# Aliens DP Solution - Recap

$DP_{i,j} :=$ minimum cost to cover first $i$ points with at most $j$ photos.

**Base Cases**

$$DP_{0,j} = 0,$$

$$DP_{i,1} = (r_{i-1} - r_0 + 1)^2$$

**Transition**

$$DP_{i,j} = \min_{x<i}\left[DP_{x,j-1} + (r_{i-1} - l_x + 1)^2 - (\max(0, r_{x-1} - l_x + 1))^2\right]$$

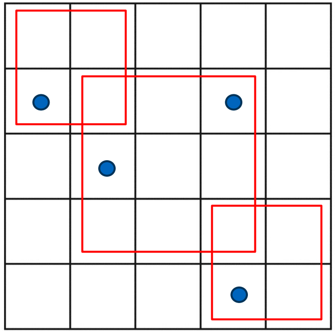
**Complexity**

$O(kn^2)$

# Aliens DP Solution - Recap

$DP_{i,j} :=$ minimum cost to cover first $i$ points with at most $j$ photos.

**Segments (i=3):**
[0,1], [1,3], [3,4]



We produce the following DP table:

| j\i | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 4 | 16 | 25 |
| 2 | 4 | 12 | 19 |
| 3 | 4 | 12 | 15 |

Complexity: $O(kn^2)$

# Knuth’s optimization

- Knuth’s optimization can bring this down to $O(n^2)$ total (or $O(nk)$ in general).
- Our transition function: $DP_{i,j} = \min_{x<i}\left[DP_{x,\,j-1} + (r_{i-1} - l_x + 1)^2 - (\max(0, r_{x-1} - l_x + 1))^2\right]$
  can be rewritten as follows:

$$f_{i,j} = \min_{0 \le x \le j} f_{i-1,x} + cost_{x,i}$$

$$\mathrm{cost}(x,i) = (r_{i-1} - l_x + 1)^2 - (\max(0, r_{x-1} - l_x + 1))^2$$

Knuth optimization applies when the cost function satisfies two properties (for a ≤ b ≤ c ≤ d):

- Monotonicity on lattice of intervals (**MLI**): cost(b,c) ≤ cost(a,d)
- Quadrangle inequality (**QI**): $\mathrm{cost}(a,c) + \mathrm{cost}(b,d) \le \mathrm{cost}(b,c) + \mathrm{cost}(a,d)\ \forall\ a \le b \le c \le d$

These ensure a convex-like structure on the DP surface.

Both of MLI and QI are easy to prove for the Aliens problem.

# Knuth’s optimization

Let's define another array in addition to the dp array - opt[N][N]. Define opt[i][j] as the maximum (or minimum) value of k for which dp[i][j] is minimized in the dp transition.

$$\left[\mathrm{opt}[i][j] = \arg\min_{i \le k < j} \left(DP[i][k] + DP[k+1][j]\right)\right]$$

The key to Knuth's optimization, and several other optimizations in DP is the following inequality:

$$\left[\boxed{\mathrm{opt}[i][j-1] \le \mathrm{opt}[i][j] \le \mathrm{opt}[i+1][j]}\right]$$

So instead of checking all t in [0, i-1], we only check between opt[i][j-1] and opt[i+1][j].
This shrinks the search range dynamically.

# Why $O(n^2)$?

When we move i forward, we keep a pointer **x_opt** that tracks the best split.

- Every time we compare x_opt and x_opt+1,
    - we move x_opt forward only when the cost gets smaller.
    - Because the cost is convex in x, once it increases, it will keep increasing.

x_opt only moves forward at **most n−1** times in total.

So across the whole layer:

- Each i does one comparison to check the next,
- Each forward move adds one comparison,
  giving ≤ 2n comparisons total — hence O(n) per layer.

# Knuth Summary

| **Property** | **Requirement** |
| --- | --- |
| Applies when | cost satisfies QI and MLI |
| Guarantees | opt[i][j-1] ≤ opt[i][j] ≤ opt[i+1][j] |
| Per layer | $O(n)$ |
| Total | $O(nk) \rightarrow O(n^2)$ if $k = n$ |

# Divide and Conquer

- Sometimes we can't prove the full quadrangle inequality.
- The cost function might not be enough for Knuth, but we can still show that **opt[i] is monotone.**
- In that case, we can use Divide and Conquer optimization.

```
fn solve(j, i_lo, i_hi, x_lo, x_hi)
  if i_lo > i_hi return
  let i_mid = (i_hi + i_lo) / 2
  // find DP[i_mid, j], opt[i_mid, j] like before
  solve(j, i_lo, i_mid - 1, x_lo, opt[i_mid, j])
  solve(j, i_mid + 1, i_hi, opt[i_mid, j], x_hi)
```

# Divide and Conquer

- We recursively compute midpoints.
- We use the known opt boundaries to limit our search.
- Each level does O(n) total work.
- The recursion has O(log n) depth → O(n log n) per layer.

```
fn solve(j, i_lo, i_hi, x_lo, x_hi)
  if i_lo > i_hi return
  let i_mid = (i_hi + i_lo) / 2
  // find DP[i_mid, j], opt[i_mid, j] like before
  solve(j, i_lo, i_mid - 1, x_lo, opt[i_mid, j])
  solve(j, i_mid + 1, i_hi, opt[i_mid, j], x_hi)
```

# Summary

| Property | Requirement | Per Layer | Requirement |
|---|---|---|---|
| Naive | None | $O(n^2)$ | $O(n^3)$ |
| Divide & Conquer | Monotone opt | $O(n \log n)$ | $O(n^2 \log n)$ |
| Knuth | Monotone opt + QI | $O(n)$ | $O(n^2)$ |