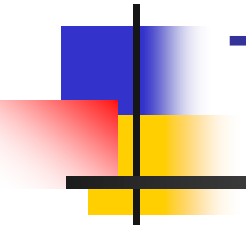
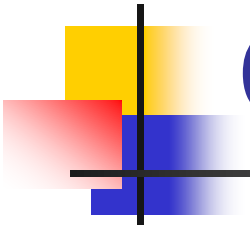


# Taller de Programación

**Parte IV**

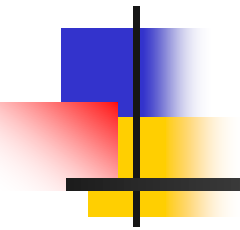
**Busqueda y Clasificación**

Jhonny Felípez Andrade
jrfelizamigo@yahoo.es



# Contenido

- Algoritmos de búsqueda.
- Clasificación u Ordenación.
- Aplicaciones de Ordenación



# Algoritmos de Búsqueda

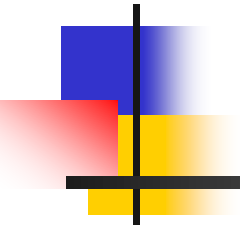
# Algoritmos de Búsqueda.
# Secuencial

```
public  class  Sec  {
   /*  Programa  de  busqueda  secuencial */
   static  int  buscar(int[]  v,  int  t)  {
      /*@  requires  v  !=  null  ;
      @  requires  t  !=  null  ;
      @  ensures    (hallo  =  -1)
      @    ||  (hallo  <  v.length  &&  hallo  >=0);
      @*/
      int  hallo  =  -1;
      //@  loop_invariant  i  <  v.length  &&  hallo  =  -1;
      for  (int  i  =  0;  i  <  v.length;  i++)
         if  (v[i]  ==  t)  { hallo  =  i; break; }
      return  (hallo);
   }
}
```

# Algoritmos de Búsqueda. Binaria

```
public  class  busqueda  {
   /*  Programa  de  busqueda  binaria */
   public  static  int  BusquedaBinaria(int[]  x,  int  t)  {
      int  l  =  0, u  =  x.length  -  1,  m  =  0,  p  =  -1;
      while  (l  <=  u)  {
         m  =  (l  +  u)  /  2;
         if  (x[m]  <  t)  l =  m + 1;
         else  if  (x[m]  ==  t)  {
                  p  =  m; break;
               } else
         u  =  m - 1;
      }
      return p;
   }
}
```

# Algoritmos de Clasificación u Ordenación



# Algoritmos de Clasificación u Ordenación

**Burbuja:** Intercambia elementos adyacentes hasta que el vector este ordenado. No se puede saber si esta ordenado hasta el final

**Inserción:** Este algoritmo tiene dos regiones, ordenada y desordenada. Lleva cada elemento a su lugar en el vector ordenado recorriendo los elementos.

**Selección:** Este algoritmo divide el arreglo de entrada en vectores ordenados y no ordenados en cada iteración encuentra el elemento más pequeño y o mueve a la parte ordenada.

# Algoritmos de Clasificación u Ordenación

**Quicksort:** Este algoritmo reduce el trabajo de ordenar dividiendo el vector en dos vectores mas pequeños a través de un proceso de partición. Este proceso separa los elementos en estos arreglos llevando los elementos mayores y menores a un elemento pivote.

**Bitsort:** Representa los elementos del vector en bits ordenados.

# Burbuja

```
void Burbuja(int[] x) {
    int i, j, temp;
    int n = x.length;
    for (i = (n - 1); i >= 0; i--) {
          for (j = 1; j <= i; j++) {
                if (x[j - 1] > x[j]) {
                      temp = x[j - 1];
                      x[j - 1] = x[j];
                      x[j] = temp;
                }
          }
    }
}
```

# Inserción

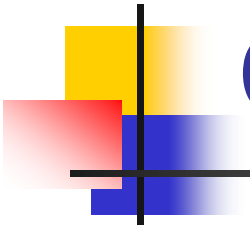
```
void Insercion(int[] x) {
   int i, j, temp;
   int n = x.length;
   for (i = 1; i < n; i++) {
       for (j = i; j > 0 && x[j - 1] > x[j]; j--) {
              temp = x[j - 1];
              x[j - 1] = x[j];
              x[j] = temp;
       }
   }
}
```

# Insercion2

```
void Insercion2(int[] x) {
   int i, j, temp;
   int n = x.length;
   for (i = 1; i < n; i++) {
       temp = x[i];
       for (j = i; j>0 && x[j - 1]>temp; j--) {
             x[j] = x[j - 1];
       }
       x[j] = temp;
   }
}
```

# Selección

```
void Seleccion(int[] x) {
    int i, j;
    int min, temp;
    int n = x.length;
    for (i = 0; i < n - 1; i++) {
           min = i;
           for (j = i + 1; j < n; j++) {
                  if (x[j] < x[min])
                         min = j;
           }
           temp = x[i];
           x[i] = x[min];
           x[min] = temp;
    }
}
```

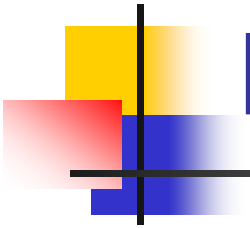


# QuickSort

```
void quickSort(int[] x) {
   qsort(x, 0, x.length - 1);
}

private void qsort(int[] x, int l, int u) {
   if (l >= u)
       return;
   int m = l, temp;
```

# QuickSort

```
    for (int i = l + 1; i <= u; i++) {
        if (x[i] < x[l]) {
                temp = x[++m];
                x[m] = x[i];
                x[i] = temp;
        }
    }

    //swap(l, m);
    temp = x[l];
    x[l] = x[m];
    x[m] = temp;
    qsort(x, l, m - 1);
    qsort(x, m + 1, u);
}
```



# Problema

Una Conversación

P: Necesito ordenar un archivo. ¿Conoces algún algoritmo?

R: ¿Porqué quieres escribir un código? ¿No puedes utilizar el del sistema?

…

R: ¿Qué exactamente estas ordenando?. ¿Cuantos registros tienes? ¿Que formato tienen los registros?

…

# Problema. Cont.

P: ¿Si el archivo es pequeño por que no ordenas en la memoria?

R: Aunque tengo muchos megabytes de memoria el programa esta al medio de un sistema y tengo cerca de 1.5 megabyte libre.

P: ¿Alguna otra cosa quiere decir sobre los registros?

R: Son enteros positivos de 7 dígitos sin duplicados.

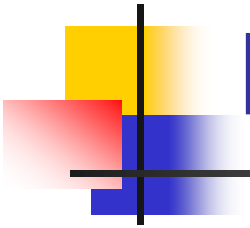
# Planteamiento preciso del problema

**Entrada (Input):** Un archivo con 10.000.000 enteros positivos donde cada n es menor a $10^7$. Se considera un error si un entero aparece duplicado en la entrada. No existe ninguna información asociada a los datos de entrada.

# Planteamiento preciso del problema. Cont.

**Salida (Output):** Una lista ordenada de enteros.

**Restricciones:** El espacio de memoria disponible es alrededor de 1.5 Mb. Y el tiempo de proceso puede ser de hasta 10 minutos y no es necesario reducirlo a menos de 10 segundos.

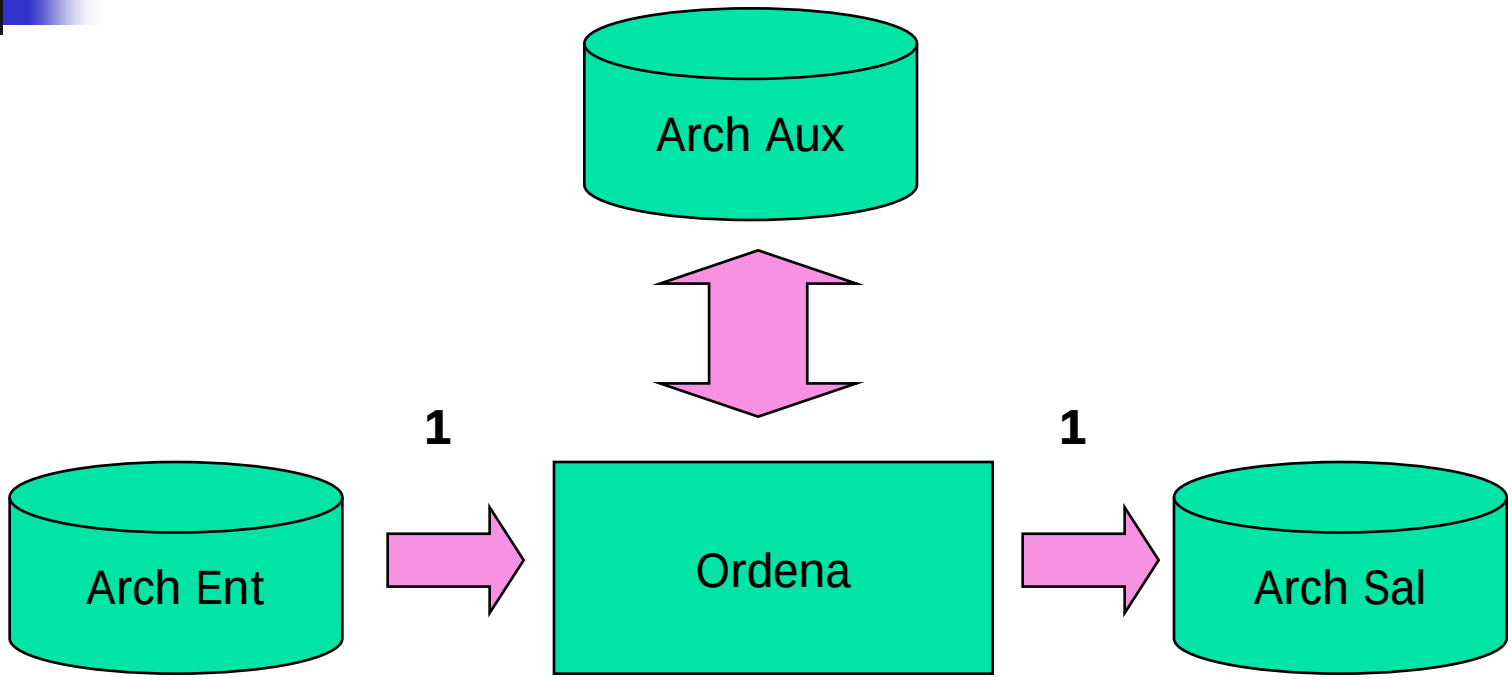


# Primera Solución Alternativa

**Ordenación en disco:** Lee una vez el archivo y escribe una vez utilizando archivos auxiliares.

# Primera Solución Alternativa

## Ordenación en disco mediante un archivo auxiliar




**Desventaja mucho uso de disco.**

# Segunda Solución Alternativa

Análisis:

- Espacio de memoria disponible 1.5 MB = 1.572.864 bytes.
- C/registro es un entero de 7 dígitos (9.999.999) ingresa en un **long** (0 a 4.294.967.295 o -2.147.483.648 a 2.147.483.647) ocupa 4 bytes.

¿Cuántos enteros de 4 bytes se almacenaran en 1 MB (menos de 1.5 MB)?

R. 1MB / 4 = 262.144 > 250.000 enteros

# Segunda Solución Alternativa

¿Cuántos grupos de 250.000 enteros se tiene en 10.000.000 enteros?

R. 10.000.000 / 250.000 = 40 grupos

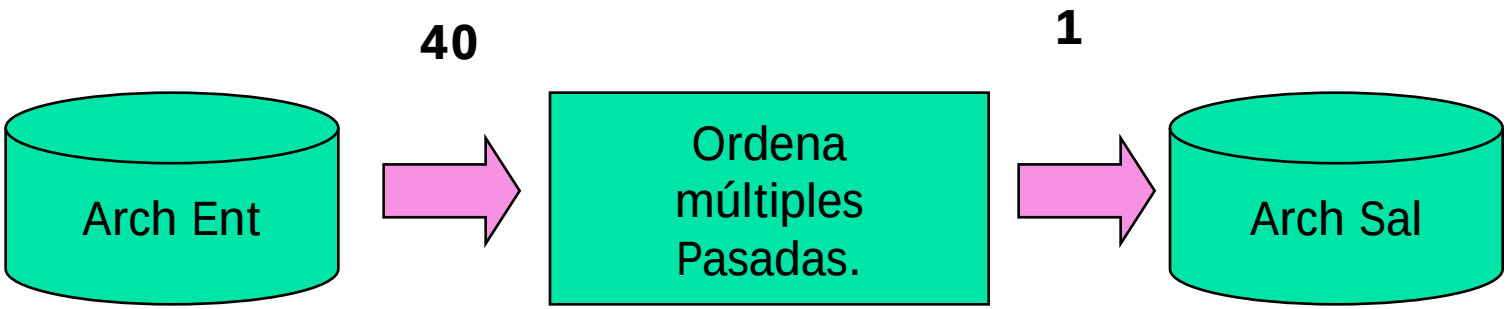
Agrupando en 40 grupos los 10.000.000 enteros se tiene:

| | | | | |
|---|---|---|---|---|
| Del | 0 | al | 249.999 | primera pasada |
| Del | 250.000 | al | 499.999 | segunda pasada |
| Del | 500.000 | al | 749.999 | tercera pasada |
| Del | 750.000 | al | 999.999 | cuarta pasada |
| … | | | | |
| Del | 9.750.000 | al | 9.999.999 | 40ava pasada |

# Segunda Solución Alternativa

**Ordenación de Múltiples Pasadas:** Lee la entrada 40 veces y graba la salida una vez ordenado, sin utilizar archivos auxiliares.

# Solución Alternativa
## Ordenación de Múltiples Pasadas

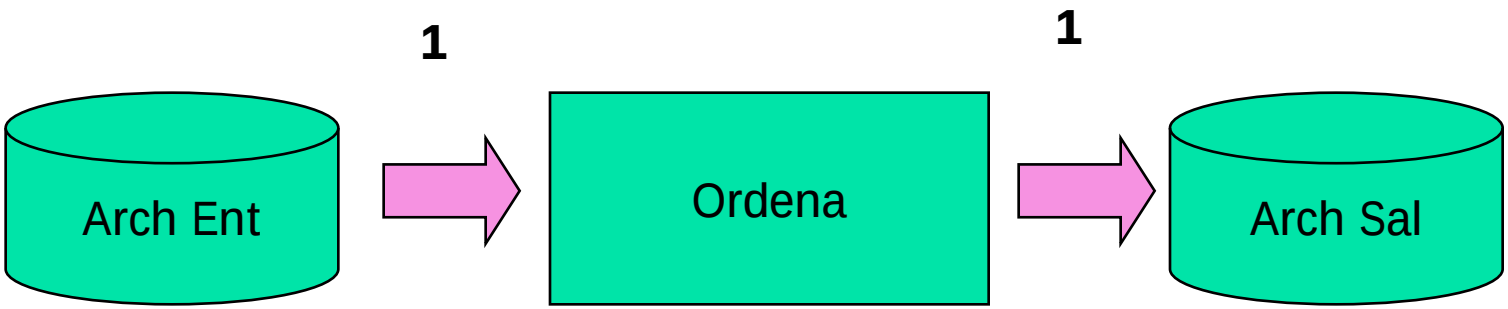



**Desventaja se lee el archivo de entrada 40 veces.**

# Tercera Solución Alternativa

**Ordenación Bitsort:** Lee la entrada una vez y no utiliza archivos intermedios.

# Tercera Solución Alternativa
## Ordenación Bitsort




**Solución Optima!**

# Tercera Solución Alternativa
## Ordenación Bitsort

Podemos hacer lo anterior si representamos todos los enteros del archivo de entrada en 1.5 MB disponibles.

# Tercera Solución Alternativa
## Ordenación Bitsort

Analizando esta idea.

Dada una lista de números menores a 10 es posible representar en una lista de 10 bits.

Por ejemplo se puede almacenar el conjunto {1,2,3,6,8} en esta cadena 0 1 1 1 0 0 1 0 1 0.

Los bits que representan a los números del conjunto son 1, y todos los otros bits son 0.

# Tercera Solución Alternativa
## Ordenación Bitsort

En el problema real, los siete dígitos decimales de cada entero denota un número menor que diez millones.

Representamos el archivo por una cadena de diez millones de bits en el cual el i-ésimo bit está encendido si solo si el entero i está en el archivo.

# Tercera Solución Alternativa
## Ordenación Bitsort. Etapas

```
/* etapa 1 Inicializar el vector bit en cero */
   for i = 0  to n-1
      bit[i] = 0
/* etapa 2 Insertar los elementos al vector */
   for each i en el archivo de entrada
      bit[i] = 1
/* etapa 3 Escribir la salida ordenada */
   for i = 0 to n-1
      if bit[i] == 1
         write i en el archivo de salida
```

# Implementación en C

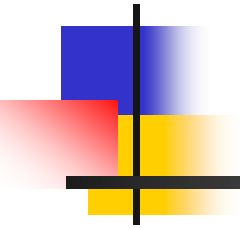
```
/* bitsort.c 'Programming Pearls' by Jon Bentley
 * Ordena enteros distintos en el rango [0..N-1] */

#include <stdio.h>

#define BITSPERWORD 32
#define SHIFT 5
#define MASK 0x1F
#define N 10000000
int a[1 + N/BITSPERWORD];

void set(int i) {        a[i>>SHIFT] |=  (1<<(i & MASK)); }
void clr(int i) {        a[i>>SHIFT] &= ~(1<<(i & MASK)); }
int  test(int i){ return a[i>>SHIFT] &   (1<<(i & MASK)); }
```

# Implementación en C

```
int main()
{  int i;
// for (i = 0; i < N; i++)
//     clr(i);
   int top = 1 + N/BITSPERWORD;
   for (i = 0; i < top; i++)
       a[i] = 0;
   while (scanf("%d", &i) != EOF)
       set(i);
   for (i = 0; i < N; i++)
       if (test(i))
              printf("%d\n", i);
   return 0;
}
```

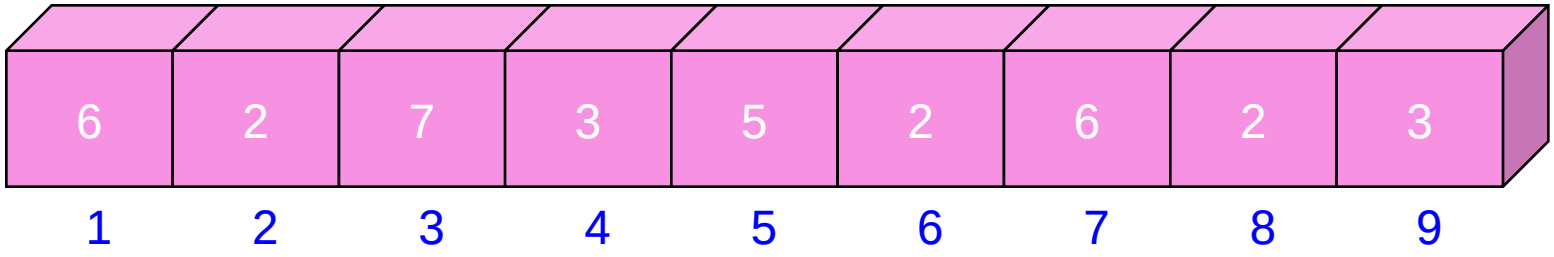
# Implementación en JAVA

```
public static void main(String[] args){
   StringTokenizer st;
   String s;
   int i;
   int BITSPERWORD = 32, SHIFT = 5;
   int MASK = 0x1F, N = 10000000;
   int top = 1 + N/BITSPERWORD;
   int [] a = new int[top];
   for (i = 0; i < top; i++)
      a[i] = 0;
   while ((s=Main.readLn(255)) != null){
      st = new StringTokenizer(s);
      i = Integer.parseInt(st.nextToken());
      a[i>>SHIFT] |= (1<<(i & MASK));
   }
   for (i = 0; i < N; i++){
      if ((a[i>>SHIFT] & (1<<(i & MASK))) > 0)
         System.out.println(i);
   }
}
```



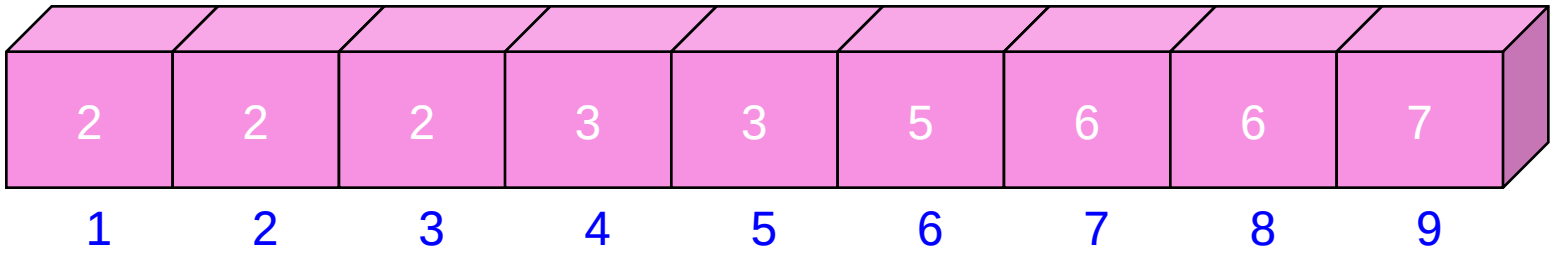
# Aplicaciones de Ordenación

# Aplicación de Ordenación
## Comprobar Duplicados



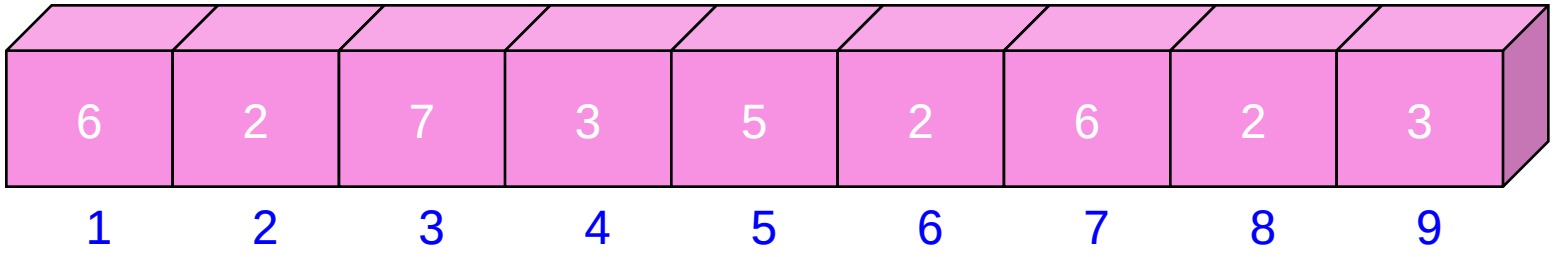

¿Cómo probar que en un grupo de elementos no existen elementos duplicados?

# Aplicación de Ordenación
## Comprobar Duplicados



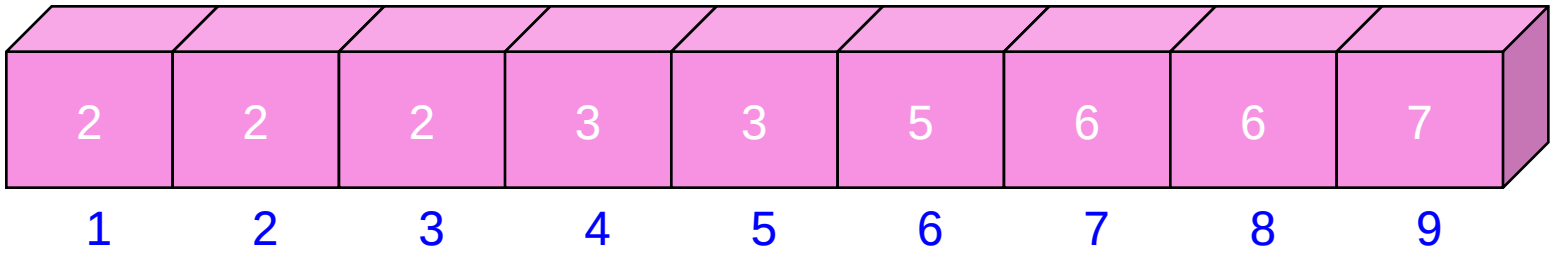

Ordenar los elementos y luego verificar que no exista x[i] = x[i + 1] recorriendo la lista.

# Aplicación de Ordenación
# Eliminar Duplicados




¿Cómo podemos eliminar los elementos duplicados?

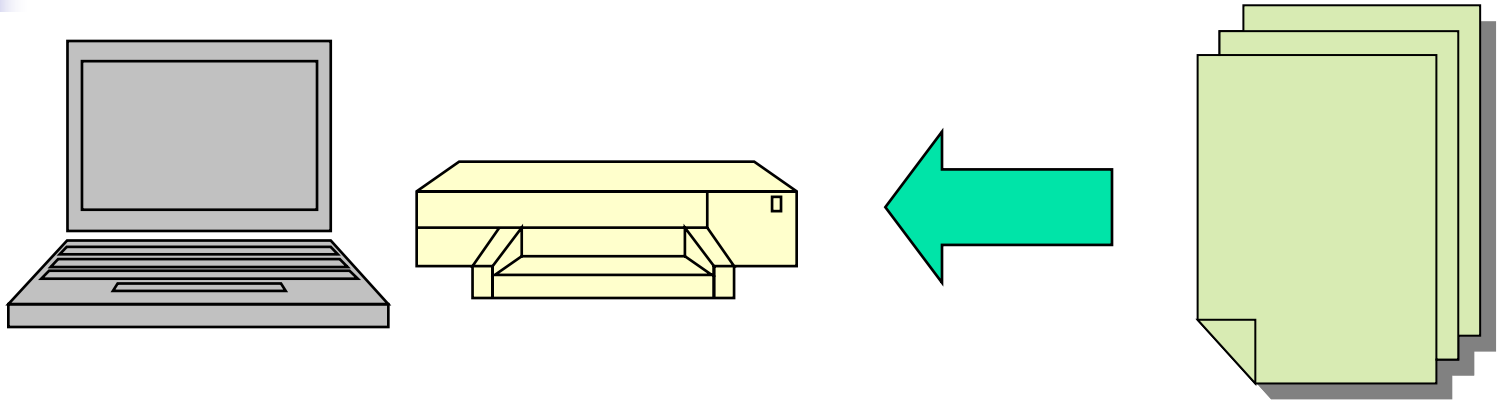
# Aplicación de Ordenación
# Eliminar Duplicados




**Ordenar los elementos y luego aplicar**:

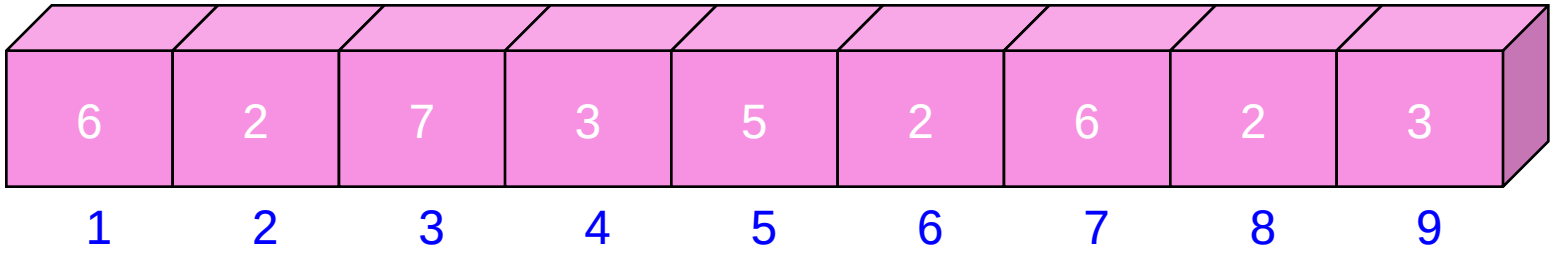
```
i=1; j=2;
  while (j<= n) {
     if (x[i] != x[j]) then  {i = i+1; x[i] = x[j];}
     j=j+1;
  }
  n=i;
```

# Aplicación de Ordenación
## Lista de Trabajos



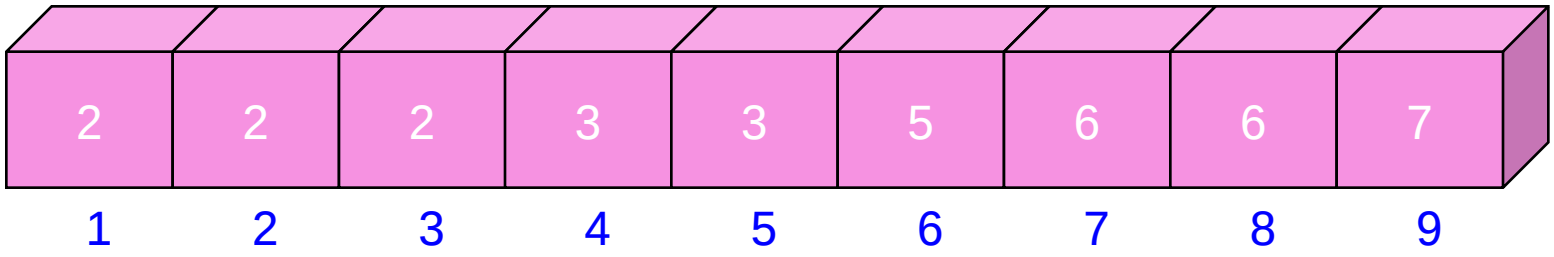
- En una lista de trabajos (ej. cola de impresión), se puede asignar una prioridad. Ordenar los trabajos según la prioridad y luego se procesa en este orden.

# Aplicación de Ordenación
# Frecuencias



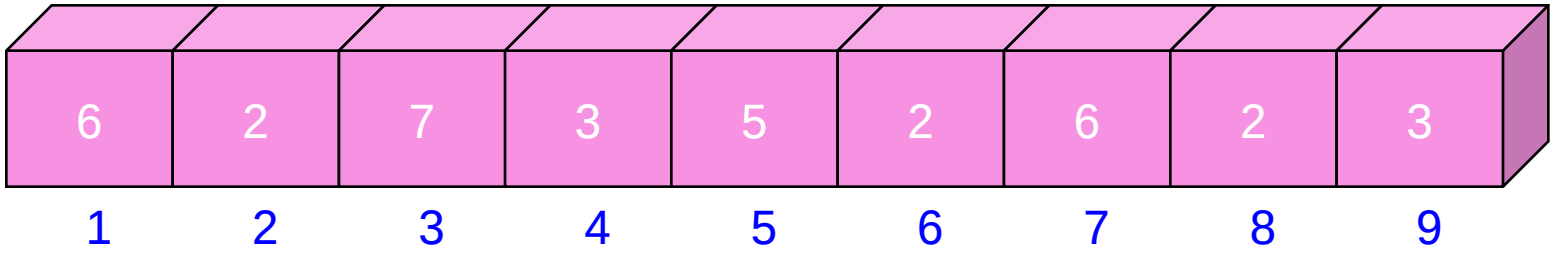

¿Cómo obtener la frecuencia de números?

# Aplicación de Ordenación
# Frecuencias



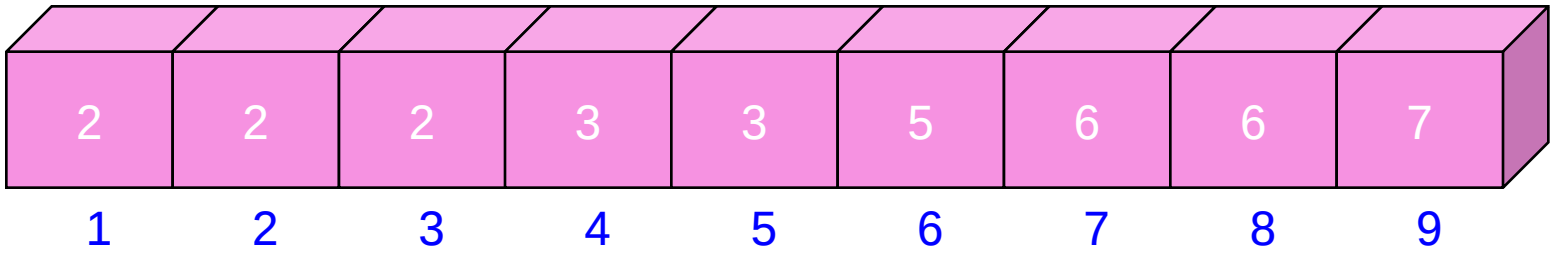

Ordenar los elementos y luego contar por grupos mientras X[i] = X[i+1], luego cada vez que sea distinto imprimir el total. Esto se denomina corte de control!

# Aplicación de Ordenación
# Mediana




¿Cómo obtener la Mediana?

# Aplicación de Ordenación
## Mediana




**Ordenar los elementos y luego obtener:**

k = int(n/2), la mediana es el elementos k-ésimo es decir el X[k].

# Aplicación de Ordenación
# Unión Conjuntos

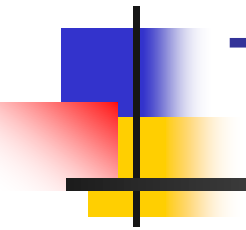
{a, b, c, d, e} U {x, c, z}

= {a, b, c, d, x, z}

- Para realizar unión de conjuntos se pueden ordenar los conjuntos. Cuando hay dos elementos iguales eliminar los duplicados.

# Bibliografía


- Fundamentos de Programación, Jorge Teran Pomier, 2006.
- Programming Pearls, Jon Bentley. Segunda edición, Addison Wesley, 2000.



# Taller de Programación

Gracias