

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

PRÁCTICA I: ORDENAMIENTO POR INSERCIÓN Y POR SELECCIÓN

ELÍAS LÓPEZ RIVERA

2CV5

ALGORITMOS Y ESTRUCTURAS DE DATOS

MAESTRO: MANUEL PORTILLO

Fecha de entrega: 07/04/2025



Ordenamiento Burbuja

A. Algoritmo

Bubble sort

***Paso 1:** Iteramos por todas las posiciones del arreglo, con la variable i*

***Paso 2:** Por cada iteración volvemos a iterar sobre todo el arreglo excepto la última posición, con la variable j*

***Paso 3:** Si la posición j y $j + 1$ no están ordenadas correctamente, intercambiamos sus valores*

B. Pseudocódigo

Algorithm 1 Bubble Sort

Require: $[10000, 2, 400, \dots, n]$

Ensure: $[1, 2, 3, \dots, n]$

INICIO

Desde $i = 1$ hasta $i < tamarray$

Desde $j = 0$ hasta $j < tamarray - 1$

Si $array[j] < array[j + 1]$

$AUX = array[j]$

$array[j] = array[j + 1]$

$array[j + 1] = AUX$

$j = j + 1$

$i = i + 1$

FIN

C. Código fuente en C

```

1 //Bubble sort
2 #include <stdio.h>
3
4 void swap(char*a,char*b)/*Funcion que intercambia
5 los valores de dos variables char a nivel de memoria*/
6 {
7     char aux=*a;
8     *a=*b;
9     *b=aux;
10 }
11
12 void imprimirarreglo(char A[5])/*imprime un arreglo de
13 caracteres a traves de un bucle for*/
14 {
15     for (int l = 0; l < 5; l++)
16     {
17         printf("%c ",*(A+l));
18     }
19     printf("\n");
20 }
21
22 void bubblesort(char A[5])/*Ordenamiento burbuja*/
23 {
24     for(int i=0;i<5;i++)/*Iteramos sobre todas las posiciones
25 del arreglo*/
26     {
27         for(int l=0;l<4;l++)/*Iteramos las
28 posciones del arreglo excepto la n-1*/
29         {
30             if(*(A+l)<*(A+l+1))/*si los elementos
31 no se encuentran ordenaos*/
32             {
33                 swap((A+l),(A+l+1));/*Los intercambiamos*/
34             }
35         }
36     }
37 }
38
39
40 int main()
41 {
42     char A[5]={ 'D', 'L', 'A', 'Z', 'W' };//inicializamos el arreglo
43     imprimirarreglo(A);
44     bubblesort(A);
45     imprimirarreglo(A);
46 }

```

[Link copiar código](#)

D. Compilación

```
PS C:\Users\Elías López\downloads\programacion\ejercicios-ets> gcc bubble_sort.c -o ejecutable.exe
PS C:\Users\Elías López\downloads\programacion\ejercicios-ets> ./ejecutable.exe
D L A Z W
Z W L D A
PS C:\Users\Elías López\downloads\programacion\ejercicios-ets> █
```

E. Complejidad

Como siempre estamos haciendo dos iteraciones completas sobre el arreglo la complejidad es $O(n^2)$

Ordenamiento por Mezcla

A. Algoritmo

Merge sort

Paso 1: Dividimos en mitades el arreglo principal hasta obtener subarreglos de dos o un indices.

Paso 2: Ordenamos cada uno de estos subaareglos simples y posteriormente los mezclamos (unimos) las respectivas mitades **Paso 3:** Ordenamos el arreglo que resulto de unir las dos mitades

Paso 4: Repetimos este proceso hasta obtener nuestro arreglo original ordenado

B. Pseudocodigo

Algorithm 2 Merge Sort

Require: $[10000, 2, 400, \dots, n]$
Ensure: $[1, 2, 3, \dots, n]$
INICIO
funcion mezcla ($\text{arr}[], p, q, r$)

 $n1 = q - p + 1$
 $n2 = r - q$
 $\text{char } L[n1], M[n2]$
Desde $j=0$ **hasta** $j < n1$
 $L[j] = \text{arr}[p + j]$
Desde $i=0$ **hasta** $i < n2$
 $M[i] = \text{arr}[q + 1 + i]$
 $i = 0, j = 0, k = p$
Mientras $i < n1 \ \&\& \ j < n2$
Si $L[i] \geq M[j]$
 $\text{arr}[k] = L[i]$
 $i++$
Si no
 $\text{arr}[k] = M[j]$
 $j++$
 $k++$
Mientras $i < n1$
 $\text{arr}[k] = L[i]$
 $i++$
Mientras $j < n2$
 $\text{arr}[k] = M[j]$
 $i++$
 $k++$
funcion sort ($\text{arr}[], l, r$)

Si $l < r$
 $m = (l + (r - l)) / 2$
 $\text{sort}(\text{arr}, l, m)$
 $\text{sort}(\text{arr}, m + 1, r)$
 $\text{merge}(\text{arr}, l, m, r)$
FIN

C. Código fuente

```
1
2 #include <stdio.h>
3
4 // Merge two subarrays L and M into arr
5 void merge(char arr[], int p, int q, int r)
6 {
7
8     // separamos la cantidad de elementos que tendra cada subarreglo
9     int n1 = q - p + 1;
10    int n2 = r - q;
11
12    char L[n1], M[n2];
13    //Copiamos la informacion a cada subarreglo
14    for (int i = 0; i < n1; i++)
15        L[i] = arr[p + i];
16    for (int j = 0; j < n2; j++)
17        M[j] = arr[q + 1 + j];
18    // Creamos indices para iniciar la mezcla de los arreglos
19    int i, j, k;
20    i = 0;
21    j = 0;
22    k = p;
23
24
25    /*Mezclamos los arreglos y los reescribimos en el
26    arreglo original*/
27    while (i < n1 && j < n2) {
28        if (L[i] >= M[j])
29            /*Vemos que posocion del arreglo es la correcta respecto al orden
30            y la copiamos en el arreglo original*/
31            {
32                arr[k] = L[i];
33                i++;
34            }
35        else
36        {
37            arr[k] = M[j];
38            j++;
39        }
40        k++;
41    }
42
43    /*finalmente copiamos los arreglos restantes
44    en el original*/
45    while (i < n1) {
46        arr[k] = L[i];
47        i++;
48        k++;
49    }
50
51    while (j < n2) {
52        arr[k] = M[j];
```

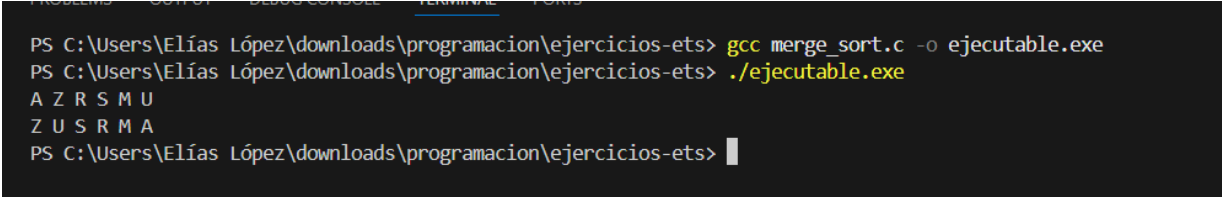
```

53     j++;
54     k++;
55 }
56 }
57
58 /*Divide el arreglo recursivamente el arreglo en mitades
59 y las ordena*/
60 void mergeSort(char arr[], int l, int r) {
61     if (l < r)/*condicion de paro
62     para la recursi n, indica si aun es posible
63     dividir el subarreglo en mitades*/
64     {
65
66         //m es el punto medio donde se dividira el arreglo
67         int m = l + (r - l) / 2;
68
69         mergeSort(arr, l, m);//generaamos 2 subarreglos
70         mergeSort(arr, m + 1, r);
71
72         //genera y ordena(Mezcla) los subarreglos
73         merge(arr, l, m, r);
74     }
75 }
76
77 // Imprime el arreglo con un ciclo for
78 void imprimirarreglo(char arr[], int size)
79 {
80     for (int i = 0; i < size; i++)
81         printf("%c ", arr[i]);
82     printf("\n");
83 }
84
85 // Implementacion
86 int main()
87 {
88     //inicializamos el arreglo char
89     char arr[] = {'A', 'Z', 'R', 'S', 'M', 'U'};
90     //encontramos la dimensiones del arreglo
91     int size = sizeof(arr) / sizeof(arr[0]);
92     imprimirarreglo(arr, size);
93     mergeSort(arr, 0, size - 1);
94     imprimirarreglo(arr, size);
95 }

```

[Link copiar codigo](#)

D. Compilación



```
PS C:\Users\Elías López\downloads\programacion\ejercicios-ets> gcc merge_sort.c -o ejecutable.exe
PS C:\Users\Elías López\downloads\programacion\ejercicios-ets> ./ejecutable.exe
A Z R S M U
Z U S R M A
PS C:\Users\Elías López\downloads\programacion\ejercicios-ets> |
```

E. Complejidad

Al reducir las operaciones a la mitad por cada iteración durante la recursión (partir a la mitad los arreglos) tenemos una complejidad $O(\log(n))$, sin embargo al mezclar los arreglos hacemos n iteraciones en el peor caso por tanto la complejidad total es de $O(n \log(n))$