Tema 3: Divide y Vencerás

El problema del máximo subarreglo

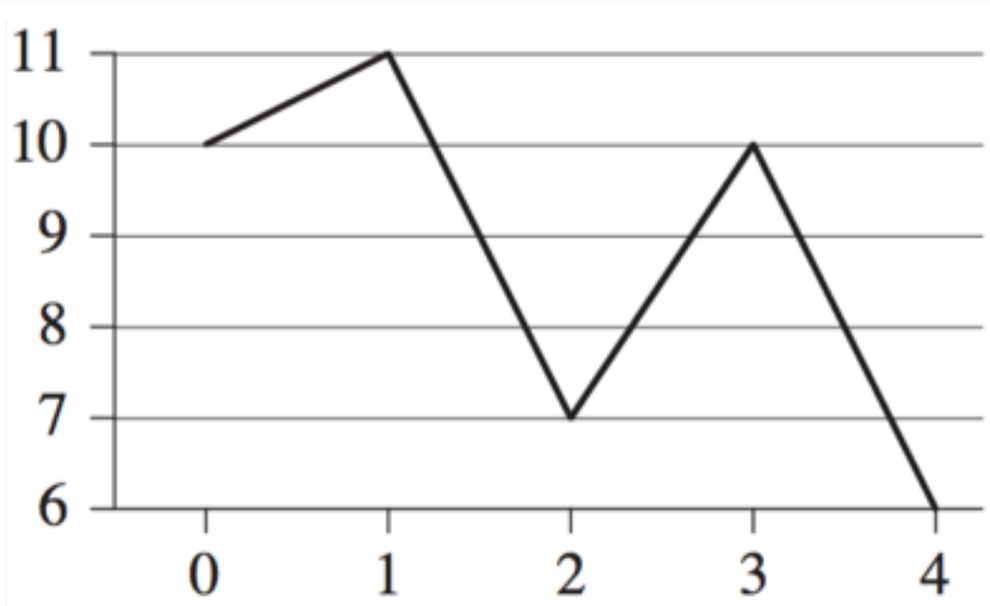
- Tenemos la oportunidad de invertir en la bolsa

- Sólo puedes comprar acciones de una compañía una vez y posteriormente vender una vez.

- Sin embargo, supongamos que podemos ver el comportamiento a futuro de las acciones.

- Una frase famosa dice: "compra barato, vende caro", lo cual nos orillaría a buscar el menor precio de las acciones y ese día comprar, luego buscar el mayor precio y ese día vender, sin embargo hay situaciones en que esto no funciona.

- De funcionar siempre, bastaría con buscar el menor precio y luego el mayor y comprar en uno y vender en el otro. Sin embargo este es un CONTRA-EJEMPLO.



Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	−4	3	−4

- En esta gráfica, la máxima ganancia sería de $3 y se obtendría al comprar el día 2 y vender el día 3.
- Sin embargo $7 no es el menor precio ($6 en el día 4) ni $10 es el mayor precio ($11 en el día 1).
- Tampoco se vale vender antes de comprar!!!.

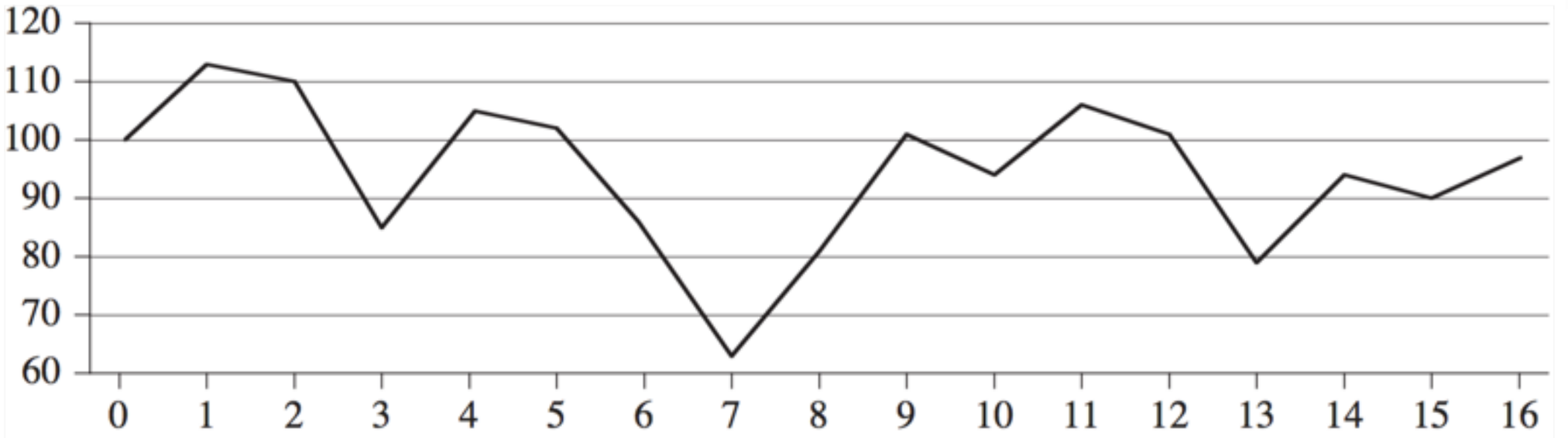
- También estaríamos tentados a ordenar los precios del menor (A) al mayor (B) y proponer los días en que se dan los precios A y B como respuesta

- Pero si B ocurre antes que A, entonces tomamos como A el siguiente al menor o tomar como B el anterior al mayor, lo que dé la mayor diferencia y A esté antes de B. Sin embargo no es claro quien se acerca primero (A o B), supongamos que alternamos (mientras A suceda después de B, avanzamos A y luego B)

- Sin embargo en el siguiente ejemplo las parejas (A,B) suceden en los días (7,1), (13,1), (13,2), (3,11).

- En este momento A sucede antes que B y la diferencia es 106-85=21.

- Pero la forma en que estamos avanzando no nos garantiza el encontrar la máxima diferencia, que en este caso ocurre entre los días 7 y 11, y que es de 106-63=43, pues ya abandonamos a 7 como posible día para compra.



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Días ordenados por precio

7 <— A
13
3
15
14
10
16
0
9
12
5
4
11
2
1 <— B

- Otro intento es irnos a la segura, es decir, A FUERZA BRUTA: tomar todas las parejas de días y hallar la máxima diferencia de precios de esos días.

- Como sabemos, con *n* datos tenemos *n(n-1)/2* parejas, aún teniendo el primer día antes del segundo, tendríamos que encontrar el máximo en un arreglo de tamaño aproximado de n^2, esto tardaría $\Theta(n^2)$.

- Sin embargo hay otra forma. TRANSFORMEMOS EL PROBLEMA:

- A partir del día 1, calculemos la diferencia de precio (el CAMBIO) con el día anterior. Esto se puede hacer en tiempo Θ(n), para todos los valores.

- Que significa tomar el i-ésimo valor de este arreglo de n-1 cambios?: la ganancia (o pérdida, si el cambio es negativo) con respecto al día anterior.

- Por ejemplo, en la figura anterior, tomar el valor en la posición 1, de 13, indica que al comprar el día 0 y vender el día 1, tendríamos una ganancia de $13

- Fíjate que pasa al sumar los cambios en las posiciones 1 y 2: 13 + (-3) = 10

- Esto nos da la ganancia de comprar el día 0 y vender el día 2

- De manera similar, al sumar los cambios en los días 8 al 11 tendremos: 18+20+(-7)+12 = 43, lo cual es la ganancia de comprar el día 7 (uno antes que el inicio del intervalo de días [8,11]) y vender el día 11 (último día del intervalo [8,11])

- Así, si encontramos la suma de los valores en el intervalo de días [*a,b*], nos dará la ganancia de comprar el día *a*-1 y vender el día *b*.

- Por lo que nuestro problema se ha convertido en encontrar el intervalo [*a,b*] que dé la máxima suma de los valores de sus diferencias.

- ¿Habremos ganado algo con esta transformación que aparentemente complica más las cosas?

- Pero el revisar cada uno de los (n-1)(n-2)/2 subarreglos (uno para cada pareja) aparentemente no nos aportó mucha ganancia, al contrario, todavía hay que sumar cada elemento del subarreglo.

- Además nota que el problema de encontrar “un” máximo subarreglo es interesante sólo si hay valores negativos. De lo contrario, la máxima suma se tiene al sumar todo el arreglo.

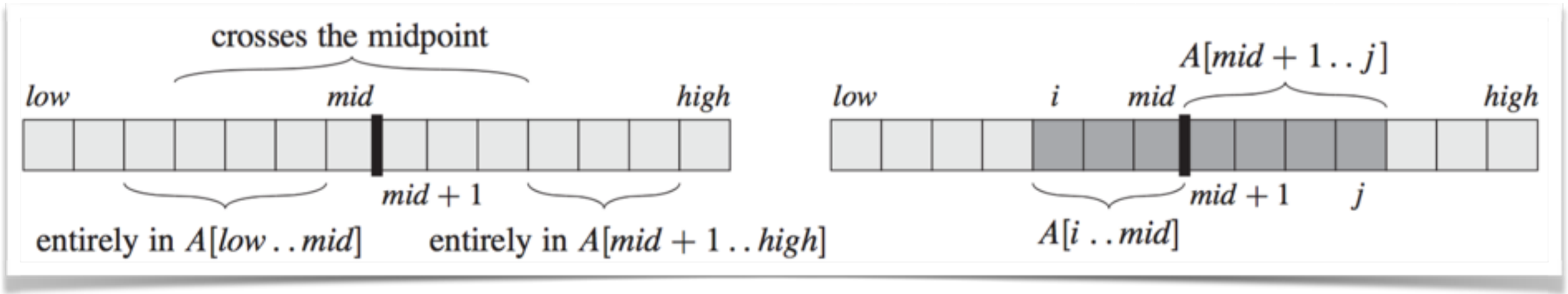
- Sin embargo tenemos una ventaja (aunque todavía no la veamos), de que ahora no hablamos de elementos individuales (que pueden estar uno antes del otro), sino a todo un subintervalo.

- Si intentamos resolverlo por divide y vencerás, tendríamos que dividir el arreglos en dos subarreglos de más o menos el mismo tamaño y entonces tendríamos 3 casos para el máximo subintervalo:

  - Que esté del lado izquierdo

  - Que esté del lado derecho

  - Que empiece del lado izquierdo y termine del lado derecho



- En los pasos recursivos podemos resolver los dos primeros casos, así nos resta el 3ero. Supongamos que nuestro intervalo en un paso recursivo, va del índice *low* a *high*, y *mid* = (*low* + *high*)/2.

- Así el subintervalo izquierdo es *A[low…mid]* y el derecho es *A[mid +*1*…high].*

- El encontrar un subintervalo máximo que empiece en el lado izquierdo y termine en el derecho, parece no ayudar mucho. Aún tenemos que hallar el máximo, con la desventaja adicional de que pasa por la mitad.

- Sin embargo veremos que esto se puede resolver en tiempo $\Theta(n)$, donde *n* es el tamaño del intervalo en el paso recursivo que actualmente se está considerando, es decir: *high* - *low* + 1.

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2       return (low, high, A[low])              // base case: only one element
 3  else mid = ⌊(low + high)/2⌋
 4       (left-low, left-high, left-sum) =
              FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5       (right-low, right-high, right-sum) =
              FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6       (cross-low, cross-high, cross-sum) =
              FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7       if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8            return (left-low, left-high, left-sum)
 9       elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10            return (right-low, right-high, right-sum)
11       else return (cross-low, cross-high, cross-sum)
```

FIND-MAXIMUM-SUBARRAY(A, 1, A.length)