Ordenamiento de raíz (radix sort).

Este ordenamiento se basa en los valores de los dígitos reales en las representaciones de posiciones de los números que se ordenan.

Por ejemplo el número 235 se escribe 2 en la posición de centenas, un 3 en la posición de decenas y un 5 en la posición de unidades.

Reglas para ordenar.

- Empezar en el dígito más significativo y avanzar por los dígitos menos significativos mientras coinciden los dígitos correspondientes en los dos números.

- El número con el dígito más grande en la primera posición en la cual los dígitos de los dos números no coinciden es el mayor de los dos (por supuesto sí coinciden todos los dígitos de ambos números, son iguales).

Este mismo principio se toma para Radix Sort, para visualizar esto mejor tenemos el siguiente ejemplo. En el ejemplo anterior se ordeno de izquierda a derecha. Ahora vamos a ordenar de derecha a izquierda.

Archivo original.

25 57 48 37 12 92 86 33

Asignamos colas basadas en el dígito menos significativo.

Parte delantera Parte posterior

0

1

2 12 92

3 33

4

5 25

6 86

7 57 37

8 48

9

10

Después de la primera pasada:

12 92 33 25 86 57 37 48

Colas basadas en el dígito más significativo.

Parte delantera Parte posterior

0

1 12

2 25

3 33 37

4 48

5 57

6

7

8 86

9 92

10

Archivo ordenado: 12 25 33 37 48 57 86 92

A S O R T I N G E X A M P L E

A E O L M I N G E A X T P R S

A E A E G I N M L O

A A E E G

A A

A A

E E G

E E

I N M L O

L M N O

L M

N O

S T P R X

S R P T

P R S

R S

```
#include
#include
#define NUMELTS 20

void radixsort(int x[], int n)
{
  int front[10], rear[10];
  struct {
    int info;
    int next;
  } node[NUMELTS];
  int exp, first, i, j, k, p, q, y;
```

```
/* Inicializar una lista viculada */
for (i = 0; i < n-1; i++) {
  node[i].info = x[i];
  node[i].next = i+1;
} /* fin del for */
node[n-1].info = x[n-1];
node[n-1].next = -1;
first = 0; /*first es la cabeza de la lista vinculada */
for (k = 1; k < 5; k++) {
  /* Suponer que tenemos n£meros de cuatro dÁgitos */
  for (i = 0; i < 10; i++) {
    /*Inicializar colas */
    rear[i] = -1;
    front[i] = -1;
  } /*fin del for */
  /* Procesar cada elemento en la lista */
  while (first != -1) {
    p = first;
    first = node[first].next;
    y = node[p].info;
    /* Extraer el kâsimo dÁgito */
    exp = pow(10, k-1);     /* elevar 10 a la (k-1)âsima potencia */
    j = (y/exp) % 10;
    /* Insertar y en queue[j] */
    q = rear[j];
    if (q == -1)
          front[j] = p;
    else
          node[q].next = p;
    rear[j] = p;
  } /*fin del while */

  /* En este punto, cada registro est  en su cola bas ndose en el dÁgito k
     Ahora formar una lista £nica de todos los elementos de la cola. Encontrar
     el primer elemento. */
  for (j = 0; j < 10 && front[j] == -1; j++);
    ;
  first = front[j];

  /* Vincular las colas restantes */
  while (j <= 9) {              /*Verificar si se ha terminado */
    /*Encontrar el elemento siguiente */
    for (i = j+1; i < 10 && front[i] == -1; i++);
          ;
    if (i <= 9) {
          p = i;
```

```
            node[rear[j]].next = front[i];
       } /* fin del if */
       j = i;
     } /* fin del while */
     node[rear[p]].next = -1;
   } /* fin del for */

   /* Copiar de regreso al archivo original */
   for (i = 0; i < n; i++) {
     x[i] = node[first].info;
     first = node[first].next;
   } /*fin del for */
} /* fin de radixsort*/


int main(void)
{
   int x[50] = {NULL}, i;
   static int n;

   printf("\nCadena de n£meros enteros: \n");
   for (n = 0;; n++)
     if (!scanf("%d", &x[n])) break;
   if (n)
     radixsort (x, n);
   for (i = 0; i < n; i++)
     printf("%d ", x[i]);
   return 0;
}
```

Estado de la lista

i	Node[i].info	Node[i].next			
		Inicialización	K = 1	K = 2	K = 3
0	65	1	3	1	2
1	789	2	-1	-1	-1
2	123	3	0	3	3
3	457	-1	1	0	1

rear = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1}

2 0 3 1

front = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1}

2 0 3 1

k = 1

p = 0 p = 1 p = 2 p = 3

first = 1 first = 2 first = 3 first = -1

y = 65 y = 789 y = 123 y = 457

exp = 1 exp = 1 exp = 1 exp = 1

j = 5 j = 9 j = 3 j = 7

q = -1 q = -1 q = -1 q = -1

si q == -1 si q == -1 si q == -1 si q == -1

front[5] = 0 front[9] = 1 front[3] = 2 front[7] = 3

rear[5] = 0 rear[9] = 1 rear[3] = 2 rear[7] = 3

```
j = 3
first = 2
while ( j <= 9)
        i = 5
        si i <= 9
                p = 5
                 node[2].next = 0
                j = 5
        i = 7
        si i <= 9
                p = 7
                node[0].next = 3
                j = 5
        i = 9
```

```
            si i <= 9
                        p = 9
                         node[2].next = 1
                        j = 9
    fin del while
    p = 9
    node[1].next = -1
```

Características.

- Debido a que el ciclo for (*k* = 1; *k* <= *m*; *k*++) externo se recorre *m* veces (una para cada dígito) y el ciclo interior *n* veces (una para cada elemento en el archivo) el ordenamiento es de aproximadamente (*m***n*).

- Si las llaves son complejas (es decir, si casi cada número que puede ser una llave lo es en realidad) *m* se aproxima a log *n*, por lo que (*m***n*) se aproxima a (*n* log *n*).

- Si la cantidad de dígitos es grande, en ocasiones es más eficiente ordenar el archivo aplicando primero el ordenamiento de raíz a los dígitos más significativos y después utilizando inserción directa sobre el archivo ordenado.

Ventajas.

- **El ordenamiento es razonablemente eficiente si el número de dígitos en las llaves no es demasiado grande.**

- **Si las máquinas tienen la ventaja de ordenar los dígitos (sobre todo si están en binario) lo ejecutarían con mucho mayor rapidez de lo que ejecutan una comparación de dos llaves completas.**