Trabajo práctico obligatorio

Métodos de ordenamiento

Leo Joaquin Bruno - FAI 3268

Maria Elvira Monserrat Vidal - FAI 1829

Jeremias Ezequiel Herrera - FAI 3297

Enlace a GITHUB: https://github.com/LeoBruno400/MetodosDeOrdenamientos

11 de junio del 2022

1. **Implemente el algoritmo de Búsqueda Secuencial.**

   (a) ¿A qué clase pertenece y por qué razón?

   Pertenece a la clase de fuerza bruta ya que va recorriendo elemento por elemento hasta encontrar el valor deseado.

   (b) Realice el algoritmo en pseudocódigo y java

   Pseudocódigo

```
ALGORITMO ejercicio01 () RETORNA ∅
        ENTERO [] arreglo ← CREAR ENTERO {4,5,10,-7,2,20,22,45,22,1,66,2,55,30}
        ESCRIBIR(busquedaSecuencial(arreglo, -7))
FIN ALGORITMO

MODULO busquedaSecuencial (ENTERO [] arreglo) RETORNA ENTERO
        ENTERO posicion←-1
        ENTERO i←0
        REPETIR
        i ← i + 1
                MIENTRAS(i<longitud(arreglo) AND arreglo[i]<>buscado) HACER
                        SI(arreglo[i]=buscado)
                        posicion = i
                        FIN SI
                FIN MIENTRAS
        RETORNA posicion
FIN MODULO
```

   Java

```
public class busquedaSecuencial {
    public static void main(String[] args) {
        int [] arreglo = {4,5,10,-7,2,20,22,45,22,1,66,2,55,30};
        System.out.println(busquedaSecuencial(arreglo, -7));
    }

    public static int busquedaSecuencial (int [] arreglo, int buscado){
        int posicion=-1, i=0;
        do{
            i++;
        }while(i<arreglo.length && arreglo[i]!=buscado);
if(arreglo[i]==buscado){
                posicion = i;
            }
        return posicion;
    }
}
```

(c) Realice la traza para el arreglo:

arreglo:

89	45	63	90	29	34	17

MODULO busquedaSecuencial(arreglo, 63)

arreglo	buscado	i	posicion	RETORNA
*	63	0	-1	2
		1	2	
		2		

(d) Una variante del algoritmo sería aquel que realiza la búsqueda de atrás hacia adelante. Realice una implementación de esta variante.

```
public static int busquedaSecuencialReversa (int [] arreglo, int buscado){
      int posicion=-1, i=arreglo.length-1;
      do{
         i--;
      }while(i>=0 && arreglo[i]!=buscado);
      if(arreglo[i]==buscado){
         posicion = i;
      }
      return posicion;
   }
```

(e) Calcule el Tiempo de Ejecución y Orden.

```
1 public static int busquedaSecuencialReversa (int [] arreglo, int buscado){
2         int posicion=-1, i=arreglo.length-1;
3         do{
4             i--;
5         }while(i>=0 && arreglo[i]!=buscado);
6         if(arreglo[i]==buscado){
7             posicion = i;
8         }
9         return posicion;
10    }
```

Tn = T3 + T4 = 1 + (5n + 10) = 5n + 11
T3 = 1
T4 = T9 + T10 + T13 + T16 = 2 + (5n + 4) + 3 + 1 = 5n + 10
T9 = 2
T10 = n * (4 + 1) + 4 = 5n + 4
T13 = 2 + T14 = 2 + 1 = 3
T14 = 1
T16 = 1

f(n) = 5n + 11
Complejidad lineal.
O(n)

2. **Implemente el algoritmo de Búsqueda Binaria.**
   (a) ¿A qué clase pertenece y por qué razón?
   Pertenece a la clase Divide y Vencerás porque un problema se divide en varios subproblemas más pequeños y fáciles de resolver, con temáticas similares . La solución se obtiene resolviendo cada uno de los subproblemas de manera independiente en forma recursiva o iterativa, y luego combinar sus soluciones para obtener la solución al problema original.
   (b) Realice el algoritmo en pseudocódigo y java
   Pseudocódigo

```
MODULO busquedaBinaria (ENTERO[] arreglo, ENTERO valorBuscado) RETORNA ENTERO
        LOGICO encontrado
        ENTERO inicio, fin, medio, posicion
        inicio ← 0
        fin ← LONGITUD(arreglo)
        encontrado ← falso
        posicion ← -1
            MIENTRAS inicio <= fin AND !encontrado HACER
                medio ← (inicio + fin)/2
                SI arreglo [medio] == valorBuscado ENTONCES
                    posicion ← medio
                    encontrado ← verdadero
                SINO
                    SI arreglo[medio] > valorBuscado ENTONCES
                        fin ← medio -1
                    SINO
                        inicio ← medio +1
                    FIN SI
                FIN SI
            FIN MIENTRAS
        RETORNA posicion
FIN MODULO
```

Java

```
  public static int busquedaBinaria(int[] arreglo, int valorBuscado){
      //busca de manera binaria un valor numérico en un arreglo. Devuelve la posición de encontrarlo, -1 de no encontrarlo.
      boolean encontrado;
      int inicio, fin, medio, posicion;
      inicio = 0;
      fin = arreglo.length-1;
      encontrado = false;
      posicion = -1;
      while (inicio <= fin && !encontrado){
         medio= (inicio + fin)/2;
         if(arreglo[medio] == valorBuscado){
            posicion = medio;
            encontrado = true;
         }else{
            if(arreglo[medio]> valorBuscado){
               fin = medio -1;
            }else{
               inicio = medio +1;
            }
         }
      }
      return posicion;
  }
```

(c) Realice la traza para el arreglo:

arreglo:

2	4	6	8	10	12	14	16	18	20

MODULO busquedaBinaria(arreglo, 12)

encontrado	inicio	fin	medio	posicion	RETORNA
false	~~0~~	~~9~~	~~4~~	~~-1~~	
	5				
			~~7~~		
		6			
			5		
				5	
					5

(d) Una variante del algoritmo sería aquel que realiza la búsqueda en un arreglo ordenado en forma decreciente. Realice una implementación de esta variante.

Pseudocódigo

```
MODULO busquedaBinariaInvertida (ENTERO[] arreglo, ENTERO valorBuscado) RETORNA ENTERO
        LOGICO encontrado
        ENTERO inicio, fin, medio, posicion
        inicio ← 0
        fin ← LONGITUD(arreglo)
        encontrado ← falso
        posicion ← -1
        MIENTRAS inicio <= fin AND !encontrado HACER
                medio ← (inicio + fin)/2
                SI arreglo [medio] == valorBuscado ENTONCES
                        posicion ← medio
                        encontrado ← verdadero
                SINO
                        SI arreglo[medio] < valorBuscado ENTONCES
                                fin ← medio -1
                        SINO
                                inicio ← medio +1
                        FIN SI
                FIN SI
        FIN MIENTRAS
        RETORNA posicion
FIN MODULO
```

Java

```java
  public static int busquedaBinariaInvertida(int[] arreglo, int valorBuscado){
      //busca de manera binaria un valor numérico en un arreglo. Devuelve la posición de encontrarlo, -1 de no encontrarlo.
      boolean encontrado;
      int inicio, fin, medio, posicion;
      inicio = 0;
      fin = arreglo.length-1;
      encontrado = false;
      posicion = -1;
      while (inicio <= fin && !encontrado){
          medio= (inicio + fin)/2;
          if(arreglo[medio] == valorBuscado){
              posicion = medio;
              encontrado = true;
          }else{
              if(arreglo[medio] < valorBuscado){
                  fin = medio -1;
              }else{
                  inicio = medio +1;
              }
          }
      }
      return posicion;
  }
```

(e) Calcule el Tiempo de Ejecución y Orden.

```java
public static int busquedaBinaria(int[] arreglo, int valorBuscado){
        //busca de manera binaria un valor numérico en un arreglo. De
        boolean encontrado;
        int inicio, fin, medio, posicion;
        inicio = 0;
        fin = arreglo.length-1;
        encontrado = false;
        posicion = -1;
        while (inicio <= fin && !encontrado){
            medio= (inicio + fin)/2;
            if(arreglo[medio] == valorBuscado){
                posicion = medio;
                encontrado = true;
            }else{
                if(arreglo[medio] < valorBuscado){
                    fin = medio -1;
                }else{
                    inicio = medio +1;
                }
            }
        }
        return posicion;
    }
```

$T_n = T_5 + T_6 + T_7 + T_8 + T_9 + T_{22}$

$T_5 = 1 \quad T_6 = 1 \quad T_7 = 1 \quad T_8 = 1 \quad T_{22} = 1$

$T_9 = \log n \cdot (3 + T_{10} + T_{11}) + 3 = \log n (3+3+6) + 3 = 12 \cdot \log n + 3$

$T_{10} = 3$

$T_{11} = 2 + \max(T_{12}, T_{15}) \qquad T_{12} = T_{12} + T_{13} = 1 + 1 = 2$

$T_{11} = 2 + 4 = 6 \qquad T_{15} = 2 + \max(T_{16}, T_{18}) = 2 + 2 = 4$

$T_{16} = 2 \qquad T_{18} = 2$

$T_n = 1 + 1 + 1 + 1 + 1 + 12 \cdot \log n + 3 = 12 \cdot \log n + 8$

f(n) = 12*log(n) +8

Complejidad Logarítmica

O(logn)

3. Implemente el algoritmo de Ordenamiento por Selección

(a) ¿A qué clase pertenece y por qué razón?

(b) Realice el algoritmo en pseudocódigo y java

(c) Realice la traza para el arreglo:

7	6	11	17	3	15	5	19	30	14

(d) Una variante del algoritmo sería aquel que realiza el ordenamiento de mayor a menor. Realice una implementación de esta variante

a) Pertenece a la clase de fuerza bruta ya que debe recorrer todos los elementos si o si.

b)

```
public static void main(String[] args) {
        int [] arreglo = {4,5,10,-7,2,20,22,45,22,1,66,2,55,30};
        ordenarPorSeleccion(arreglo);
    }
    public static void ordenarPorSeleccion (int [] arreglo){
        int posMenor;
        for (int i = 0; i < arreglo.length; i++) {
            posMenor = buscarMenor(arreglo, i);
            if(arreglo[posMenor]<arreglo[i]){
                intercambiar(arreglo, posMenor, i);
            }
        }
    }

    public static void intercambiar(int [] arreglo,int posMenor,int i){
        int aux;
        aux = arreglo[i];
        arreglo[i] = arreglo[posMenor];
        arreglo[posMenor] = aux;
    }

    public static int buscarMenor (int[] arreglo, int i){
        int menorNum = 999999, posicion=i;
        for (int j = i; j < arreglo.length; j++) {
            if(arreglo[j]<menorNum){
                menorNum = arreglo[j];
                posicion = j;
            }
        }
        return posicion;
    }
}
```

```
ALGORITMO ejercicio03 () RETORNA ∅
        ENTERO [] arreglo ← {4,5,10,-7,2,20,22,45,22,1,66,2,55,30}
        ordenarPorSeleccion(arreglo)
FIN ALGORITMO

MODULO ordenarPorSeleccion(ENTERO [] arreglo)
        ENTERO posMenor, i
        PARA i←0 HASTA longitud(arreglo)-1 PASO 1
                posMenor ←buscarMenor(arreglo, i)
                SI(arreglo[posMenor]<arreglo[i])
                        intercambiar(arreglo, posMenor, i)
                FIN SI
        FIN PARA
FIN MODULO

MODULO intercambiar(ENTERO [] arreglo, ENTERO posMenor, ENTERO i)
        ENTERO  aux
        aux ← arreglo[i]
        arreglo[i] ← arreglo[posMenor]
        arreglo[posMenor] ← aux
```

```
FIN MODULO

MODULO buscarMenor (ENTERO [] arreglo, ENTERO i)
        ENTERO menorNum←9999999
        ENTERO posicion←i
        ENTERO j
        PARA j←i HASTA longitud(arreglo)-1 PASO 1
                SI(arreglo[j]<menorNum)
                        menorNum←arreglo[j]
                        posicion←j
                FIN SI
        FIN PARA
FIN MODULO
```

d)

```
public static void ordenarPorSeleccionVariante (int [] arreglo){
    int posMayor;
    for (int i = arreglo.length-1; i >= 0; i--) {
        posMayor = buscarMenorVariante(arreglo, i);
        if(arreglo[posMayor]<arreglo[i]){
            intercambiar(arreglo, posMayor, i);
        }
    }
}
public static int buscarMenorVariante (int[] arreglo, int i){
    int menorNum = 999999, posicion=i;
    for (int j = 0; j <= i; j++) {
        if(arreglo[j]<menorNum){
            menorNum = arreglo[j];
            posicion = j;
        }
    }
    return posicion;
}
```

7	6	11	17	3	15	5	19	30	14

i	posMenor	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Cambio
~~0~~	~~4~~	7	6	11	17	3	15	5	19	30	14	Posicion 4 por 0 →
~~1~~	~~6~~	3	6	11	17	7	15	5	19	30	14	Posicion 6 por 1 →
~~2~~	~~6~~	3	5	11	17	7	15	6	19	30	14	Posicion 6 por 2 →
~~3~~	~~4~~	3	5	6	17	7	15	11	19	30	14	Posicion 4 por 3 →
~~4~~	~~6~~	3	5	6	7	17	15	11	19	30	14	Posicion 6 por 4 →
~~5~~	~~9~~	3	5	6	7	11	15	17	19	30	14	Posicion 9 por 5 →
~~6~~	~~9~~	3	5	6	7	11	14	17	19	30	15	Posicion 9 por 6 →
~~7~~	~~9~~	3	5	6	7	11	14	15	19	30	17	Posicion 9 por 7 →
~~8~~	~~9~~	3	5	6	7	11	14	15	17	30	19	Posicion 9 por 8 →
~~9~~	9	3	5	6	7	11	14	15	17	19	30	No hace cambio
10												

No hago traza de las invocaciones del resto de modulos porque sino se haria muy extenso

4. **Implemente el algoritmo de Ordenamiento Burbuja.**
   (a) ¿A qué clase pertenece y por qué razón?
   El ordenamiento burbuja pertenece a la clase de fuerza bruta porque enumera sistemáticamente todos los posibles candidatos para la solución de un problema con el fin de chequear si dicho candidato satisface la solución al mismo.
   (b) Realice el algoritmo en pseudocódigo y java

   Pseudocódigo

```
MODULO burbuja (ENTERO[] arreglo) RETORNA Ø
        ENTERO i, j, longitud, aux
        longitud = LONGITUD(arreglo)
        PARA i← 0 HASTA longitud -1 PASO 1 HACER
                PARA j ← HASTA longitud - i - 2 PASO 1 HACER
                        SI arreglo[j] > arreglo[j+1] ENTONCES
                                aux ← arreglo[j]
                                arreglo[j] ← arreglo[j+1]
                                arreglo[j+1] ← aux
                        FIN SI
                FIN PARA
        FIN PARA
FIN MODULO
```

   Java

```
public static void burbuja (int[] arreglo){
    int i,j, longitud, aux;
    longitud = arreglo.length;
    for (i= 0; i < longitud; i++) {
        for(j=0; j<(longitud-i-1); j++){
            if(arreglo[j]>arreglo[j+1]){
                aux = arreglo[j];
                arreglo [j] = arreglo[j+1];
                arreglo [j+1] = aux;
            }
        }
    }
}
```

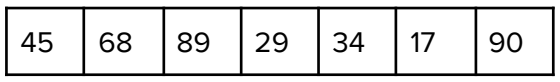
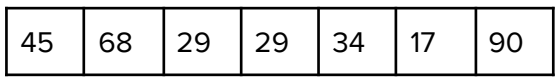
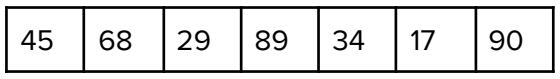
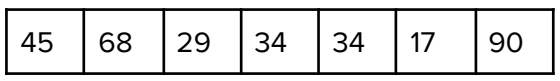
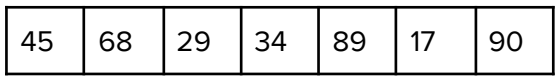
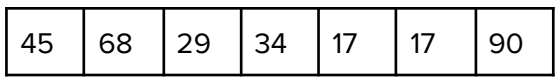
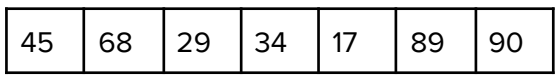
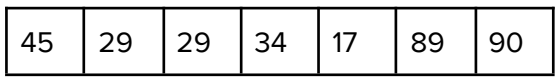
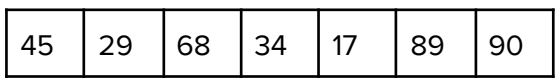
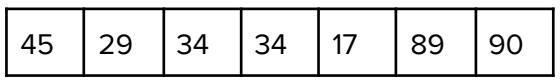
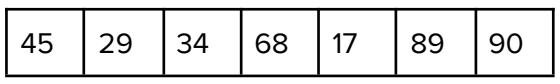
(c) Realice la traza para el arreglo:

arreglo:

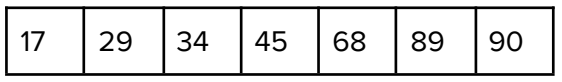
89	45	68	90	29	34	17

MODULO burbuja(arreglo)

i	j	longitud	arreglo
			89 45 68 90 29 34 17
		7	
0			
	0		
			45 45 68 90 29 34 17 45 89 68 90 29 34 17
	1		
			45 68 68 90 29 34 17 45 68 89 90 29 34 17
	2		
	3		
			45 68 89 29 29 34 17 45 68 89 29 90 34 17
	4		
			45 68 89 29 34 34 17 45 68 89 29 34 90 17
	5		
			45 68 89 29 34 17 17

			
	6		
1			
	0		
	1		
	2		
			 
	3		
			 
	4		
			 
	5		
2			
	0		
	1		
			 
	2		
			 
	3		

			45 29 34 17 17 89 90 45 29 34 17 68 89 90
	4		
3			
	0		
			29 29 34 17 68 89 90 29 45 34 17 68 89 90
	1		
			29 34 34 17 68 89 90 29 34 45 17 68 89 90
	2		
			29 34 17 17 68 89 90 29 34 17 45 68 89 90
	3		
4			
	0		
	1		
			29 17 17 45 68 89 90 29 17 34 45 68 89 90
	2		
5			
	0		
			17 17 34 45 68 89 90 17 29 34 45 68 89 90

	1		
6			
	0		
7			
			

(d) Una variante del algoritmo sería aquel que realiza el ordenamiento de mayor a menor. Realice una implementación de esta variante

Pseudocódigo

```
MODULO burbujaInvertida (ENTERO[] arreglo) RETORNA Ø
        ENTERO i, j, longitud, aux
        longitud = LONGITUD(arreglo)
        PARA i← 0 HASTA longitud -1 PASO 1 HACER
                PARA j ← HASTA longitud - i - 2 PASO 1 HACER
                        SI arreglo[j] < arreglo[j+1] ENTONCES
                                aux ← arreglo[j]
                                arreglo[j] ← arreglo[j+1]
                                arreglo[j+1] ← aux
                        FIN SI
                FIN PARA
        FIN PARA
FIN MODULO
```

Java

```
  public static void burbujaInvertida (int[] arreglo){
      int i,j, longitud, aux;
      longitud = arreglo.length;
      for (i= 0; i < longitud; i++) {
         for(j=0; j<(longitud-i-1); j++){
            if(arreglo[j] < arreglo[j+1]){
               aux = arreglo[j];
               arreglo [j] = arreglo[j+1];
               arreglo [j+1] = aux;
            }
         }
      }
  }
```

(e) Calcule el Tiempo de Ejecución y Orden.

```
1     public static void burbuja (int[] arreglo){
2         int i,j, longitud, aux;
3         longitud = arreglo.length;
4         for (i= 0; i < longitud; i++) {
5             for(j=0; j<(longitud-i-1); j++){
6                 if(arreglo[j]>arreglo[j+1]){
7                     aux = arreglo[j];
8                     arreglo [j] = arreglo[j+1];
9                     arreglo [j+1] = aux;
10                }
11            }
12        }
13    }
```

```
1     public static void burbuja (int[] arreglo){
2         int i,j, longitud, aux;
3         longitud = arreglo.length;
4         for (i= 0; i < longitud; i++) {
5             for(j=0; j<(longitud-i-1); j++){
6                 if(arreglo[j]>arreglo[j+1]){
7                     aux = arreglo[j];
8                     arreglo [j] = arreglo[j+1];
9                     arreglo [j+1] = aux;
10                }
11            }
12        }
13    }
```

$T_n = T_3 + T_4 \qquad T_3 = 2$

$T_4 = 1 + \sum_{i=0}^{n} (1 + T_5 + 2) + 1$

$T_5 = 1 + \sum_{j=0}^{n-i-1} (3 + T_6 + 2) + 1$

$T_6 = 4 + T_7 + T_8 + T_9 = 4+2+4+3 = 13$

$T_7 = 2 \quad T_8 = 4 \quad T_9 = 3$

$T_5 = 2 + \sum_{j=0}^{n-1-i} (3+13+2) = 2 + \sum_{j=0}^{n-1-i} 18 = 2 + 18(n-1-i) = 18n - 18i - 16$

$T_4 = 2 + \sum_{i=0}^{n} (3 + 18n - 18i - 16) = 2 - 13n + 18n^2 - 18 \sum_{i=0}^{n} i =$

$= 18n^2 - 13n + 2 - 18\left(\frac{n^2+n}{2}\right) = 18n^2 - 13n + 2 - 9n^2 - 9n = 9n^2 - 22n + 2$

$T_n = 2 + 9n^2 - 22n + 2 = 9n^2 - 22n + 4$

f(n) = $9n^2 - 22n + 4$

Complejidad cuadrática

O(n^2)

5. Implemente el algoritmo de Ordenamiento por Inserción.

(a) ¿A qué clase pertenece y por qué razón?
Pertenece a la clase Fuerza Bruta ya que recorre todo ordenando los elementos tantas veces como sea necesario para que quede ordenado

(b) Realice el algoritmo en pseudocódigo y java

Pseudocódigo

```
MODULO insercion(ENTERO[] arr) RETORNA Ø
        ENTERO i, j, aux
        ENTERO n ← LONGITUD(arr) - 1
        PARA i ←1 HASTA n PASO 1 HACER
                aux ← arr[j]
                j←i
                MIENTRAS (j > 0 AND arr[j-1] > aux ) HACER
                        arr[j] ← arr[j-1]
                        j ← j-1
                FIN MIENTRAS
                arr[j] ← aux
        FIN PARA
FIN MODULO
```

Java

```
public static void insercion(int[] arr) {
        int i, j, aux;
        int n = arr.length - 1;

        for (i = 1; i <= n; i++) {
            aux = arr[i];
            j = i;
            while (j > 0 && arr[j - 1] > aux) {
                arr[j] = arr[j - 1];
                j = j - 1;
            }
            arr[j] = aux;
        }
    }
```

(c) Realice la traza para el arreglo:

arr:

89	45	68	90	29	34	17

MODULO insercion(arr)

i	j	aux	longitud	arr
				89 45 68 90 29 34 17
			7	
1				
		45		
	1			89 89 68 90 29 34 17
	0			45 89 68 90 29 34 17
2				
		68		
	2			45 89 89 90 29 34 17
	1			45 68 89 90 29 34 17
3				
		90		
	3			45 68 89 90 29 34 17
4				
		29		
	4			45 68 89 90 90 34 17
	3			45 68 89 89 90 34 17
	2			45 68 68 89 90 34 17
	1			45 45 68 89 90 34 17
	0			29 45 68 89 90 34 17
5				
		34		
	5			29 45 68 89 90 90 17
	4			

				29 | 45 | 68 | 89 | 89 | 90 | 17
	3			29 | 45 | 68 | 68 | 89 | 90 | 17
	2			29 | 45 | 45 | 68 | 89 | 90 | 17
	1			29 | 34 | 45 | 68 | 89 | 90 | 17
6				
		17		
	6			29 | 34 | 45 | 68 | 89 | 90 | 90
	5			29 | 34 | 45 | 68 | 89 | 89 | 90
	4			29 | 34 | 45 | 68 | 68 | 89 | 90
	3			29 | 34 | 45 | 45 | 68 | 89 | 90
	2			29 | 34 | 34 | 45 | 68 | 89 | 90
	1			29 | 29 | 34 | 45 | 68 | 89 | 90
	0			17 | 29 | 34 | 45 | 68 | 89 | 90

(d) Una variante del algoritmo sería aquel que realiza el ordenamiento de mayor a menor. Realice una implementación de esta variante.

Pseudocódigo

```
MODULO insercion(ENTERO[] arr) RETORNA Ø
        ENTERO i, j, aux
        ENTERO n ← LONGITUD(arr) - 1
        PARA i ←1 HASTA n PASO 1 HACER
                aux ← arr[j]
                j←i
                MIENTRAS (j > 0 AND arr[j-1] < aux ) HACER
                        arr[j] ← arr[j-1]
                        j ← j-1
                FIN MIENTRAS
                arr[j] ← aux
        FIN PARA
FIN MODULO
```

Java

```java
public static void insercion(int[] arr) {
        int i, j, aux;
        int n = arr.length - 1;

        for (i = 1; i <= n; i++) {
            aux = arr[i];
            j = i;
            while (j > 0 && arr[j - 1] < aux) {
                arr[j] = arr[j - 1];
                j = j - 1;
            }
            arr[j] = aux;
        }
    }
```

(e) Calcule el Tiempo de Ejecución y Orden.

```
1 public static void insercion(int[] arr) {
2         int i, j, aux;
3         int n = arr.length - 1;
4
5         for (i = 1; i <= n; i++) {
6             aux = arr[i];
7             j = i;
8             while (j > 0 && arr[j - 1] > aux) {
9                 arr[j] = arr[j - 1];
10                j = j - 1;
11            }
12            arr[j] = aux;
13        }
14    }
```

$T_N = T_3 + T_5$

$T_3 = 3$

$T_5 = 1 + \sum_{i=1}^{N} (T_6 + T_7 + T_8 + T_{12} + 2 + 2) + 2$

$T_6 = 2 \quad T_7 = 1$

$T_8 = N \cdot (5 + (T_9 + T_{10})) + 5$

$T_9 = 4 \quad T_{10} = 2 \quad T_{12} = 2$

$T_8 = N \cdot (5 + (4 + 2)) + 5 = N \cdot (5 + 6) + 5 = 16N$

$T_5 = 3 + \sum_{i=1}^{N} (2 + 1 + 16N + 2 + 4) = 3 + \sum_{i=1}^{N} (16N + 9) = 3 + N \cdot (16N + 9) = 16N^2 + 9N + 3$

$T_N = 16N^2 + 9N + 3 + 3 = 16N^2 + 9N + 6$

f(n) = $16n^2 + 9n + 3$

Complejidad cuadrática.

O(n^2)

6. **Implemente el algoritmo de Ordenamiento Burbuja Mejorado.**
   (a) ¿A qué clase pertenece y por qué razón?
   Pertenece a la clase fuerza bruta, ya que recorre el arreglo las veces que sea necesario para dejar los elementos más grandes al final
   (b) Realice el algoritmo en pseudocódigo y java

   Pseudocódigo

```
MODULO burbujaMejorado(ENTERO[] arr) RETORNA Ø
        LOGICO ordenado ← falso
        ENTERO i ← 0, j, aux
        ENTERO n ← LONGITUD(arr)

        MIENTRAS i < n-1 AND no ordenado HACER
        ordenado ← verdadero //considero que esta ordenado
                PARA j ← 0 HASTA n-i-2 PASO 1 HACER
                        SI a[j] > a[j+1] ENTONCES
                                ordenado ← falso
                                aux = arr[j];
                                arr [j] = arr[j+1];
                                arr [j+1] = aux;
                        FIN SI
                FIN PARA
                i ← i + 1
        FIN MIENTRAS
FIN MODULO
```

   Java

```
public static void burbujaMejorada(int[] arr) {
        boolean ordenado = false;
        int i = 0, j, aux;
        int k = arr.length;
        int n = k-1-i;

        while (i < k - 1 && !ordenado) {
            ordenado = true;
            for (j = 0; j < n; j++) {
                if (arr[j] > arr[j + 1]) {
                    ordenado = false;
                    aux = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = aux;
                }
            }
            i++;
        }
    }
```

   (c) Realice la traza para el arreglo:

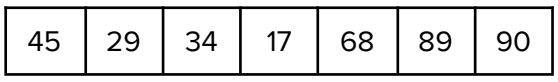
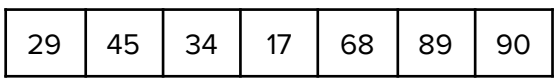
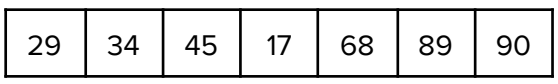
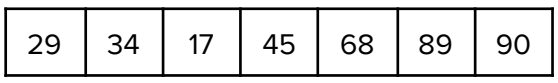
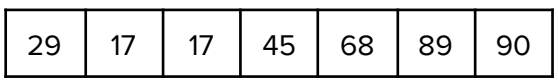
arr:

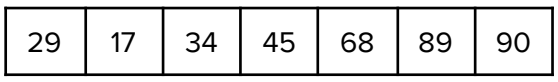
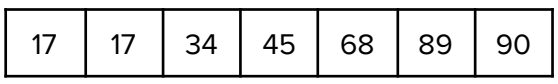
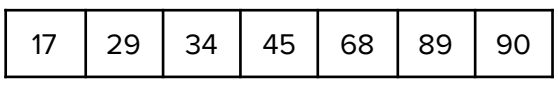
89	45	68	90	29	34	17

ordenado	i	j	aux	n	arreglo
false	0			8	
				7	
true					
		0			
false					
			89		
					45 45 68 90 29 34 17
					45 89 68 90 29 34 17

		1			
			89		
					45 68 68 90 29 34 17
					45 68 89 90 29 34 17
		2			
		3			
			90		
					45 68 89 29 29 34 17
					45 68 89 29 90 34 17
		4			
			90		
					45 68 89 29 34 34 17
					45 68 89 29 34 90 17
		5			
			90		
					45 68 89 29 34 17 17
					45 68 89 29 34 17 90
		6			
	1				
true					
		0			
		1			
		2			
false					
			89		
					45 68 29 29 34 17 90

					45 | 68 | 29 | 89 | 34 | 17 | 90
		3			
			89		
					45 | 68 | 29 | 34 | 34 | 17 | 90
					45 | 68 | 29 | 34 | 89 | 17 | 90
		4			
			89		
					45 | 68 | 29 | 34 | 17 | 17 | 90
					45 | 68 | 29 | 34 | 17 | 89 | 90
		5			
	2				
true					
		0			
		1			
false					
			68		
					45 | 29 | 29 | 34 | 17 | 89 | 90
					45 | 29 | 68 | 34 | 17 | 89 | 90
		2			
			68		
					45 | 29 | 34 | 34 | 17 | 89 | 90
					45 | 29 | 34 | 68 | 17 | 89 | 90
		3			
			68		

					
					
		4			
	3				
true					
		0			
false					
			45		
					
					
		1			
			45		
					
					
		2			
			45		
					
					
		3			
	4				
true					
		0			
		1			
false					
			34		
					

					
		2			
	5				
true					
		0			
false					
			29		
					
					
		1			
	6				

(d) Una variante del algoritmo sería aquel que realiza el ordenamiento de mayor a menor (con la mejora). Realice una implementación de esta variante.

Java

```java
public static void burbujaMejoradaInvertida(int[] arr) {
        boolean ordenado = false;
        int i = 0, j, aux;
        int k = arr.length;
        int n = k-1-i;

        while (i < k - 1 && !ordenado) {
            ordenado = true;
            for (j = 0; j < n; j++) {
                if (arr[j] < arr[j + 1]) {
                    ordenado = false;
                    aux = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = aux;
                }
            }
            i++;
        }
    }
}
```

(e) Calcule el Tiempo de Ejecución y Orden.

```
1  public static void burbujaMejorada(int[] arr) {
2          boolean ordenado = false;
3          int i = 0, j, aux;
4          int k = arr.length;
5          int n = k-1-i;
6
7          while (i < k - 1 && !ordenado) {
8              ordenado = true;
9              for (j = 0; j < n; j++) {
10                 if (arr[j] > arr[j + 1]) {
11                     ordenado = false;
12                     aux = arr[j];
13                     arr[j] = arr[j + 1];
14                     arr[j + 1] = aux;
15                 }
16             }
17             i++;
18         }
19     }
```

$$T_N = T_3 + T_4 + T_5 + T_7$$

$$T_3 = 1 \quad T_4 = 2 \quad T_5 = 3$$

$$T_7 = N \cdot (4 + (T_8 + T_9 + T_{17})) + 4$$

$$T_8 = 1 \qquad T_{17} = 2$$

$$T_9 = 1 + \sum_{J=1}^{N-1} (T_{10} + 1 + 2) + 1$$

$$T_{10} = 4 + T_{11} + T_{12} + T_{13} + T_{14}$$

$$T_{10} = 4 + 1 + 2 + 4 + 3 = 14$$

$$T_{11} = 1 \quad T_{12} = 2 \quad T_{13} = 4 \quad T_{14} = 3$$

$$T_9 = 2 + \sum_{J=1}^{N-1} (14 + 1 + 2) = 2 + \sum_{J=1}^{N-1} 17 = 2 + 17 \cdot (N-1) = 2 + 17N - 17 = 17N - 15$$

$$T_7 = N(4 + (1 + 17N - 15 + 2)) + 4 = 17N^2 - 8N + 4$$

$$T_N = 1 + 2 + 3 + 17N^2 - 8N + 4 = 17N^2 - 8N + 10$$

f(n) = $17n^2 - 8n + 10$

Complejidad cuadrática.

O(n^2)