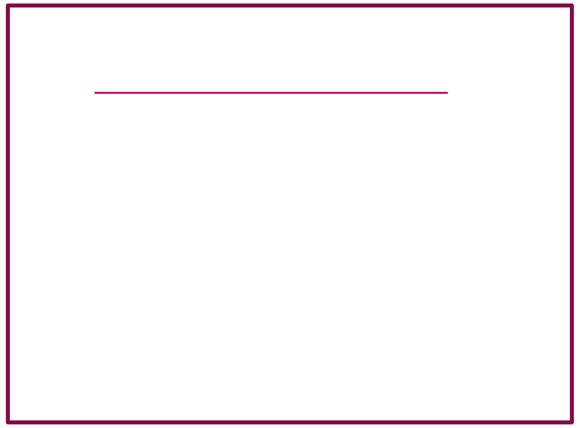
Lab8 Solution

YAO ZHAO

Lab8.A: Yan_ice loves lines

- Yan_ice once dreamed of an infinitely large plane that contained N lines. He surprisingly found that any pairs among these lines did not coincide, and any triples did not intersect at one point. He carefully counted the intersections in his dream, but when he woke up, he suddenly forgot everything.
- Please list the possible number of intersections of the N lines for him.

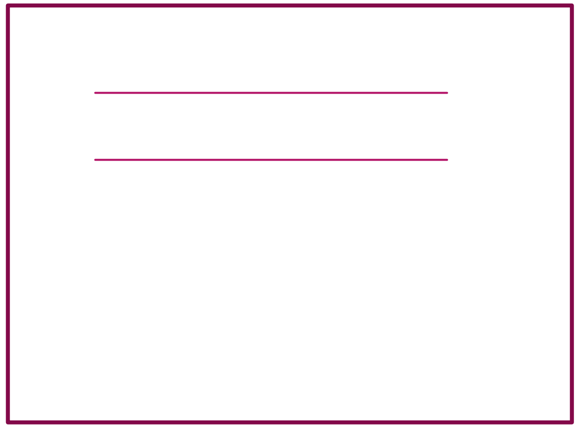
N = 1



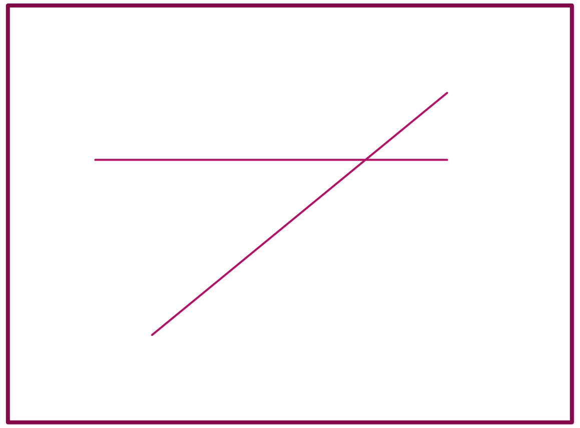
0

possible number of intersections:
0

N = 2



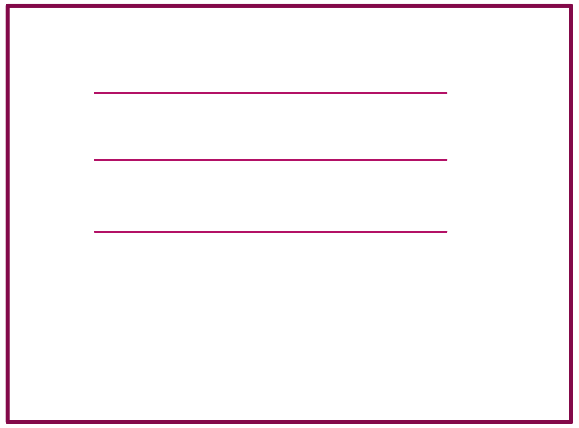
0



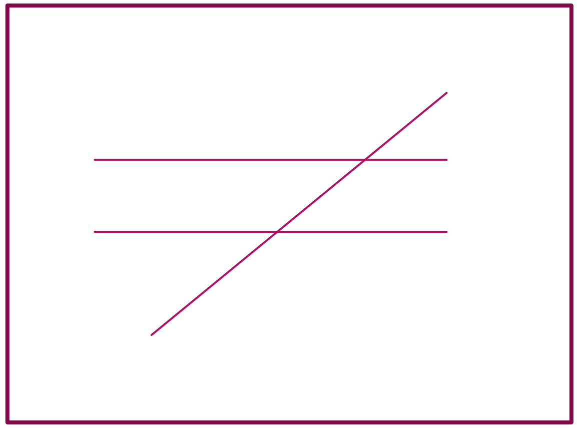
1

possible number of intersections:
2 parallel lines: 0
no parallel lines: 1

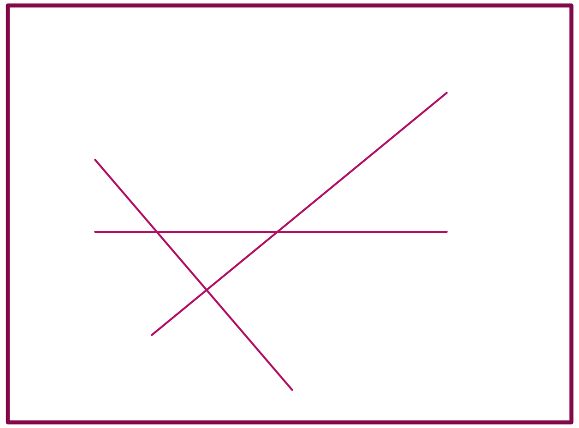
N = 3



0



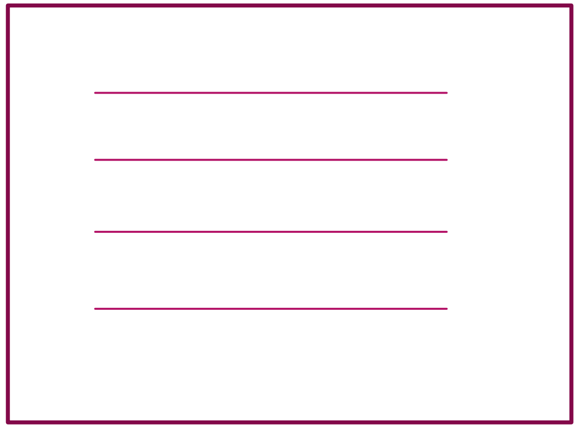
2

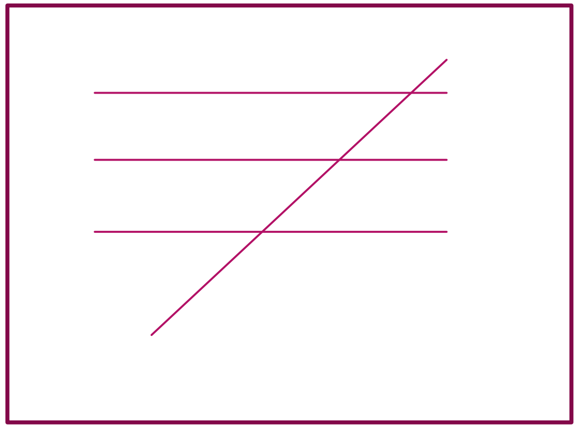


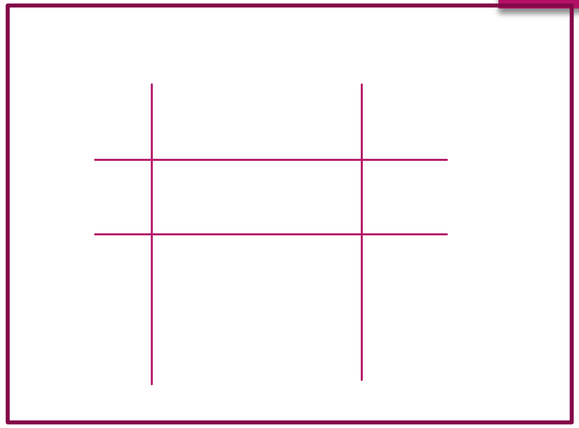
3

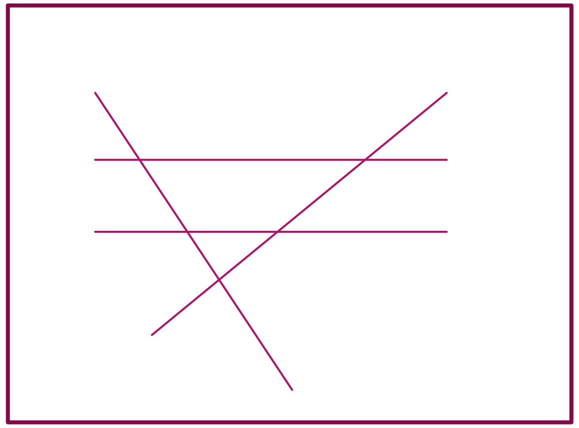
possible number of intersections:
3 parallel lines: 0
2 parallel lines: 2
no parallel lines: 3

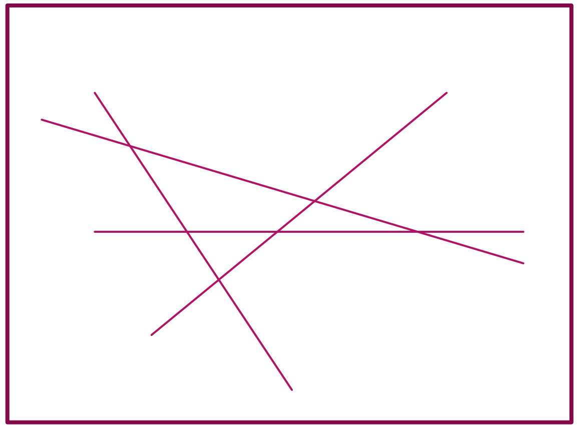
N = 4


0


3


4


5


6

possible number of intersections:

4 parallel lines:

3 parallel lines:

3*(4-3) + 0

2 parallel lines:

2*(4-2) + 0

2*(4-2) +1

no pre lines parallel to new line:

1*(4-1) + 0

1*(4-1) + 2

1*(4-1) + 3

$s_0 = \{0\}, s_1 = \{0\},$
$s_n = \varnothing \quad s_n = s_n \cup \{i * (n - i) + s_{n-i}\} \quad n \geq 2, 1 \leq i \leq n$

Lab8.B: Mr. Sorry & Satan

- Mr. Sorry and Satan are two top agents in CRA (Central Rabi Agency). One day CRA detected N bugs in a 2D plane and sent the two agents to destroy them.
- Close as the two agents are, they would accomplish missions separately for greater efficiency. Yet they must reach the coordinate of certain bug to destroy it, and they must destroy the bugs according to the given order (You know some bugs appear only when you wipe out the previous bugs). The energy consumed for each agent equals to the sum of Manhattan distance between every two adjacent coordinates he reach. Please calculate the minimum sum of energy consumed by the two agents.

Sample Input 1

3
0 1
1 0
1 1

1st way:
1:(0, 1) → (1, 0)→(1, 1)
2 + 1 =3
null:0
Total: 3+0 = 3

2nd way:
1:(0, 1) → (1, 0)
2
2: (1, 1)
Total: 2+0 =2

3rd way:
1: (0, 1) → (1, 1)
1
2: (1, 0)
Total: 1+0 =1

4th way:
1: (0, 1) → (1, 1)
2: (1, 0) →(1, 1)
1
Total: 0+1 =1

get minimum

Sample Output 1

1

Sample Input 2

4
1 0
9 8
3 2
5 9

1st way:

1:(0, 1) → (9, 8)→(3, 2)→(5, 9)
16 + 16 + 9 = 32

null:0

Total: 33+0 = 32

2nd way:

1:(0, 1)

2:(9, 8)→(3, 2)→(5, 9)
16 + 9

Total: 16 + 10 = 25

4th way:

1:(3, 2)

2:(0, 1)→(9, 8)→(5, 9)
16 + 5

Total: 16 + 10 = 21

3rd way:

1: (9, 8)

2: (0, 1) →(3, 2)→(5, 9)
4 9

Total: 4+9 =13

5th way:

1: (5, 9)

2: (0, 1) →(9, 8)→(3, 2)
16 16

Total: 16+16 =32

6th way:
1:(0, 1) $\rightarrow$(9, 8)
16
2:(3, 2)$\rightarrow$(5, 9)
9
Total: 16 + 9 = 25

7th way:
1:(0, 1) $\rightarrow$(5, 9)
13
2:(9, 8)$\rightarrow$(3, 2)
12
Total: 13 + 12 = 25

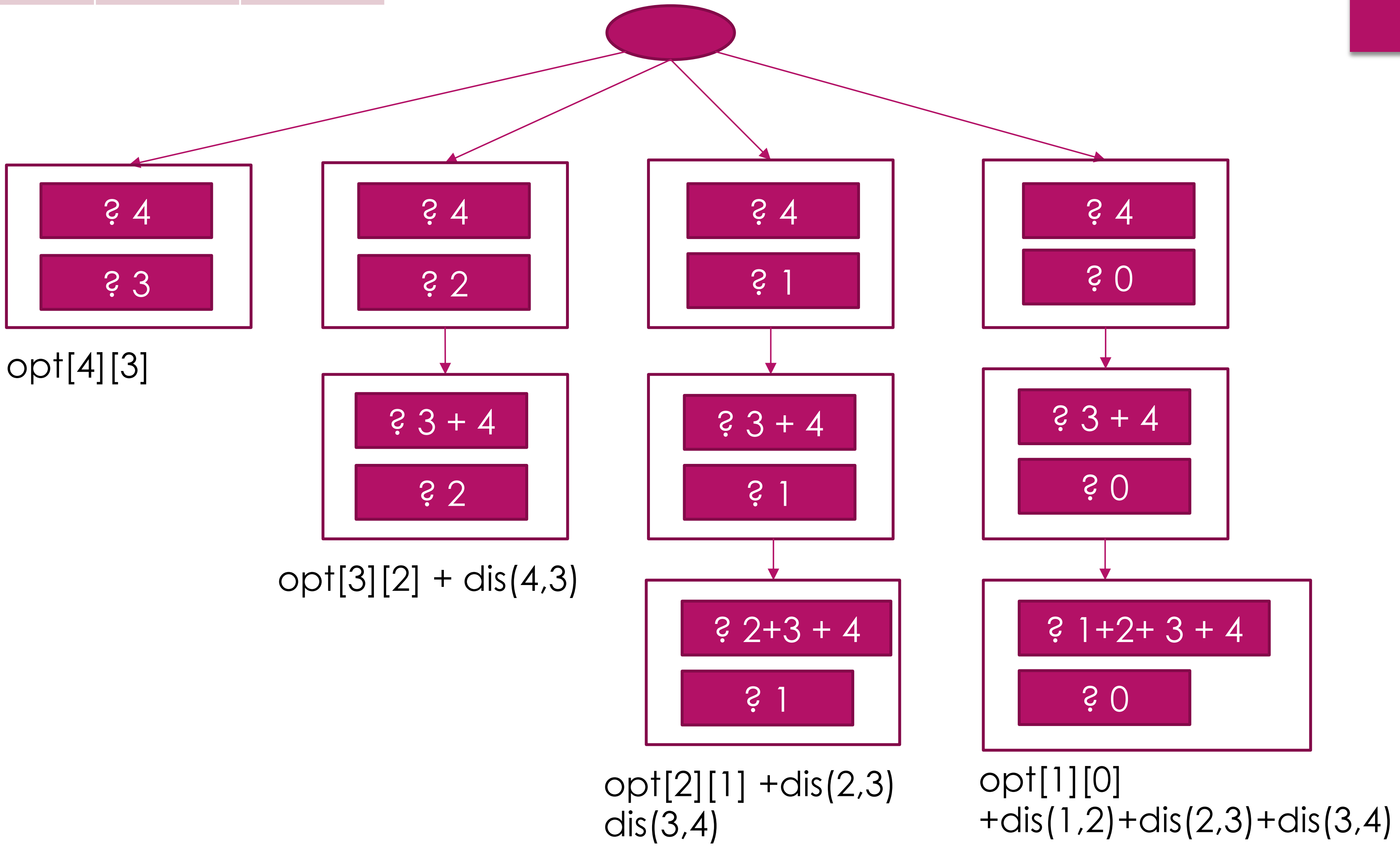
8th way:
1: (0, 1) $\rightarrow$(3, 2)
4
2: (9, 8) $\rightarrow$(5, 9)
5
Total: 4+5 =9

Sample Output 2

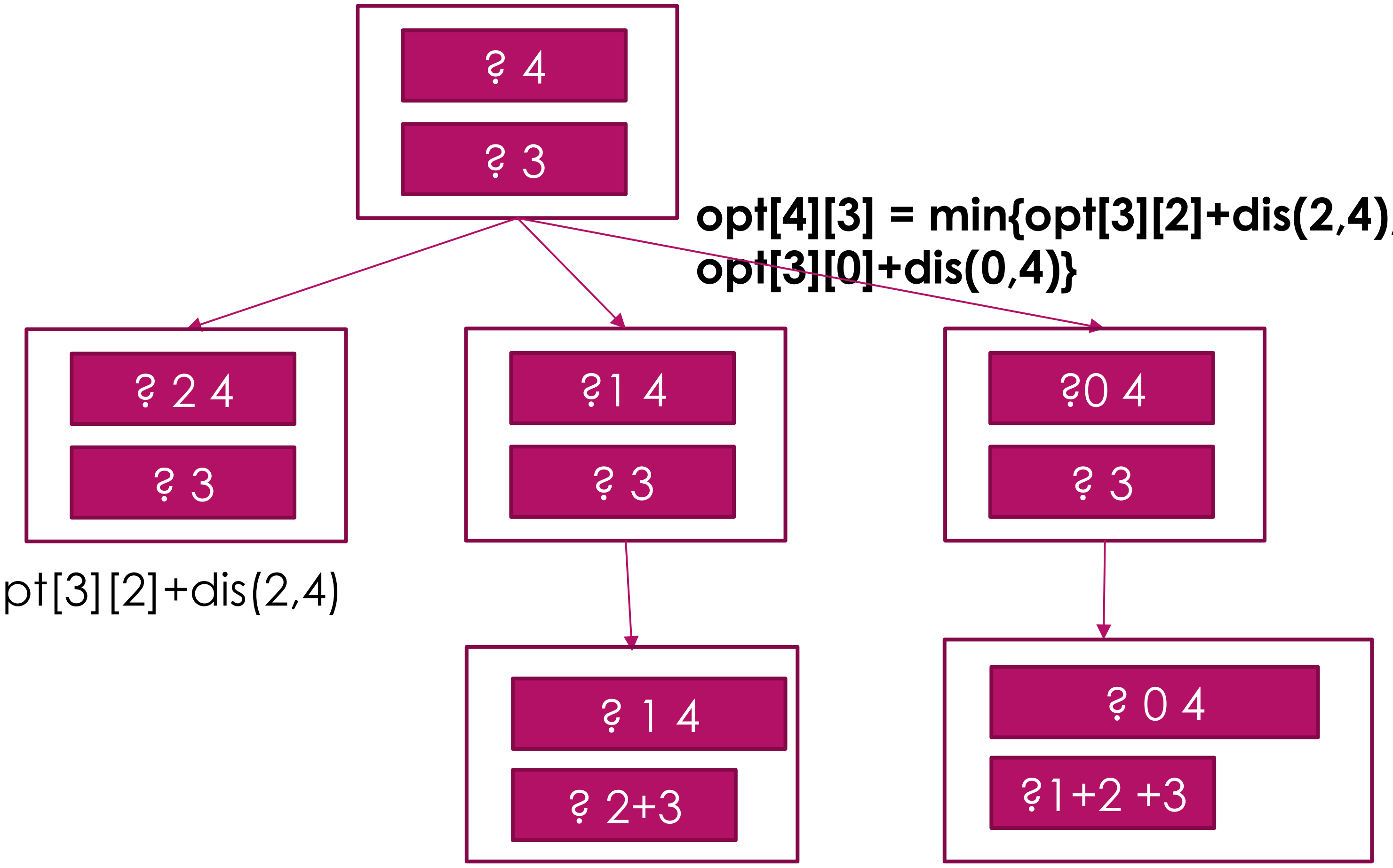
9

1	2	3	4
(1,0)	(9,8)	(3,2)	(5,9)

answer= min{opt[4][3], opt[4][2], opt[4][1], opt[4][0]}



opt[4][3] = min{opt[3][2]+dis(2,4), opt[3][1]+dis(1,4), opt[3][0]+dis(0,4)}



opt[3] [2]+dis(2,4)

opt[2] [1] +dis(2,3)   + dis(1, 4)

opt[1] [0]+dis(1,2)+dis(2,3)   + dis(0,4)

$$\textbf{opt[i][i-1] = min\{opt[k][k-1]+}\underbrace{\textbf{dis(k,k+1)+dis(k+1,k+2),...dis(i-2,i-1)}}_{\text{Continuous interval}}\textbf{ + dis(k-1,i)\} (k = 1,...i-1)}$$

Continuous interval → prefix sum

$$opt[i][i-1] = \min\{opt[k][k-1]+\underbrace{dis(k,k+1)+dis(k+1,k+2),\ldots dis(i-2,i-1)}_{\text{Continuous interval} \;\rightarrow\; \text{prefix sum}} + dis(k-1,i)\} \quad (k = 1,\ldots i-1)$$

$$\Big\downarrow \quad \begin{array}{l} \text{let } opt'[i] = opt[i][i-1] \\ \text{let } s[i-1] = dis(1,2)+dis(2,3)\ldots+dis(i-2,\ i-1) \\ \text{let } s[k] = dis(1,2)+dis(2,3)\ldots+dis(k-1,\ k) \end{array}$$

$$opt'[i] = \min\{opt'[k]+ (s[i-1]-s[k])+ dis(k-1,i)\} \quad (k = 1,\ldots i-1)$$

$$\Big\downarrow$$

$$opt'[i] = \min\{opt'[k] -s[k]+ dis(k-1,i)\} + s[i-1] \quad (k = 1,\ldots i-1)$$

answer= $\min\{opt'[n], opt'[n-1]+s[n]-s[n-1],\ldots opt'[i]+s[n]-s[i], opt'[1]+s[n]-s[1]\}$

opt'[i] = min{opt'[k] -s[k]+ dis(k-1,i)} + s[i-1] (k = 1,...i-1)

only k   only k   contain i

dis(k-1,i) = $|x_i - x_{k-1}| + |y_i - y_{k-1}|$

if $x_i \ge x_{k-1},\ y_i \ge y_{k-1}$: dis(k-1,i) = $x_i - x_{k-1} + y_i - y_{k-1} = - x_{k-1} - y_{k-1} + x_i + y_i$

opt'[i] = min{opt'[k] -s[k] $- x_{k-1} - y_{k-1}$} + s[i-1] $+ x_i + y_i$ (k = 1,...i-1)

if $x_i < x_{k-1},\ y_i \ge y_{k-1}$: dis(k-1,i) = $x_{k-1} - x_i + y_i - y_{k-1} = +x_{k-1} - y_{k-1} - x_i + y_i$

opt'[i] = min{opt'[k] -s[k] $+x_{k-1} - y_{k-1}$} + s[i-1] $- x_i + y_i$ (k = 1,...i-1)

if $x_i \ge x_{k-1},\ y_i < y_{k-1}$: dis(k-1,i) =$x_i - x_{k-1} + y_{k-1} - y_i = -x_{k-1} + y_{k-1} + x_i - y_i$

opt'[i] = min{opt'[k] -s[k] $-x_{k-1} + y_{k-1}$} + s[i-1] $+ x_i - y_i$ (k = 1,...i-1)

if $x_i < x_{k-1},\ y_i < y_{k-1}$: dis(k-1,i) = $x_{k-1} - x_i + y_{k-1} - y_i = +x_{k-1} + y_{k-1} - x_i - y_i$

opt'[i] = min{opt'[k] -s[k] $+x_{k-1} + y_{k-1}$} + s[i-1] $- x_i - y_i$ (k = 1,...i-1)

The formula for min only contains k

Why do this conversion?

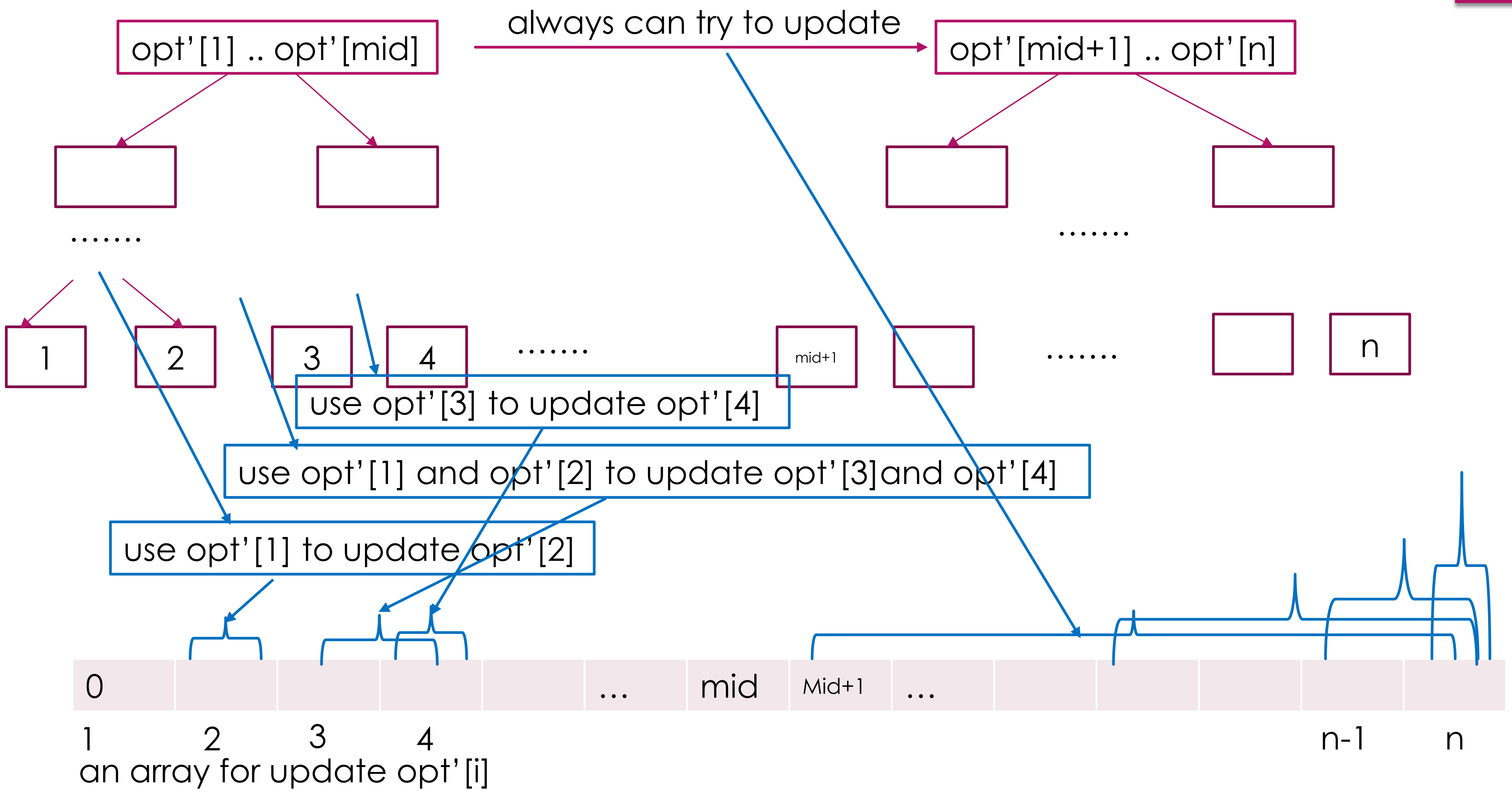
This means that each calculation for opt'[i] can reuse the results of the previous i-1

for example:

$$\begin{tikzcd}
\text{opt'}[1] \arrow[r] \arrow[drr] \arrow[ddrrr] \arrow[dddrrrr] & \text{opt'}[2] \arrow[dr] \arrow[ddrr] \arrow[dddrrr] & & & \\
& & \text{opt'}[3] \arrow[dr] \arrow[ddrr] & & \\
& & & \text{opt'}[4] \arrow[dr] & \\
& & & & \text{opt'}[5] \\
& & & & \ldots
\end{tikzcd}$$

$O(n^2)$

try divide the problem



opt'[1] .. opt'[mid]
always can try to update
opt'[mid+1] .. opt'[n]
....
....
1
2
3
4
....
mid+1
....
n
use opt'[3] to update opt'[4]
use opt'[1] and opt'[2] to update opt'[3]and opt'[4]
use opt'[1] to update opt'[2]
0
...
mid
Mid+1
...
1
2
3
4
n-1
n
an array for update opt'[i]

```
divide-and-conquer(l, r){
    mid = $\left\lfloor \frac{l+r}{2} \right\rfloor$;
    divide-and-conquer(l, mid);
    updateRight(l, r);
    divide-and-conquer(mid+1, r);
}
```

If the time complexity of updateRight(l, r) can be optimized to $O(nlogn)$, then the total time complexity can be optimized to $O(n(logn)^2)$

updateRight(l, r);
already: get **opt'[l] .. opt'[mid]**
to do: try to update **opt'[mid+1] .. opt'[r]**

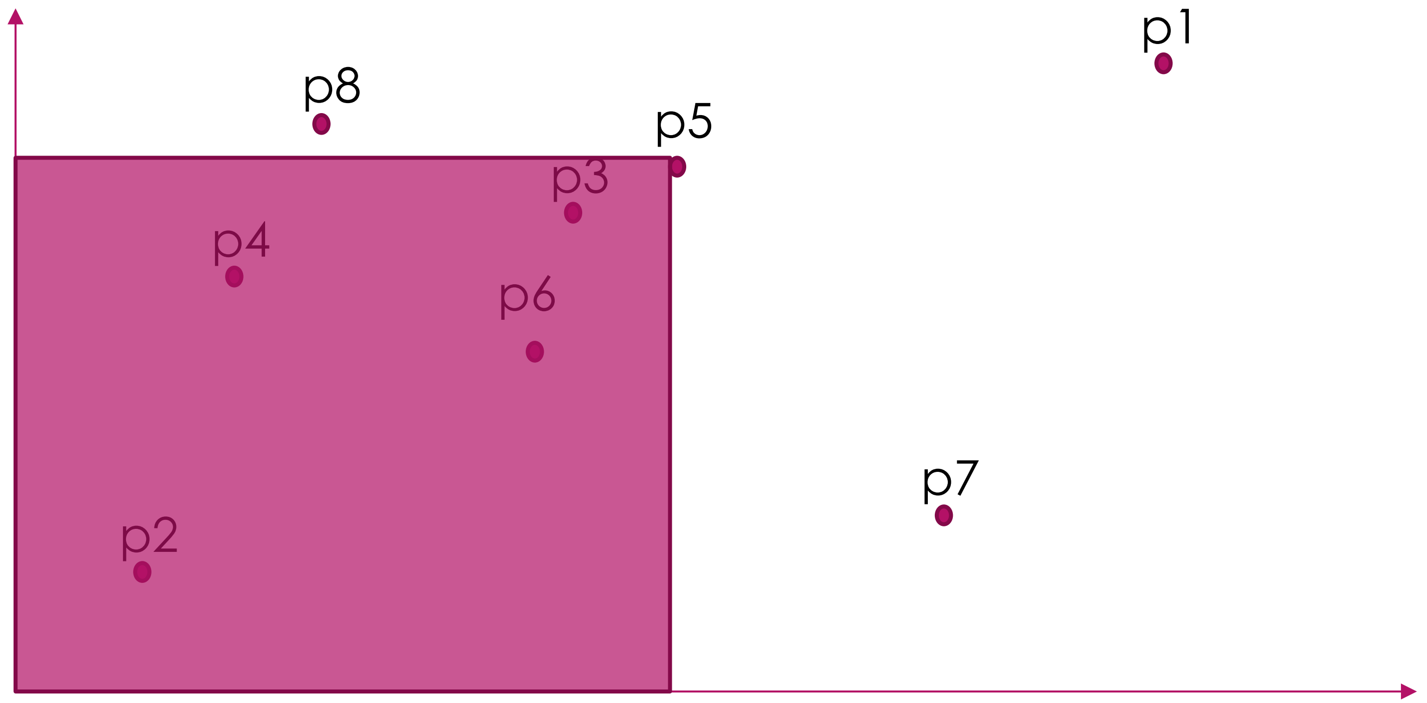
update case 1:

if $x_i \geq x_{k-1},\ y_i \geq y_{k-1}$: **dis(k-1,i)** = $x_i - x_{k-1} + y_i - y_{k-1} = -\ x_{k-1} -\ y_{k-1} + x_i + y_i$

opt'[i] can be updated by min{opt'[k] -s[k] $-\ x_{k-1} -\ y_{k-1}$**} + s[i-1]** $+ x_i + y_i$ **(k = l,…mid i = mid+1,…r)**

let l=1, r=8

case1:
opt'[5] can be update by **opt'[2] opt'[3] and opt'[4]**



updateRight(l, r);
already: get **opt'[l] .. opt'[mid]**
to do: try to update **opt'[mid+1] .. opt'[r]**

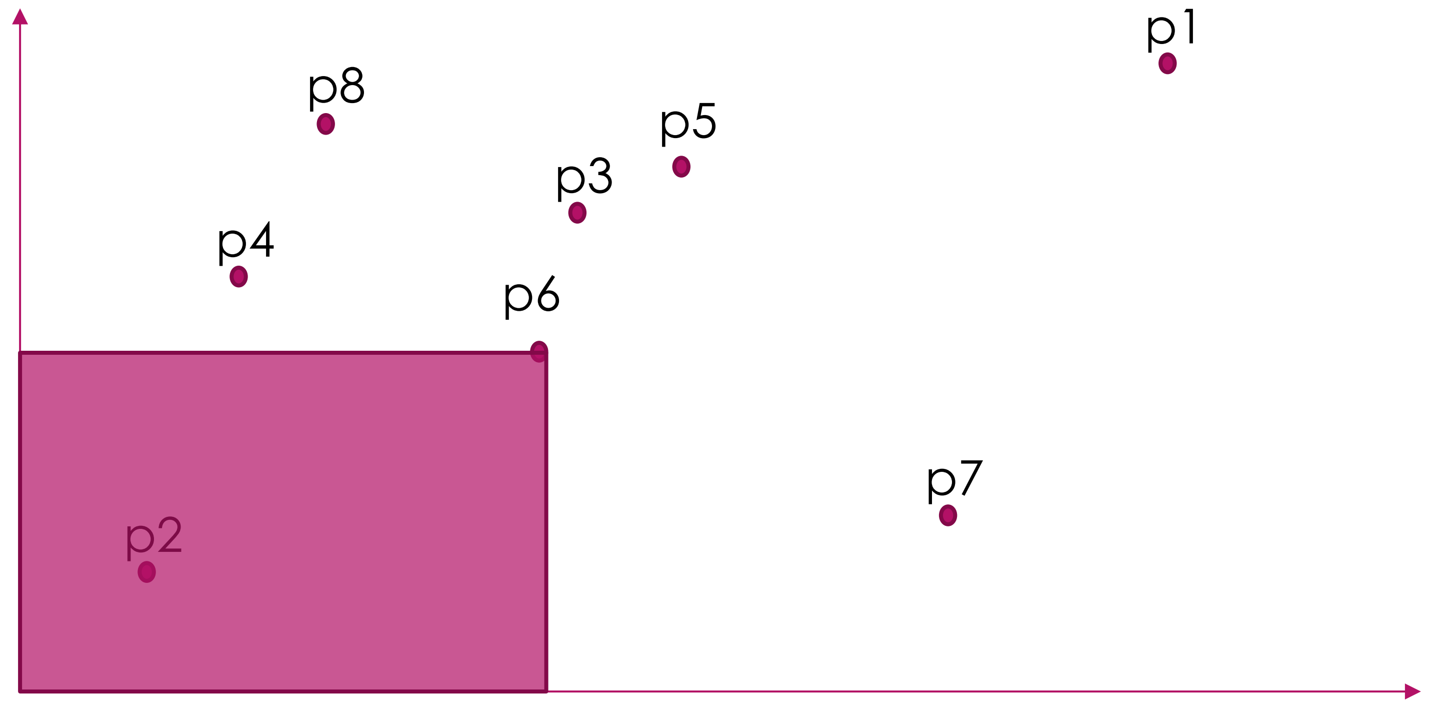
update case 1:

if $x_i \geq x_{k-1},\ y_i \geq y_{k-1}$: **dis(k-1,i)** = $x_i - x_{k-1} + y_i - y_{k-1} = - x_{k-1} - y_{k-1} + x_i + y_i$

opt'[i] can be updated by min{opt'[k] -s[k] $- x_{k-1} - y_{k-1}$**} + s[i-1]** $+ x_i + y_i$ **(k = l,…mid i = mid+1,…r)**

let l=1, r=8

case1:
opt'[6] can be update by **opt'[2]**



updateRight(l, r);
already: get **opt'[l] .. opt'[mid]**
to do: try to update **opt'[mid+1] .. opt'[r]**

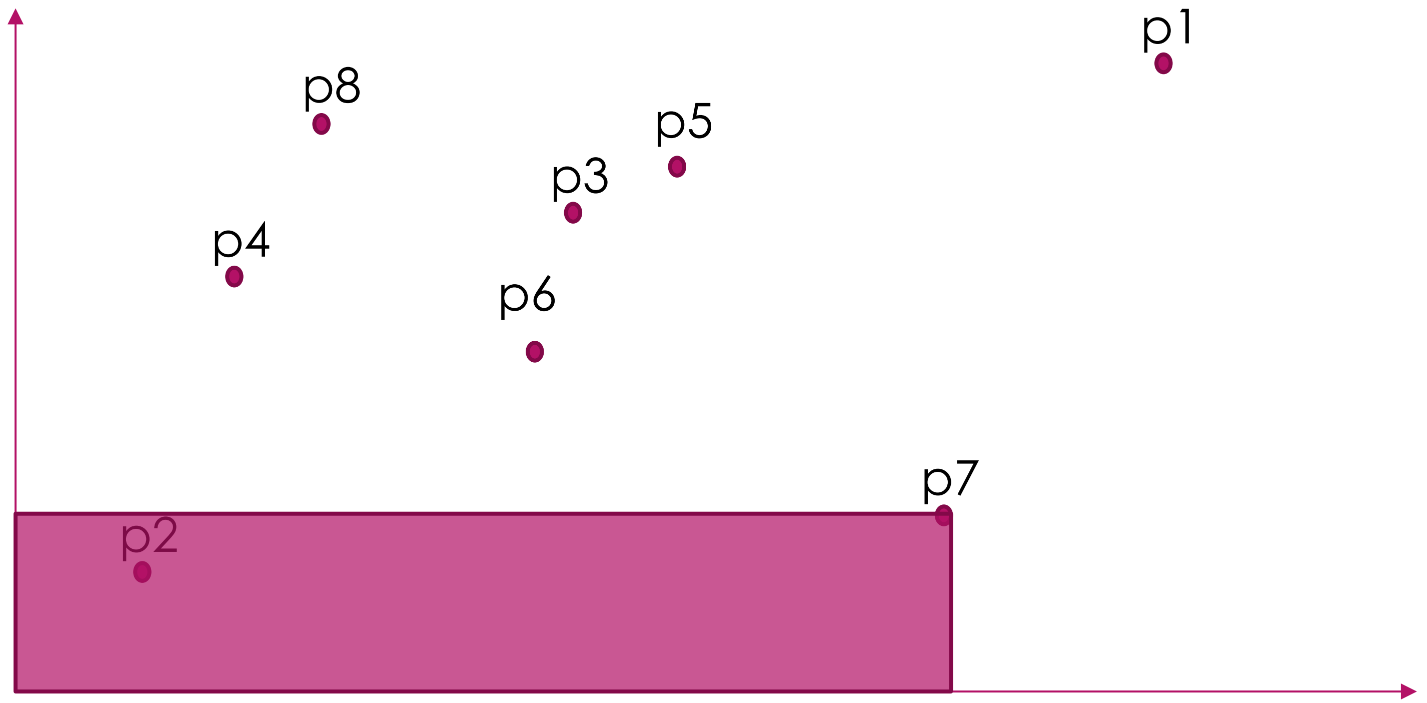
update case 1:

if $x_i \geq x_{k-1},\ y_i \geq y_{k-1}$: **dis(k-1,i)** = $x_i - x_{k-1} + y_i - y_{k-1} = - x_{k-1} - y_{k-1} + x_i + y_i$

opt'[i] can be updated by min{opt'[k] -s[k] $- x_{k-1} - y_{k-1}$**} + s[i-1]** $+ x_i + y_i$ **(k = l,…mid i = mid+1,…r)**

let l=1, r=8

case1:
opt'[7] can be update by **opt'[2]**



updateRight(l, r);
already: get **opt'[l] .. opt'[mid]**
to do: try to update **opt'[mid+1] .. opt'[r]**

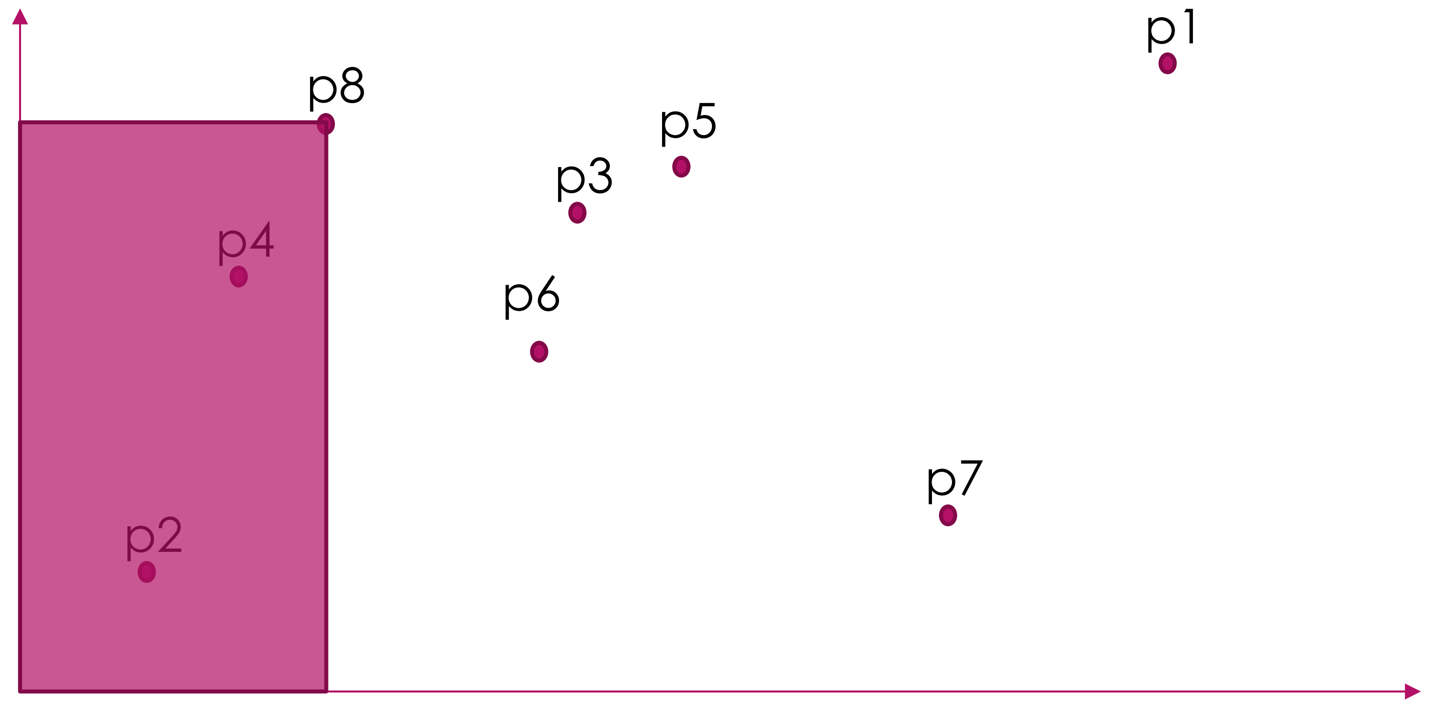
update case 1:

if $x_i \geq x_{k-1}$, $y_i \geq y_{k-1}$: **dis(k-1,i)** = $x_i - x_{k-1} + y_i - y_{k-1} = - x_{k-1} - y_{k-1} + x_i + y_i$

opt'[i] can be updated by min{opt'[k] -s[k] – $x_{k-1} - y_{k-1}$**} + s[i-1]** $+ x_i + y_i$ **(k = l,...mid i = mid+1,...r)**

let l=1, r=8

case1:
opt'[8] can be update by **opt'[2] and opt'[4]**



updateRight(l, r);
already: get **opt'[l] .. opt'[mid]**
to do: try to update **opt'[mid+1] .. opt'[r]**

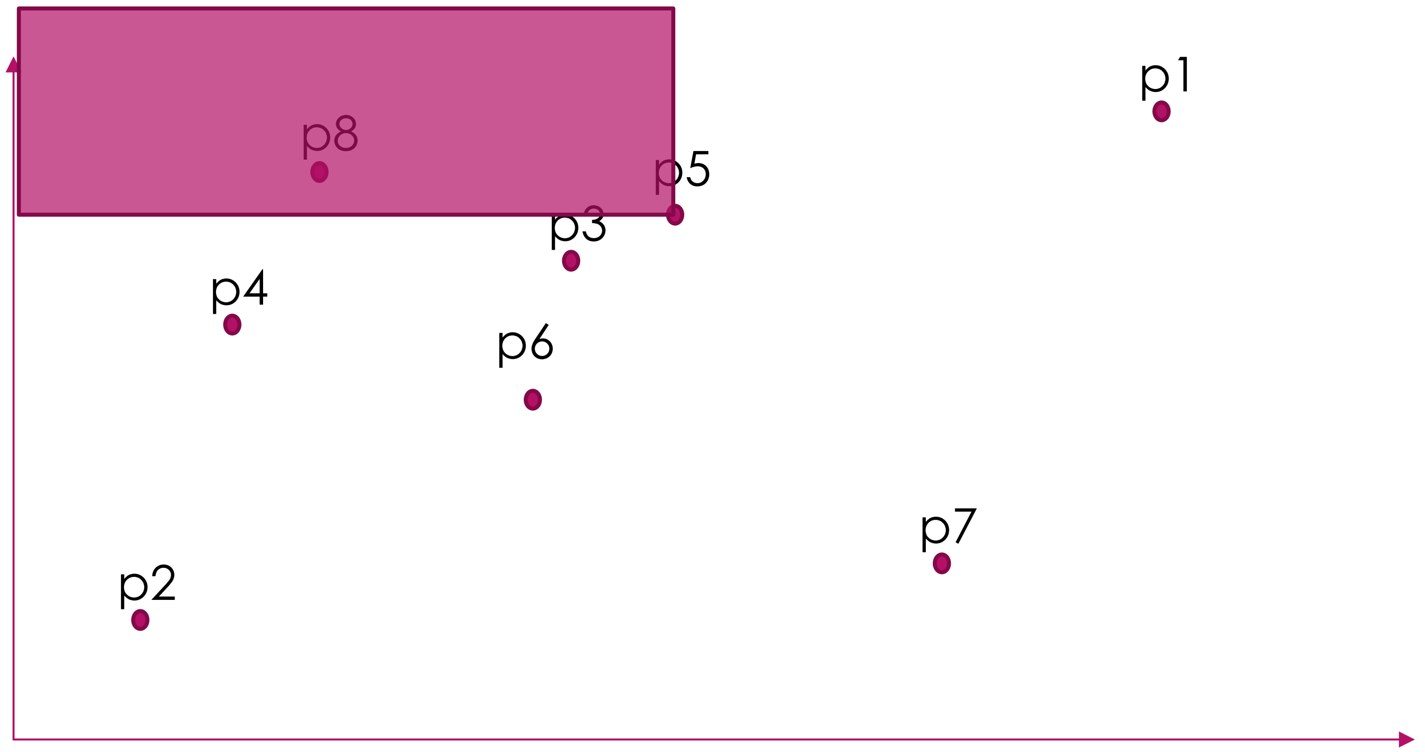
update case 2:

if $x_i < x_{k-1},\ y_i \geq y_{k-1}$: **dis(k-1,i)** = $x_{k-1} - x_i + y_i - y_{k-1} = +x_{k-1} - y_{k-1} - x_i + y_i$

opt'[i] can be updated by min{opt'[k] -s[k] $+x_{k-1} - y_{k-1}$**} + s[i-1]** $- x_i + y_i$ **(k = l,…mid i = mid+1,…r)**

let l=1, r=8

case2:
opt'[5] cannot be updated



updateRight(l, r);
already: get **opt'[l] .. opt'[mid]**
to do: try to update **opt'[mid+1] .. opt'[r]**

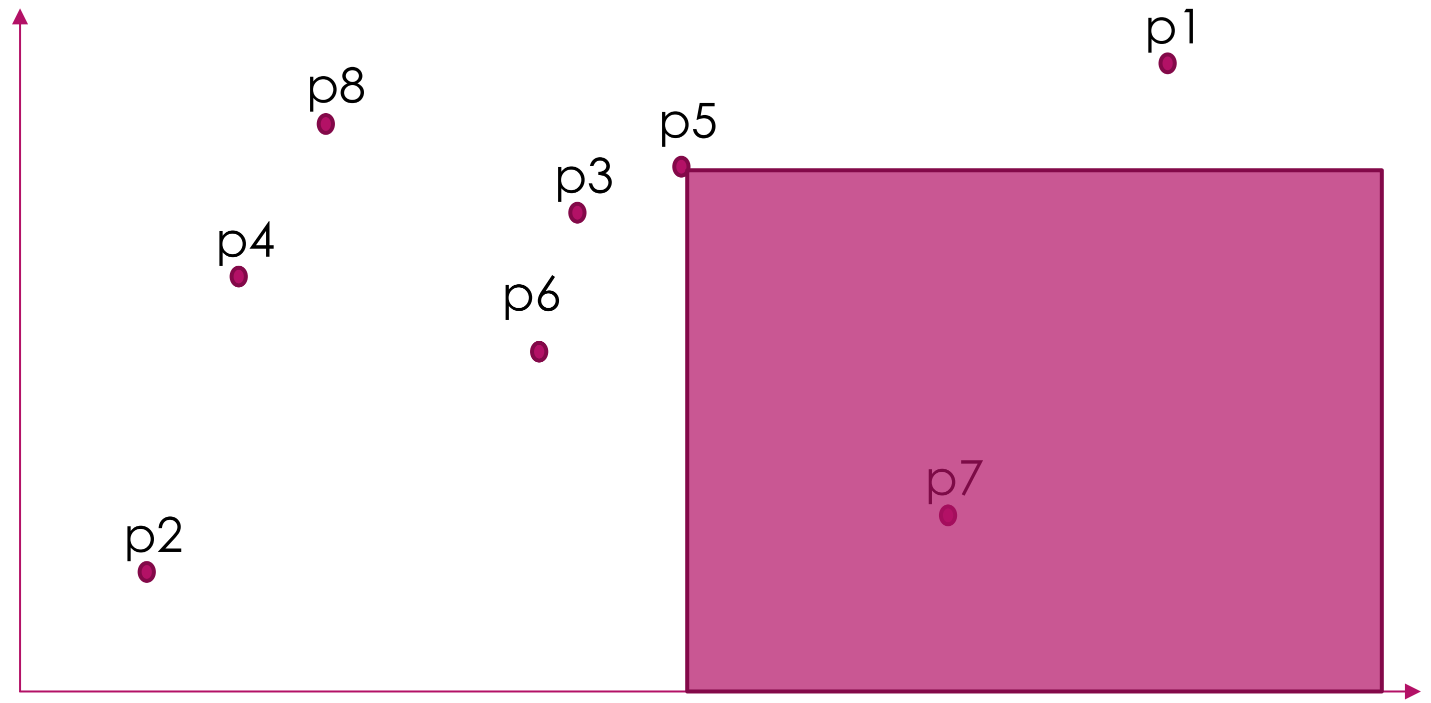
update case 3:

if $x_i \geq x_{k-1},\ y_i < y_{k-1}$: **dis(k-1,i)** $= x_i - x_{k-1} + y_{k-1} - y_i = -x_{k-1} + y_{k-1} + x_i - y_i$

opt'[i] can be updated by min{opt'[k] -s[k] $-x_{k-1} + y_{k-1}$**} + s[i-1]** $+ x_i - y_i$ **(k = l,…mid i = mid+1,…r)**

let l=1, r=8

case3:
opt'[5] cannot be updated



updateRight(l, r);
already: get **opt'[l] .. opt'[mid]**
to do: try to update **opt'[mid+1] .. opt'[r]**

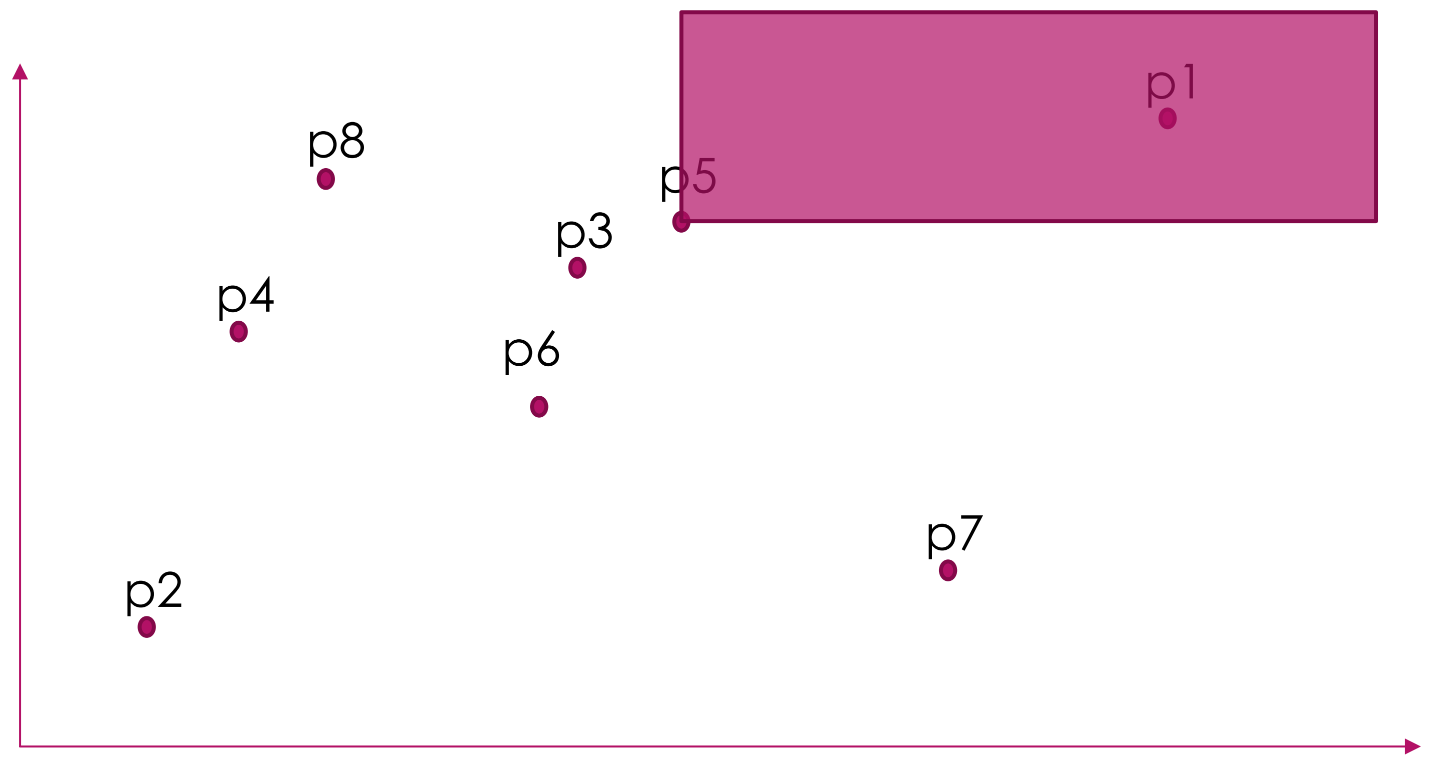
update case 4:

if $x_i \geq x_{k-1},\ y_i < y_{k-1}$: **dis(k-1,i)** $= x_i - x_{k-1} + y_{k-1} - y_i = -x_{k-1} + y_{k-1} + x_i - y_i$

opt'[i] can be updated by min{opt'[k] -s[k] $-x_{k-1} + y_{k-1}$**} + s[i-1]** $+ x_i - y_i$ **(k = l,…mid i = mid+1,…r)**

let l=1, r=8

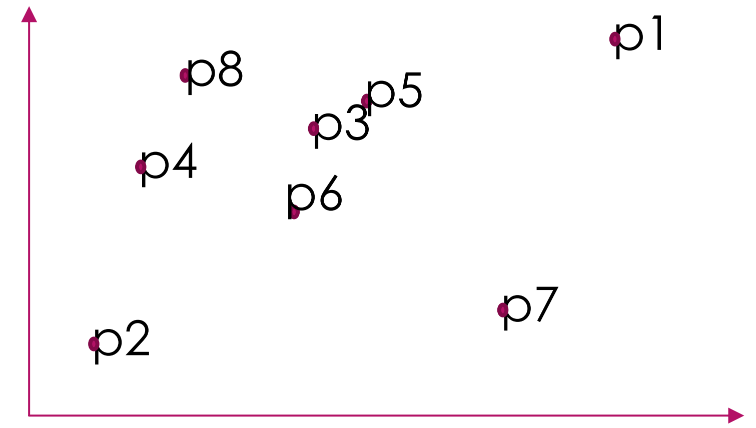
case4:
opt'[5] ca be updated by p1



For each case, find the points in [l .. mid] that satisfy the case constraints and then calculate the value to refresh the corresponding opt'[m+1] ~opt'[r].

Take case 1 for example:

sort p1~p8 by p.x



sequence no.	1	2	3	4	5	6	7	8
	p1	p2	p3	p4	p5	p6	p7	p8

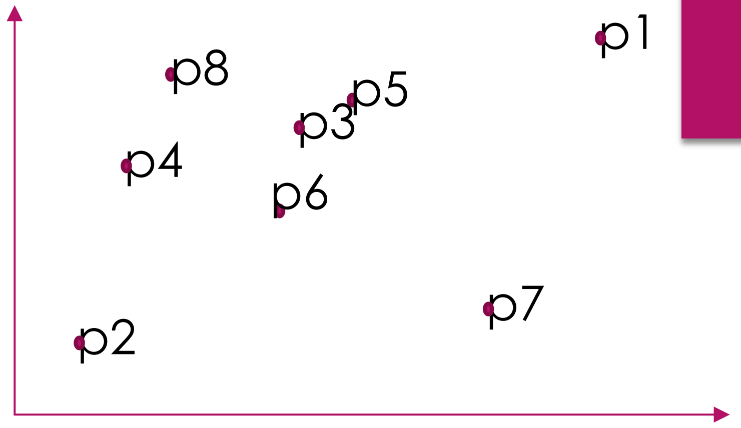
p.x rank:	1	2	3	4	5	6	7	8
	p2	p4	p8	p6	p3	p5	p7	p1

p.y rank:	1	2	3	4	5	6	7	8
	p2	p7	p6	p4	p3	p5	p8	p1

p.x rank:	1	2	3	4	5	6	7	8
sorted x:	p2	p4	p8	p6	p3	p5	p7	p1

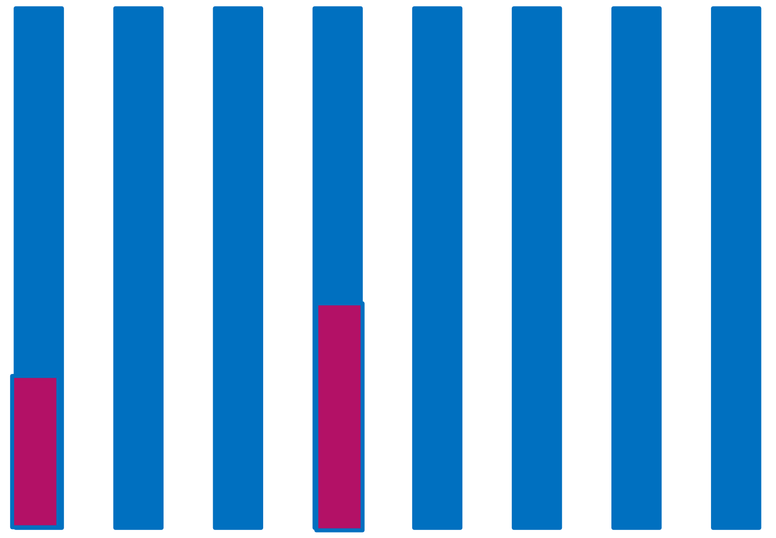
p.y rank:	1	2	3	4	5	6	7	8
sorted y:	p2	p7	p6	p4	p3	p5	p8	p1

sequence no.	1	2	3	4	5	6	7	8
p.y rank:	8	1	5	4	6	3	2	7



updateRight(l, r) case1

```
initial v[l]  … v[r] to INF
for p in sorted x from l to r:
    if p ∈ left interval
        k= p.sequenceno
        calc the value  according to the case 1 formula: **opt'[k] -s[k]** – $x_{k-1}$– $y_{k-1}$
        v[p.y rank of p] = min(value, v[p.y rank of p)
    if p ∈ right interval
        update opt'[p.sequenceno] by **min(v[l]…v[p.y rank of p-1]) + s[i-1]** + $x_i$ + $y_i$)
```



initial v[l] … v[r] to INF

	v1	v2	v3	v4	v5	v6	v7	v8
p.x rank:	1	2	3	4	5	6	7	8
sorted x:	p2	p4	p8	p6	p3	p5	p7	p1

for p in sorted x from l to r:

sequence no.	1	2	3	4	5	6	7	8
p.y rank:	8	1	5	4	6	3	2	7

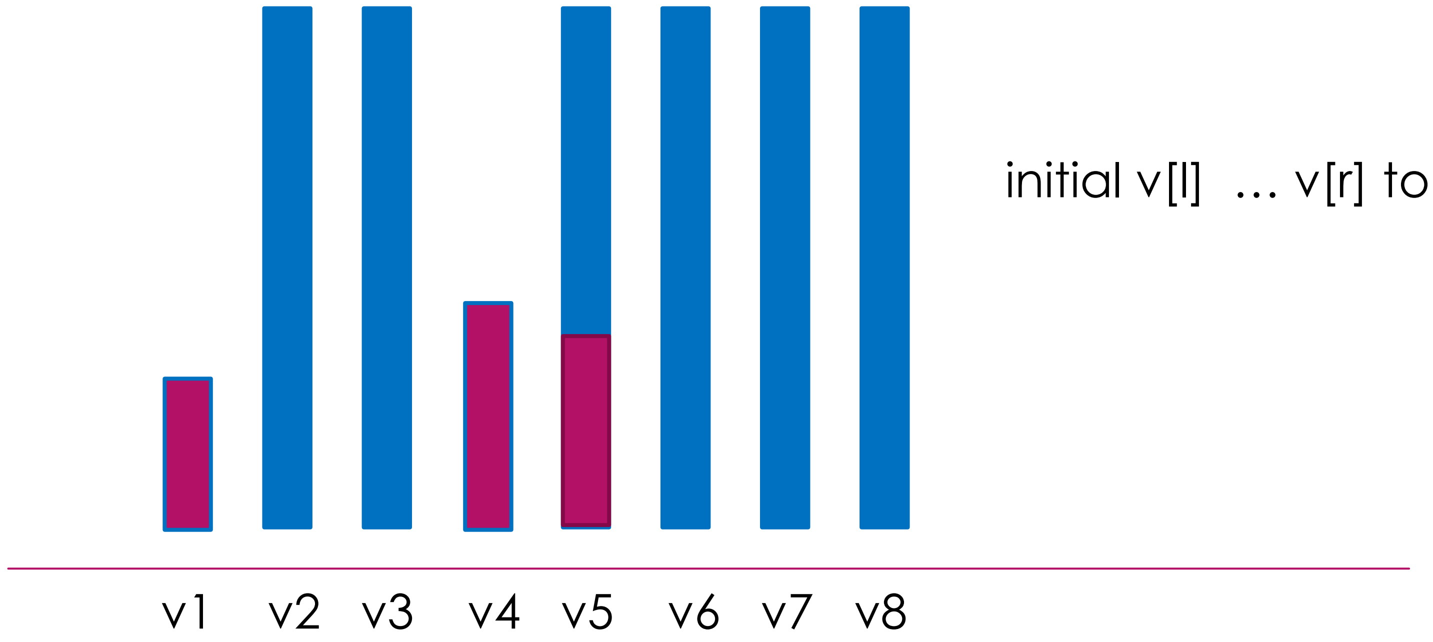
the 1st point is p2
p2 ∊ left? Yes
calc the value and update v[1]

the 2nd point is p4
p4 ∊ left? Yes
calc the value and update v[4]

the 3rd point is p8
p8 ∊ left? no
update **opt'[8]** by **min(v[1]…v[6])** **+ s[7]** $+ \boldsymbol{x_8} + \boldsymbol{y_8}$)

actual: min(v1,v4)

6 = p8.y rank - 1



initial v[l] … v[r] to INF

p.x rank:	1	2	3	4	5	6	7	8
sorted x:	p2	p4	p8	p6	p3	p5	p7	p1

for p in sorted x from l to r:

sequence no.	1	2	3	4	5	6	7	8
p.y rank:	8	1	5	4	6	3	2	7

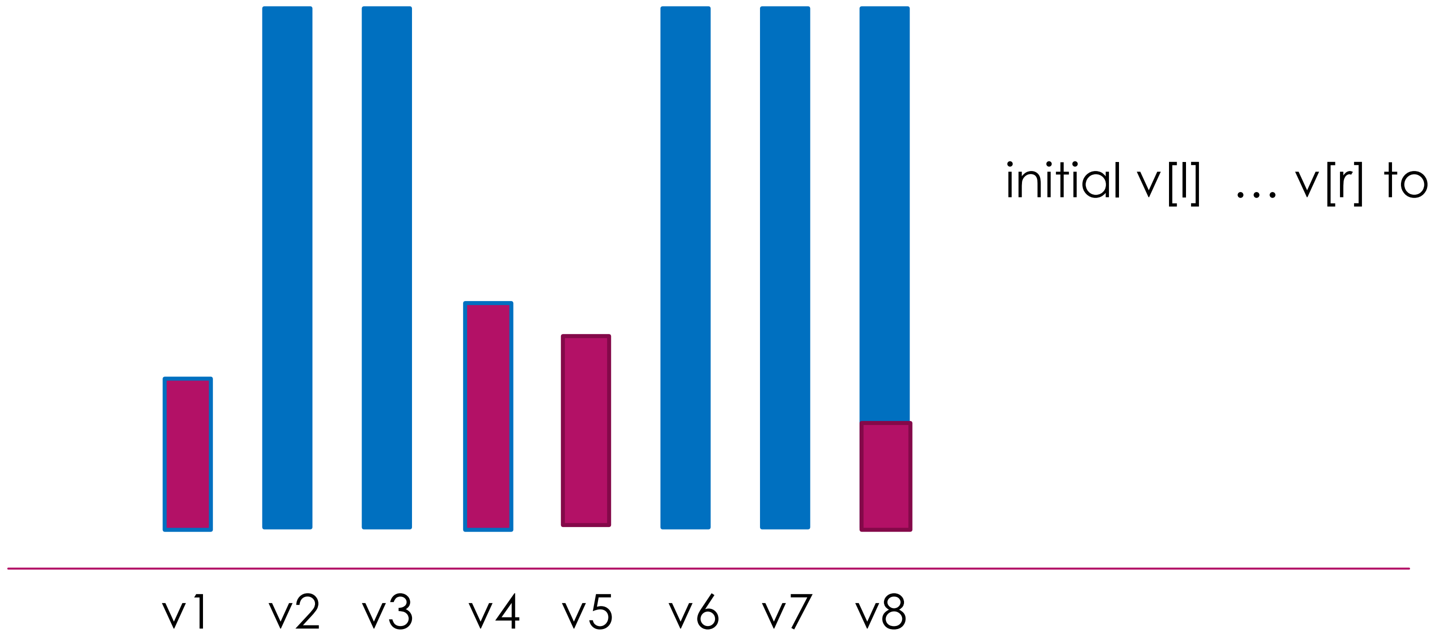
the 4th point is p6
p6 ∊ left? no
update **opt'[6]** by **min(v[1]…v[2])** **+ s[5]** $+ \boldsymbol{x_6} + \boldsymbol{y_6}$)

2 = p6.y rank - 1

the 5th point is p3
p3 ∊ left? Yes
calc the value and update v[5]

the 6th point is p5
p5 ∊ left? no
update **opt'[5]** by **min(v[1]…v[5])** **+ s[4]** $+ \boldsymbol{x_5} + \boldsymbol{y_5}$)

actual: min(v1,v4,v5) 5 = p5.y rank - 1



initial v[l] … v[r] to INF

the 7th point is p7

p7 ∈ left? no

update **opt'[7]** by **min(v[1]…v[1])** **+ s[6]** $+ \boldsymbol{x_7} + \boldsymbol{y_7}$)

1 = p7.y rank - 1

the 8th point is p1

p1∈ left? Yes

calc the value and update v[8]

p.x rank:	1	2	3	4	5	6	7	8
sorted x:	p2	p4	p8	p6	p3	p5	p7	p1

$\longrightarrow$

for p in sorted x from l to r:

sequence no.	1	2	3	4	5	6	7	8
p.y rank:	8	1	5	4	6	3	2	7

updateRight(l, r) case1

```
initial v[l]  … v[r] to INF
for p in sorted x from l to r:
    if p ∈ left interval
        k= p.sequenceno
        calc the value  according to the case 1 formula: opt'[k] -s[k] $- x_{k-1} - y_{k-1}$
        v[p.y rank of p] = min(value, v[p.y rank of p)
    if p ∈ right interval
        update opt'[p.sequenceno] by min(v[l]…v[p.y rank of p-1]) + s[i-1] $+ x_i + y_i$)
```

If set v[] and query min (v[l]…v[p.y rank of p-1]) use brute force way, the time complexity is still $O(n^2)$

Binary Indexed Tree (树状数组，简称：BIT) can be used for efficiency：
https://oi-wiki.org/ds/fenwick/

updateRight(l, r) case2, case3 and case4, You can derive from the above analysis by yourself.