 < \ldots < x_n$, find a sequence of lines that fits well.



No longer have a simple solution from calculus

Segmented Least Squares

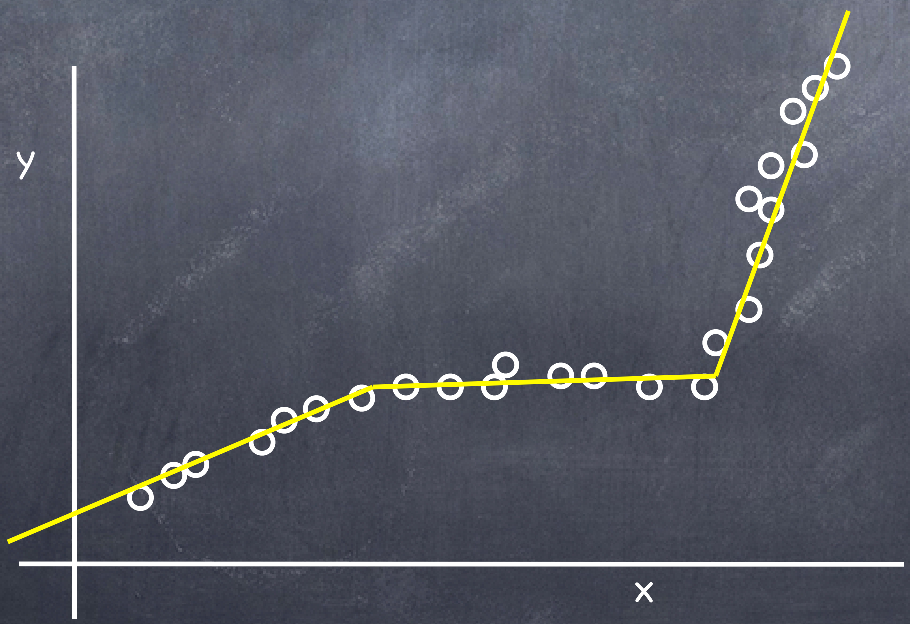
Issue: how many lines? With too many lines, you can get a prefect solution, but there may be a much simpler explanation (e.g., two lines)



Segmented Least Squares

Idea: Find a sequence to minimize some combination of:

- the total error from each segment
- the number of lines



Segmented Least Squares

- Finish problem formulation and develop recurrence on board

Segmented Least Squares: Algorithm

```
Segmented-Least-Squares() {
   for all pairs i < j
      compute the least square error e_ij for
      the segment p_i,..., p_j
   end

   M[0] = 0
   for j = 1 to n
      M[j] = min_{1 ≤ i ≤ j} (e_ij + C + M[i-1])
   end

   return M[n]
}
```

Cost

First loop: $O(n^3)$

Second loop: $O(n^2)$

Total = $O(n^3)$

Segmented Least Squares: A Second Example

Weighted Interval scheduling

- n subproblems
- Two cases: include j or don't include j

Segmented Least Squares

- n subproblems
- Up to n cases (select starting point p_i of final segment, $i \leq j$)