Computer Society
Uniandes

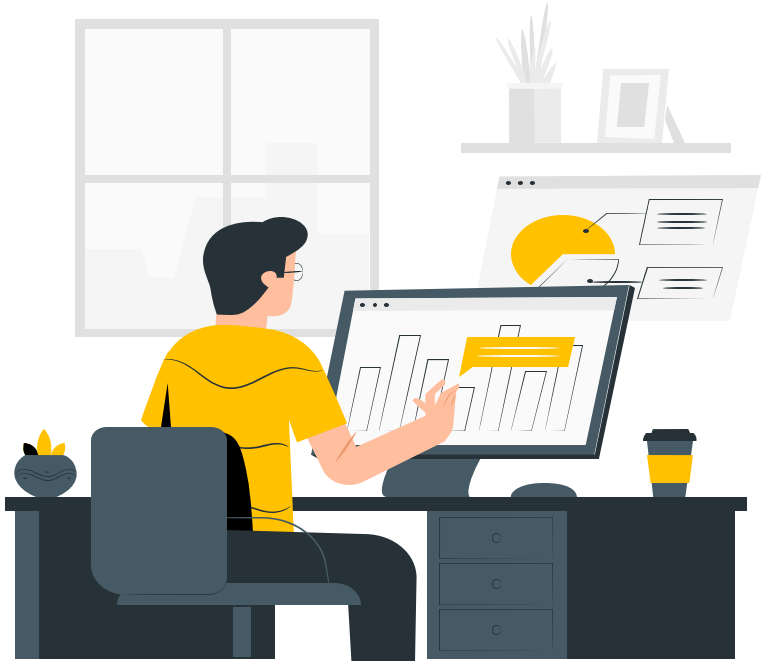
Programación Competitiva

Computer Society

Wilmer Arévalo

Tabla de Contenido

Greedy



01

Repaso del Tema Anterior

¿Cómo nos fue con los problemas de la última sesión?

¡Revisemos!

Vamos a revisar un par de problemas de aquellos que les haya parecido los más interesantes

02

Algoritmos Útiles

Algoritmos que pueden ser útiles en el aprovechamiento voraz

Two Pointers

El algoritmo de Two Pointers usa dos índices que recorren un arreglo o secuencia de manera eficiente.

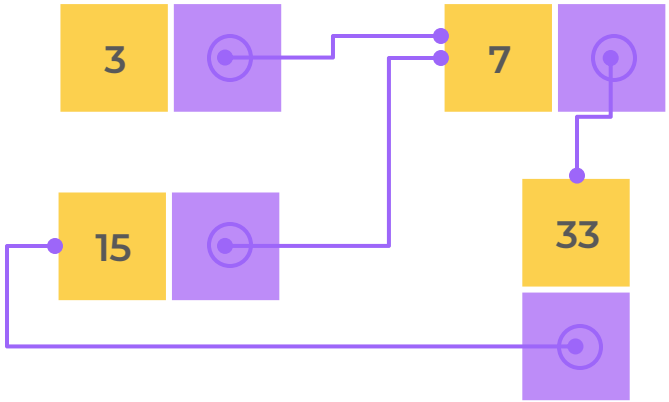
Consiste en comparar valores en los dos punteros y, con base en decisiones locales, mover los punteros

Los usos más famosos son:

- Fast/Slow pointers technic
- Start/End pointers technic

Fast/Slow

Una lista enlazada se compone de nodos y punteros que apuntan a otros nodos. Dada una lista enlazada, determine si se genera un ciclo circular, es decir, si un puntero apunta a un nodo ya visitado.



¿Alguna idea?

Fuerza Bruta

La vieja confiable, probar con todas las posibles configuraciones.

Two Pointers

¿Y si trabajamos con dos punteros?
Usemos Fast/Slow pointers technic

Ponemos los punteros en el inicio de la lista. El puntero lento avanza de a un nodo, el puntero rápido avanza de a dos nodos.

- Si un puntero llega al final (*null*), no hay ciclo.
- Si los punteros se encuentran, hay ciclo.

¿Alguna idea?

```cpp
#include <bits/stdc++.h>
using namespace std;

bool hasCircle_BruteForce(LinkedListNode* head) {
  LinkedListNode* current = head;
  while (current != nullptr) {
    LinkedListNode* checker = head;
    while (checker != current) {
      if (checker == current->next) {
        return true;
      }
      checker = checker->next;
    }
    current = current->next;
  }
  return false;
}
```

Fuerza Bruta $O(n^2)$

```cpp
#include <bits/stdc++.h>
using namespace std;

bool hasCircle(LinkedListNode* head) {
  LinkedListNode* slow = head;
  LinkedListNode* fast = head;
  while (fast != nullptr && fast->next != nullptr) {
    slow = slow->next;
    fast = fast->next->next;
    if (slow == fast) {
      return true;
    }
  }
  return false;
}
```

Two Pointers $O(n)$

Start/End

Dada una lista **ordenada** de números y un número objetivo, se debe encontrar dos números de la lista, en diferentes posiciones, cuya suma sea igual al número objetivo.

Debe retornar la posición de los números encontrados, o -1 en caso de que no se pueda.

3	5	7	9	12	15

17

¿Alguna idea?

Fuerza Bruta

La vieja confiable, probar con todas las posibles configuraciones.

Two Pointers

¿Y si trabajamos con dos punteros?
Usemos Start/End pointers technic

Ponemos los punteros al inicio y al final de la lista. En un ciclo hasta que los punteros se encuentren, sumamos los valores

- Verificar suma
- Mover punteros

¿Alguna idea?

```cpp
#include <bits/stdc++.h>
using namespace std;

pair<long long, long long> twoSum(vector<long long>& arr, long long target) {
  long long n = arr.size();

  for (long long i = 0; i < n; i++) {
    for (long long j = i + 1; j < n; j++) {

      if (arr[i] + arr[j] == target) {
        return {arr[i], arr[j]};
      }
    }
  }
  return {-1, -1};
}
```

Fuerza Bruta O(n^2)

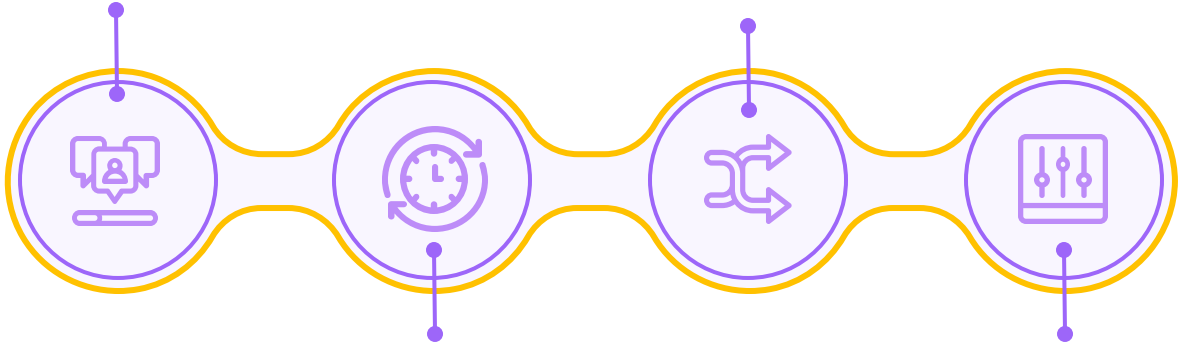
```cpp
#include <bits/stdc++.h>
using namespace std;

pair<long long, long long> twoSum(vector<long long>& arr, long long target) {
  long long i = 0, j = arr.size() - 1;

  while (i < j) {
    long long sum = arr[i] + arr[j];

    if (sum == target) {
      return {arr[i], arr[j]};
    } else if (sum < target) {
      i++;
    } else {
      j--;
    }
  }
  return {-1, -1};
}
```

Two Pointers O(n)

Estrategia Two Pointers

Punto de partida
De cada puntero

Cuando parar
Los punteros



Velocidad
De cada puntero

Variaciones
Según el problema

Sliding Window

El algoritmo Sliding Window usa una ventana de tamaño variable o fijo que se desliza sobre una secuencia.

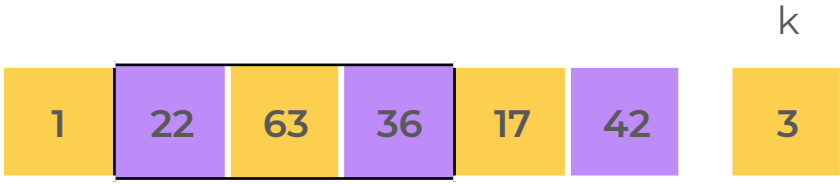
Solo se necesita un puntero* que indica el inicio de la ventana, y un entero que indica su tamaño.

Sus variantes son:

- Fixed window size technic
- Flexible window size technic

Fixed Window Size

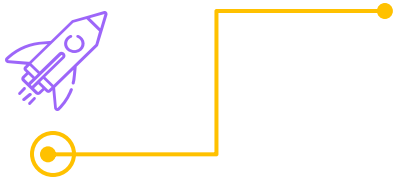
Dado un arreglo de números enteros, se debe encontrar la máxima suma de K números consecutivos.



¿Alguna idea?

Fuerza Bruta

La vieja confiable, probar con todas las posibles configuraciones.



Sliding Window

¿Y si trabajamos con Sliding Window? Usemos Fixed Window Size technic

Se debe tener en cuenta:

- Verificar el tamaño del arreglo.
- Crear la ventana.
- Iterar.
- Verificar suma.

¿Alguna idea?

```cpp
#include <bits/stdc++.h>
using namespace std;

long long maxSumSubarrayK(vector<long long>& arr, long long K) {
  long long maxSum = 0;

  for (long long i = 0; i < arr.size(); i++) {
    long long currentSum = 0;
    for (long long j = i; j < arr.size() && (j - i) < K; j++) {
      currentSum += arr[j];

      if ((j - i + 1) == K) {
        maxSum = max(maxSum, currentSum);
      }
    }
  }
  return maxSum;
}
```

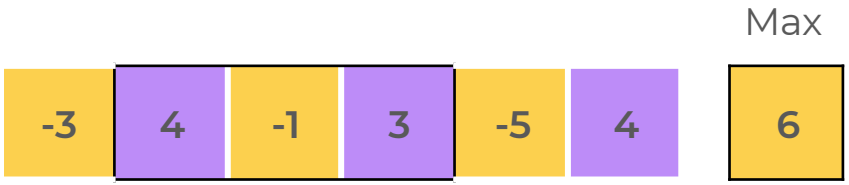
Fuerza Bruta O(n^2)

```cpp
#include <bits/stdc++.h>
using namespace std;

long long maxSumSubarrayK(vector<long long>& arr, long long K) {
  long long maxSum = 0, windowSum = 0;
  long long left = 0;

  for (long long right = 0; right < arr.size(); right++) {
    windowSum += arr[right];

    if (right >= K - 1) {
      maxSum = max(maxSum, windowSum);
      windowSum -= arr[left];
      left++;
    }
  }
  return maxSum;
}
```

Sliding Window O(n)

Flexible Window Size

Dado un arreglo de números enteros, se debe encontrar la máxima suma de números consecutivos.



¿Alguna idea?

Fuerza Bruta

La vieja confiable, probar con todas las posibles configuraciones.

Sliding Window

¿Y si trabajamos con Sliding Window? Usemos Flexible Window Size technic

Debemos tener en cuenta:

- Cuando actualizar el tamaño de la ventana
- Cuando actualizar la ubicación del puntero

¿Alguna idea?

```cpp
#include <bits/stdc++.h>
using namespace std;

long long maxSumSubarray(vector<long long>& arr) {
  long long maxSum = LLONG_MIN;

  for (long long i = 0; i < arr.size(); i++) {
    long long currentSum = 0;

    for (long long j = i; j < arr.size(); j++) {
      currentSum += arr[j];
      maxSum = max(maxSum, currentSum);
    }
  }
  return maxSum;
}
```

Fuerza Bruta O(n^2)

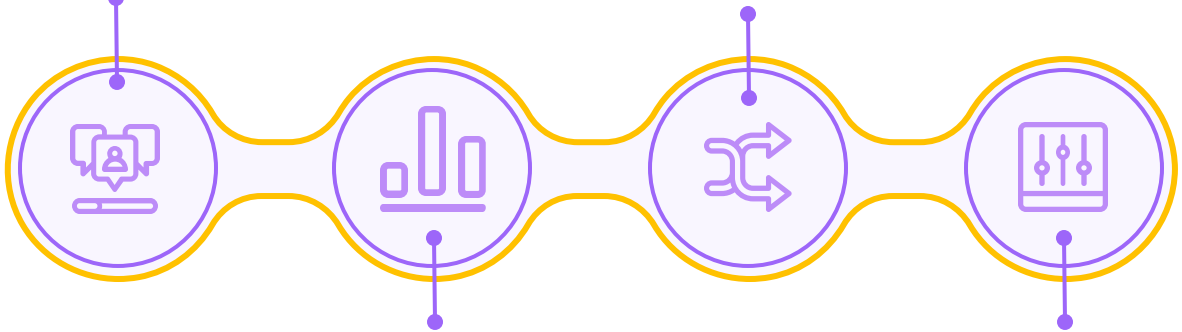
```cpp
#include <bits/stdc++.h>
using namespace std;

long long maxSumSubarray(vector<long long>& arr) {
  long long maxSum = 0, windowSum = 0;

  for (long long i = 0; i < arr.size(); i++) {
    long long num = arr[i];

    if (num >= 0){
      windowSum = windowSum < 0 ? num : windowSum + num;
      maxSum = max(maxSum, windowSum);
    } else {
      windowSum += num;
    }
  }
  return maxSum;
}
```

Sliding Window O(n)

Estrategia Sliding Window

Punto de partida
Del puntero

Cuando Actualizar
El puntero o tamaño



Tamaño
De la ventana

Variaciones
Según el problema

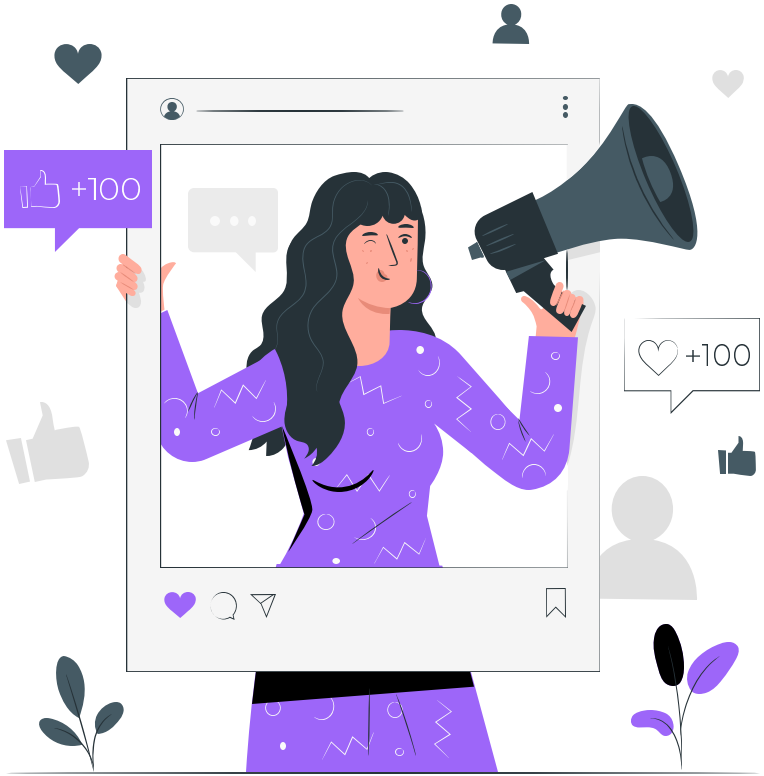
¿Por qué usar estos algoritmos?

Complejidad

Estos algoritmos mejoran la complejidad en algoritmos **usualmente** de $O(n^2)$ a $O(n)$

Aplicaciones

Podemos hacer **variaciones** de estos algoritmos que nos permitan resolver otros problemas



03

Problemas Introductorios

¡Ahora sí, empecemos!

Asignación de Tareas

Doda una lista de **n** trabajadores con sus habilidades y **m** tareas, cada una con una dificultad; Se debe asignar tareas a los trabajadores de manera que **cada trabajador solo reciba una tarea** y solo pueda completar tareas cuya dificultad sea menor o igual a su habilidad. El objetivo es maximizar el número de tareas asignadas.

n	3	10	5	8	
m	1	4	6	8	9

¿Alguna idea?

Fuerza Bruta

La vieja confiable, probar con todas las posibles configuraciones.

Two Pointers

Ordenemos
¿Y si trabajamos con Fast/Slow technic?

¿Alguna idea?

```cpp
#include <bits/stdc++.h>
using namespace std;

long long maxTasksAssigned(vector<long long>& skills, vector<long long>& tasks) {
  long long count = 0;
  vector<bool> assigned(tasks.size(), false);

  for (long long i = 0; i < skills.size(); i++) {
    for (long long j = 0; j < tasks.size(); j++) {
      if (!assigned[j] && skills[i] >= tasks[j]) {
        count++;
        assigned[j] = true;
        break;
      }
    }
  }
  return count;
}
```

Fuerza Bruta $O(n^2)$

```cpp
#include <bits/stdc++.h>
using namespace std;

long long maxTasksAssigned(vector<long long>& skills, vector<long long>& tasks) {
  sort(skills.begin(), skills.end());
  sort(tasks.begin(), tasks.end());

  long long i = 0, j = 0, count = 0;

  while (i < skills.size() && j < tasks.size()) {
    if (skills[i] >= tasks[j]) {
      count++;
      j++;
    }
    i++;
  }
  return count;
}
```

Two Pointers $O(n\log_2 n)$

Subcadena con caracteres únicos

Dada una cadena **s**, encuentra la longitud de la subcadena más larga que contenga solo caracteres únicos.
Nota: s está conformada solo por letras, no es case sensitive.

a	b	c	a	b	c	b	b

¿Alguna idea?

Fuerza Bruta

La vieja confiable, probar con todas las posibles configuraciones.

Sliding Window

¿Y si trabajamos con Flexible Window Size technic?

¿Alguna idea?

```cpp
#include <bits/stdc++.h>
using namespace std;

long long longestUniqueSubstring(string s) {
  long long maxLength = 0;

  for (long long i = 0; i < s.size(); i++) {
    unordered_set<char> seen;
    for (long long j = i; j < s.size(); j++) {
      if (seen.count(s[j])) {
        break;
      }
      seen.insert(s[j]);
      maxLength = max(maxLength, j - i + 1);
    }
  }
  return maxLength;
}
```

Fuerza Bruta O(n^2)

```cpp
#include <bits/stdc++.h>
using namespace std;

long long longestUniqueSubstring(string s) {
  unordered_map<char, long long> freq;
  long long left = 0, maxLength = 0;
  for (long long right = 0; right < s.size(); right++)
    freq[s[right]]++;
    while (freq[s[right]] > 1) {
      freq[s[left]]--;
      left++;
    }
    maxLength = max(maxLength, right - left + 1);
  }
  return maxLength;
}
```

Sliding Window O(n)

04

Más Problemas...

¡Pon a prueba tus habilidades!

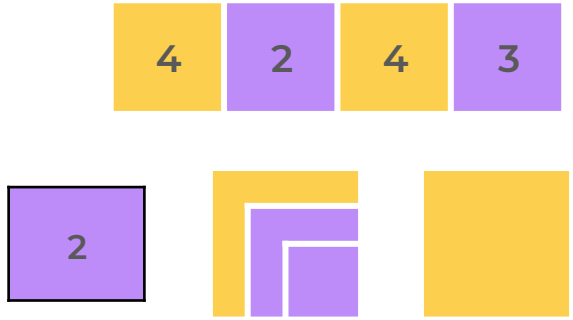
Good String

Dado un String, queremos convertirla en un "Good String". Un "Good String" es aquel cuya longitud es par y en el que cada carácter en una posición impar es diferente del siguiente carácter en la posición par. Para lograrlo, debemos eliminar la menor cantidad de caracteres posible.

G	O	O	D	0

Boxes packing

Mishka tiene n cajas cúbicas con diferentes longitudes de lado. Puede colocar una caja dentro de otra si es estrictamente más pequeña y si la caja contenedora no contiene ya otra caja. El objetivo es minimizar el número de cajas visibles, es decir, aquellas que no están dentro de otra caja.



And

Dado un arreglo de **n** elementos y un número **x**, podemos aplicar una operación a cualquier elemento **a_i** reemplazándolo por **a_i&x** (operación AND bit a bit). Queremos saber si es posible obtener al menos dos elementos iguales en el arreglo después de aplicar la operación a algunos elementos y, de ser posible, determinar el número mínimo de operaciones necesarias.

1	7	3	2

3

Mex Game 1

Alice y Bob juegan un juego con un arreglo A de tamaño n.

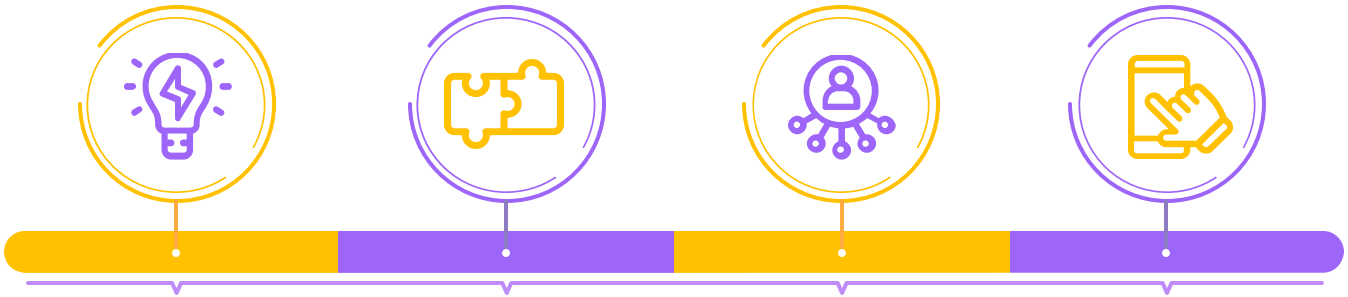
- Alice comienza con un arreglo vacío C y en cada turno elige un elemento de A, lo agrega a C y lo elimina de A.
- Bob, en su turno, simplemente elimina un elemento de A sin agregarlo a C.
- El juego termina cuando a está vacío y la puntuación final es el MEX del arreglo C, es decir, el menor número no presente en C.

Alice quiere maximizar y Bob quiere minimizar, determine el marcador si ambos juegan óptimamente

A	0	1	2	4	5

1

Recursos Adicionales



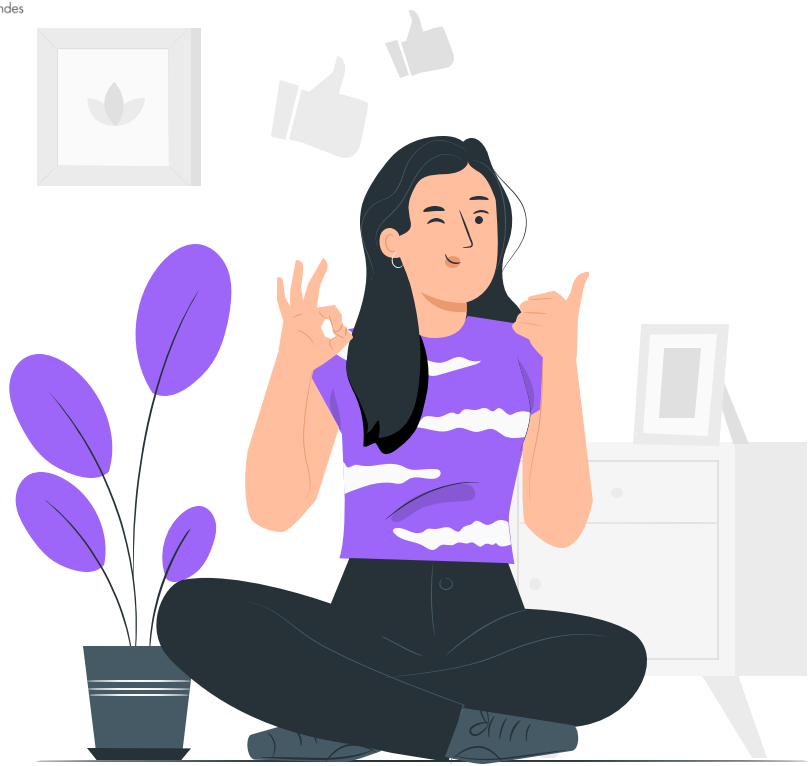
Problema

Problema

Problema

Material

Computer Society
Uniandes



¡Gracias!

Computer Society

Wilmer Arévalo