presentación.js workshop.java

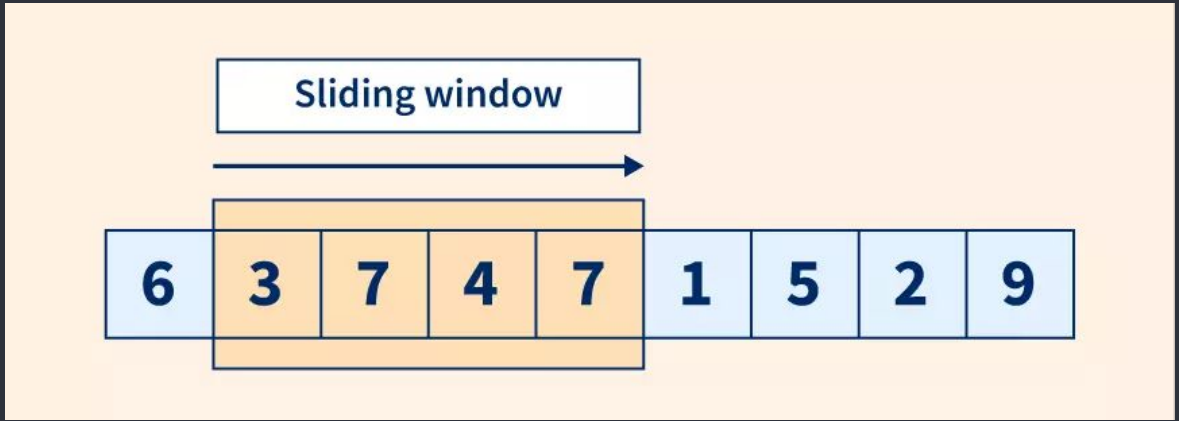
```
Club de Algoritmia US {
    [Universidad de Sevilla]

        Sesión 6: Ventanas deslizantes

}
```

Ventanas deslizantes

El algoritmo de ventana deslizante es una técnica de resolución de problemas que permite recorrer todos los subarreglos consecutivos de manera eficiente.



Ventanas deslizantes

- En situaciones donde la fuerza bruta tendría una complejidad de O(n^2) o O(n^3), la ventana deslizante nos permite reducir esta complejidad a lineal: O(n)

- La idea fundamental del algoritmo es reutilizar los cálculos realizados en el paso anterior para calcular los resultados del siguiente paso de forma óptima.

sliding_window.py

workshop.java

Ejemplo ventana deslizante

Obtener subarreglo **de tamaño** K que el cual la suma de sus elementos sea la **mayor**

Sin ventanas: O(n * k)

Con ventanas: O(n)

sliding_window.py

workshop.java

Ejemplo ventana deslizante

Obtener subarreglo **de tamaño K** que el cual la suma de sus elementos sea la **mayor**

Sin ventanas: O(n * k)

Con ventanas: O(n)

problemas.py

workshop.java

LeetCode, problema 2379

Sesión 6: Ventanas deslizantes

Problema 2379

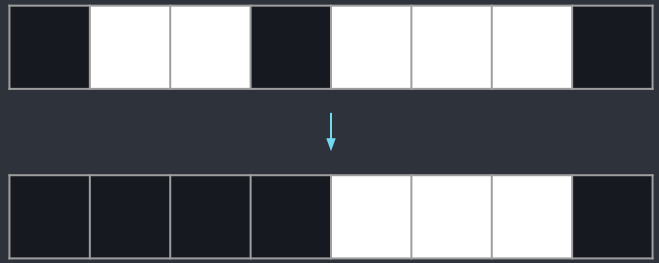
{

Tienes una cadena de caracteres “W” (white) y “B” (black) y puedes cambiar los caracteres “W” por “B”.

¿Cuál es el mínimo de cambios que necesitas realizar para que haya al menos una secuencia de “k” caracteres negros consecutivos?

}

Input: blocks = “WBBWBBBW”, k = 4
Output: 2



Solución

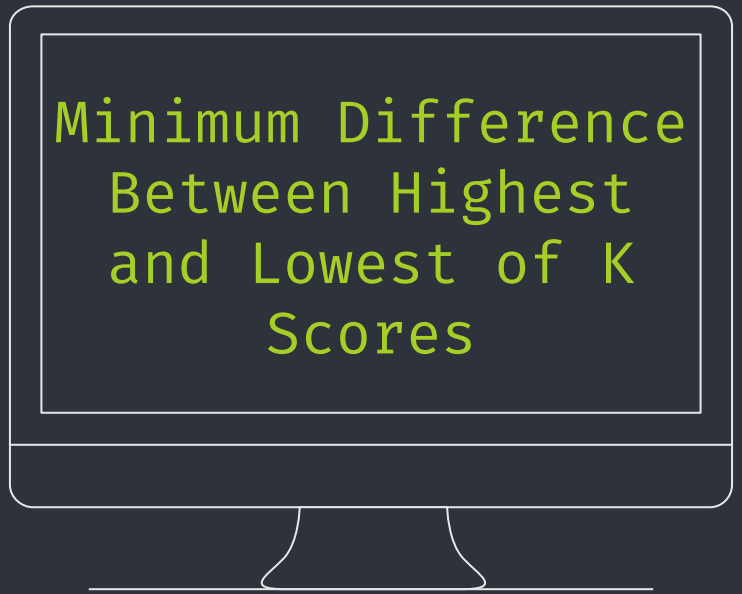
```python
def minimumRecolors(self, blocks, k):
    min_whites = 0

    for i in range(k):
        if blocks[i] == "W":
            min_whites += 1

    whites = min_whites
    for i in range(len(blocks) - k):
        if blocks[i] == "W":
            whites -= 1
        if blocks[i+k] == "W":
            whites += 1
        if whites < min_whites:
            min_whites = whites

    return min_whites
```

LeetCode, problema 1984



Problema: ‘Minimum Difference’

{

El problema consiste en tomar subgrupos de tamaño k del array y encontrar la menor diferencia entre el mayor y el menor elemento de dichos subgrupos

}

Input: nums = [90], k = 1

Output: 0

Explanation: There is one way to pick score(s) of one student

Problema: ‘Minimum Difference’

Input: nums = [9,4,1,7] k = 2

Output: 2

Explanation:
There are six ways to pick score(s) of two students.
- [**9**,**4**,1,7]. The difference between the highest and lowest score is 9 - 4 = 5.
- [**9**,4,**1**,7]. The difference between the highest and lowest score is 9 - 1 = 8.
- [**9**,4,1,**7**]. The difference between the highest and lowest score is 9 - 7 = 2.
- [9,**4**,**1**,7]. The difference between the highest and lowest score is 4 - 1 = 3.
- [9,**4**,1,**7**]. The difference between the highest and lowest score is 7 - 4 = 3.
- [9,4,**1**,**7**]. The difference between the highest and lowest score is 7 - 1 = 6.

The minimum possible difference is 2

Problema: ‘Minimum Difference’

{

Idea: Para que la diferencia entre el mayor y menor valor de un subgrupo sea mínima, sus valores deben de estar lo más cerca posible → Ordenar la lista y aplicar ventana deslizante

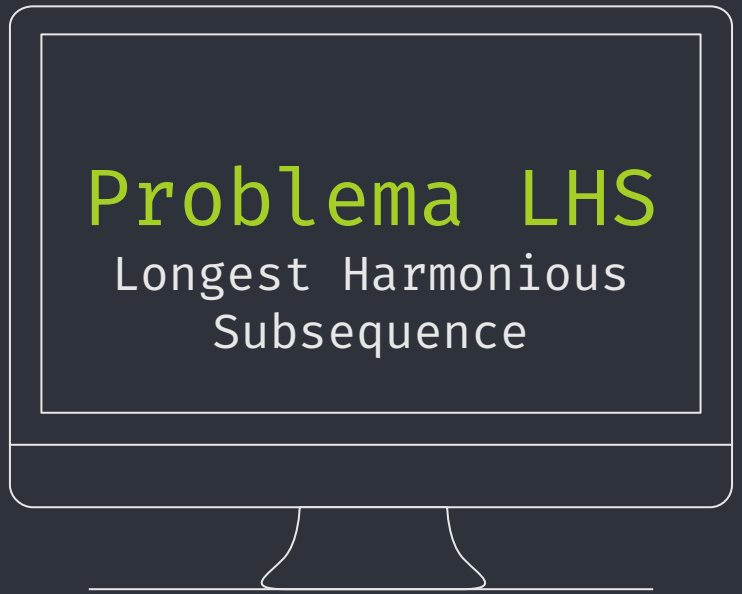
}

Solución:
Ordenar + Ventanas deslizantes

```python
class Solution:

    def minimumDifference(self, nums: List[int], k: int) -> int:
        if k == 1:
            return 0
        nums.sort()
        min_dif = float('inf')
        for i in range(len(nums) - k + 1):
            min_dif = min(min_dif, nums[i+k-1] - nums[i])
        return min_dif
```

LeetCode, problema 504



Problema: ‘LHS’

{

El problema consiste en encontrar la longitud de la subsecuencia más larga en un arreglo de enteros, donde la diferencia entre el valor máximo y el valor mínimo de la subsecuencia es exactamente 1.

}

Input: nums = [1,3,2,2,5,2,3,7]
Output: 5
Explanation: The LHS is [3,2,2,2,3].

Solución: Hash table

```python
from collections import Counter

class Solution:
    def findLHS(self, nums: List[int]) -> int:
        dicc = Counter(nums)
        acc = 0
        for n in dicc.keys():
            if n + 1 in dicc:
                acc = max(acc, dicc[n] + dicc[n + 1])

        return acc
```

Solución: Hash table

```python
from collections import Counter

class Solution:
    def findLHS(self, nums: List[int]) -> int:
        dicc = Counter(nums)
        acc = 0
        for n in dicc.keys():
            if n + 1 in dicc:
                acc = max(acc, dicc[n] + dicc[n + 1])

        return acc
```

end.py

workshop.java

¡Gracias!

¡Os esperamos la semana que viene!

- Próxima sesión: 16 de febrero
- CompliCAUS I: 23 de febrero
  - Será en HackerRank y habrá premios 🤑