Algoritmos y Estructuras de Datos

Clasificación u Ordenación
Parte II

1

Clasificación

- Introducción.
  - Existe un orden lineal definido para los elementos del conjunto a clasificar, por ejemplo, "menor que".
  - La clasificación puede dividirse en interna y externa.
  - Estabilidad del método (capacidad de mantener el orden relativo de elementos con iguales claves).
  - Los algoritmos más simples requieren tiempos de $O(n^2)$, otros $O(n*\log n)$ y algunos, para clases especiales de datos, $O(n)$.
  - Los objetos a clasificar son estructuras complejas y contienen al menos un elemento del tipo para el cual se define la relación de ordenación (clave).

2

Clasificación

- Clasificación interna.
  - Ordenar un conjunto de elementos de forma tal que los valores de sus claves formen una secuencia no decreciente.
  - Usar con cuidado el almacenamiento disponible.
  - Medida de eficiencia: contar número de comparaciones de claves ***C*** y de movimientos de elementos ***M***.
  - Los buenos algoritmos de clasificación requieren $O(n*\log n)$ comparaciones, los más sencillos, $O(n^2)$
  - Métodos directos e indirectos.
    - Los directos son más cortos y fáciles de entender
    - Las operaciones de los indirectos son más complejas, por lo que los métodos directos pueden ser más rápidos para pequeños conjuntos de datos

3

Clasificación interna

- Podemos agrupar los métodos de la siguiente forma:
  - Clasificación por inserción
  - Clasificación por intercambio.
  - **Clasificación por selección**
  - **Clasificación por enumeración o cuenta.**
  - **Especiales: ej.: Clasificación por urnas y por residuos**

Algoritmos y Estructuras de Datos 4

4

Clasificación

- Ejemplos de métodos de clasificación por inserción.
  - Inserción directa.
    - Orden del tiempo de ejecución N^2 en el peor caso, orden N en el mejor.
    - Preferido para tamaños pequeños.
  - Shellsort o clasificación por disminución de incrementos.
    - Preferido para tamaños medianos.
    - Mejor caso N, peor caso $N^{1,26}$, cercano a NlogN.

Algoritmos y Estructuras de Datos 5

5

Clasificación

- Ejemplos de métodos de clasificación por intercambio.
  - Intercambio directo o "burbuja".
    - Función del tiempo de ejecución con constantes bajas.
    - Orden del tiempo de ejecución N^2.
    - Tal vez el peor método de clasificación.
  - Quicksort.
    - El método de menor tiempo de ejecución en el mejor caso.
    - Mejor caso NlogN, peor caso N^2.

Algoritmos y Estructuras de Datos 6

6

Clasificación

- Ejemplos de métodos de clasificación por selección.
  - Selección directa.
    - Orden del tiempo de ejecución N^2 en todos los casos.
    - N^2 comparaciones, pero exactamente N movimientos.
  - Heapsort o clasificación por montículo.
    - Orden NlogN en todos los casos.

7

Clasificación

- Ejemplos de métodos de distribución
  - Bucketsort
    - Orden "cuasi lineal"
  - Binsort (clasificación por urnas)
    - Sin claves repetidas.
    - Con claves repetidas: el rango de las claves es menor a la cantidad de ellas.
    - Orden N
  - Radix sort o clasificación por residuos.
    - Orden N.
    - Se descompone la clave sub tipos.
  - Cuentas por distribución
    - Orden N.

8

Clasificación

- Ordenar en forma no decreciente un vector con "N" elementos que están desde la posición 1 a la N.
- Cada elemento posee un atributo de datos y un atributo "clave" por el cual se ordenará el vector:
  - V[i] es el elemento de la posición "i" del vector V.
  - V[i].datos es el atributo de datos del elemento.
  - V[i].clave es la clave por la que se ordenará el vector.

9

Clasificación por Selección

- Se basa en los siguientes pasos:
  - Hallar la menor clave del conjunto; transferir el registro correspondiente al área de salida y sustituir la clave por el valor infinito.
  - Repetir el paso anterior; esta vez se seleccionará la segunda menor clave.
  - Continuar repitiendo el primer paso hasta que se hayan seleccionado la totalidad de los registros.
- El método **requiere** que estén presentes todos los elementos antes de que se pueda empezar la clasificación y genera la salida final uno a uno en secuencia.
- Se puede usar para la salida la misma área de memoria del conjunto original de datos, moviendo el elemento a su posición definitiva.

Algoritmos y Estructuras de Datos 10

10

Clasificación por Selección

- Selección directa
  - Arroja un orden del tiempo de ejecución $O(N^2)$ en todos los casos.
  - El orden está dado por las comparaciones, ya que realiza siempre exactamente N-1 intercambios.
- Heapsort
  - La idea es mejorar las comparaciones para obtener el menor de los elementos.
  - Se obtiene un O(N log N) en todos los casos.

Algoritmos y Estructuras de Datos 11

11

	1	2	3	4	5	6	7	8
iter	223	784	376	285	015	440	666	007
1	007	784	376	285	015	440	666	223
2	007	015	376	285	784	440	666	223
3	007	015	223	285	784	440	666	376

12

Algoritmo de selección directa

```
(1)  Desde i = 1 hasta N - 1 hacer
(2)     IndiceDelMenor ← i
(3)     ClaveMenor ← V[i].clave
(4)     Desde j = i + 1 hasta N hacer
(5)          Si V[j].clave < ClaveMenor entonces
(6)              IndiceDelMenor ← j
(7)              ClaveMenor ← V[j].clave
(8)          Fin si
(9)      Fin desde
(10)     intercambia (V[i], V[IndiceDelMenor])
     Fin desde
```


13

Selección directa: análisis del orden del tiempo de ejecución

```
(1)  Desde i = 1 hasta N - 1 hacer
(2)     IndiceDelMenor ← i
(3)     ClaveMenor ← V[i].clave
(4)     Desde j = i + 1 hasta N hacer
(5)          Si V[j].clave < ClaveMenor entonces
(6)              IndiceDelMenor ← j
(7)              ClaveMenor ← V[j].clave
(8)          Fin si
(9)      Fin desde
(10)     intercambia (V[i], V[IndiceDelMenor])
     Fin desde
```

- Las sentencias de las líneas 2,3,5,6,7 y 10 son todas de O(1).
- El bloque de sentencias que abarca la sentencia 4, se ejecuta exactamente N-i veces.
- Ese bloque, más la sentencia de intercambio, se ejecutan exactamente N-1 veces.
- Por lo tanto, este método es $O(N^2)$, con la particularidad que los intercambios son siempre N-1.

14

	1	2	3	4	5	6	7	8
iter	223	784	376	285	015	440	666	007
1	007	784	376	285	015	440	666	223
2	007	015	376	285	784	440	666	223
3	007	015	223	285	784	440	666	376
4	007	015	223	285	784	440	666	376
5	007	015	223	285	376	440	666	784
6	007	015	223	285	376	440	666	784
7	007	015	223	285	376	440	666	784
8	007	015	223	285	376	440	666	784

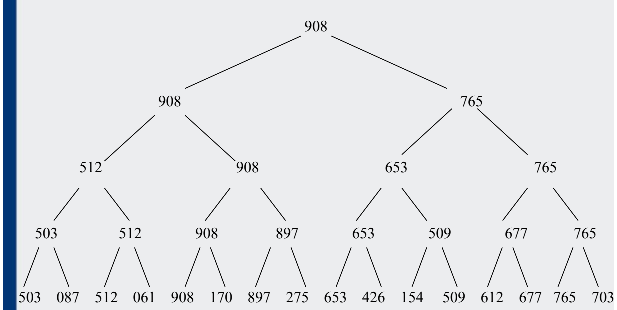
15

Selección: mejora al método directo

- Para encontrar el menor elemento de un conjunto no hace falta compararlo con cada uno de los otros.
- Se puede optimizar usando otro modelo de selección.
- Por ejemplo: “torneo de liga” vs “torneo de copa”. En la liga, juegan todos contra todos, en la copa, se van eliminando por parejas.
- Considerar el siguiente árbol binario, en el que el elemento mayor “sube” hasta la raíz con una cantidad de comparaciones proporcional al logaritmo de la cantidad de nodos.

16

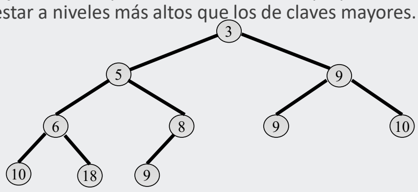
Clasificación : Selección en árbol



17

Arboles parcialmente ordenados.

- Es un ***árbol binario completo***, completado por niveles. En el nivel más bajo, si no está completo, las hojas faltantes serán del extremo derecho, es decir que el nivel se completa de izquierda a derecha
- La clave de un nodo cualquiera **v** no es mayor que la de sus hijos. Nótese que los nodos con claves pequeñas no necesitan estar a niveles más altos que los de claves mayores.



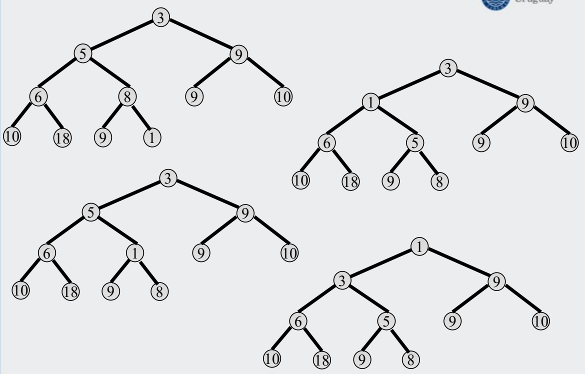
18

Arboles parcialmente ordenados

- Consideramos dos operadores: **SuprimeMinimo** e **Inserta**.
- **Inserta**.
  - Se coloca el nuevo elemento lo más a la izquierda posible en el nivel más bajo, creando un nuevo nivel si éste ya se encuentra completo.
  - Si el nuevo elemento tiene una clave menor que la de su padre, se intercambia con éste.
  - Este proceso de comparación e intercambio se repite hasta que el elemento llegue a la raíz o alcance una posición en la cual tenga clave mayor que la de su padre.

19

Ej: insertar nodo con clave “1”



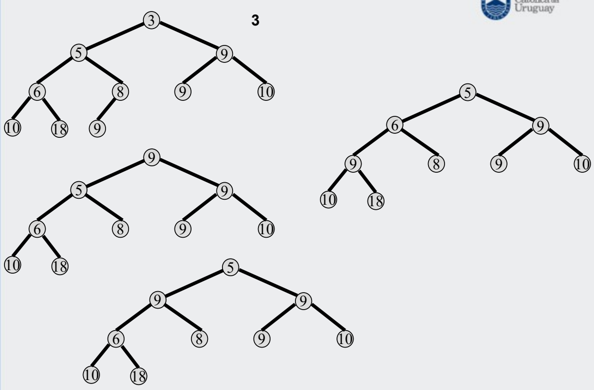
20

Arboles parcialmente ordenados

- **SuprimeMinimo**.
  - Se **devuelve el elemento con menor clave**, que se encuentra en la **raíz**. Se debe ahora **arreglar el árbol** para que se siga cumpliendo la propiedad de árbol parcialmente ordenado.
  - Para ello se **toma la** hoja de más a la derecha del nivel más bajo y se coloca en **la raíz**.
  - Luego se lleva este elemento **lo más abajo posible**, **intercambiándolo** con el hijo que tenga la prioridad más baja, hasta que el elemento se encuentre en una hoja o en una posición en la cual **las claves de los hijos sean iguales o mayores.**

21

ej: suprimeMinimo



22

Clasificación

x elemento
E conjunto de entrada
S estructura con Inserta y Suprime / Min
y elemento

```
1)  Para x en E
2)    INSERTA(x,S);
3)  Mientras no VACIA(S)
4)    y := MIN(S)
5)    procesar(y)
6)    SUPRIME(y,S)
    Fin mientras
```


23

Clasificación

- Heapsort. El peor caso y el caso promedio son O(n log n).
  - El algoritmo se puede representar en forma abstracta por medio de las cuatro operaciones **INSERTA**, **SUPRIME**, **MIN** y **VACIA**.
  - **E** es el conjunto de elementos a clasificar y **S** un conjunto de elementos que se usa para guardar los ya clasificados, el heap auxiliar para llevar a cabo la ordenación.

```
1)  Para x en E
2)    INSERTA(x,S);
3)  Mientras no VACIA(S)
4)    y := MIN(S)
5)    procesar(y)
6)    SUPRIME(y,S)
    Fin mientras
```


24

Clasificación

- En un ***heap*** las operaciones INSERTA y SUPRIME son de orden logarítmico (recorren una altura del árbol balanceado), y la operación MIN es de orden constante.

```
1)  Para x en E
2)    INSERTA(x,S);
3)  Mientras no VACIA(S)
4)    y := MIN(S)
5)    procesar(y)
6)    SUPRIME(y,S)
    Fin mientras
```

La sentencia (1) repite n veces una sentencia de orden log de n

La sentencia (3) repite n veces un bloque de orden log de n, suponiendo que "procesar" sea O(1), por ejemplo insertar al final de una lista

Por lo tanto, el algoritmo es de orden

N * LogN

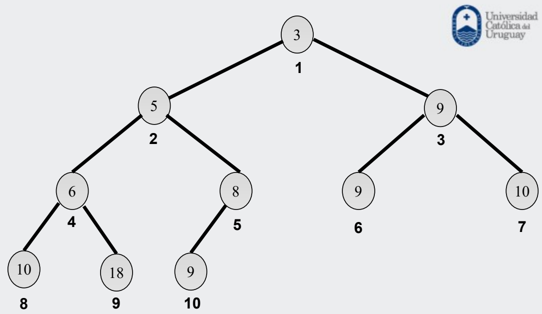
Arboles parcialmente ordenados – representación por vector

- Una estructura adecuada para representar un APO es un vector.
- En la primera posición del vector se ubica la **raíz** del árbol, y para cada nodo que se encuentre en la posición ***i***, sus hijos estarán en las posiciones ***2*i*** y ***2*i+1***.
- La **clave dominante** (más grande o más chica, de acuerdo al criterio) está en la **raíz**.
- De esta forma, los elementos que están en las posiciones de la **1** a la **(N div 2)** son *nodos internos*, los siguientes son *hojas*.

Representación de los árboles parcialmente ordenados mediante arreglos.

- **Montículo** (***heap***).
- Si hay **n** nodos, se utilizan las primeras **n** posiciones de un arreglo V. V[1] contiene la raíz. El hijo izquierdo del nodo V[i], si existe, se encuentra en V[2i], y el hijo derecho, si existe, en V[2i+1].
- Los nodos del árbol llenan V[1], V[2],..., V[n] nivel a nivel, desde arriba, y de izquierda a derecha dentro de cada nivel. El árbol de ejemplo se puede almacenar en un arreglo así:

3, 5, 9, 6, 8, 9, 10, 10, 18, 9



1	2	3	4	5	6	7	8	9	10
3	5	9	6	8	9	10	10	18	9

28

Heapsort

1	2	3	4	5	6	7	8	9	10
3	5	9	6	8	9	10	10	18	9

- El conjunto S siempre estará almacenado como un ***heap*** en la parte inferior del arreglo V, como V[1],.. V[i] si S tiene i elementos.
- El elemento más pequeño siempre estará en V[1].

29

Heapsort

- Los elementos que se van eliminando de S se pueden almacenar en V[i+1],.., V[n], clasificados en orden inverso, es decir, V[i+1] >= V[i+2] >=... V[n].
- La operación **SuprimeMinimo** puede realizarse entonces:
  - intercambiando V[1] con V[i].
  - si el nuevo V[1] viola la propiedad del árbol parcialmente ordenado, debe descender en el árbol hasta su lugar, para lo cual se usa el procedimiento ***DesplazaElemento***.

1	2	3	4	5	6	7	8	9	10
3	5	9	6	8	9	10	10	18	9

30

Heapsort

- Los elementos que se van eliminando de S se pueden almacenar en V[i+1],.., V[n], clasificados en orden inverso, es decir, V[i+1] >= V[i+2] >=... V[n].
- La operación **SuprimeMinimo** puede realizarse entonces:
  – intercambiando V[1] con V[i].
  – si el nuevo V[1] viola la propiedad del árbol parcialmente ordenado, debe descender en el árbol hasta su lugar, para lo cual se usa el procedimiento ***DesplazaElemento***.

1	2	3	4	5	6	7	8	9	10
3	5	9	6	8	9	10	10	18	9

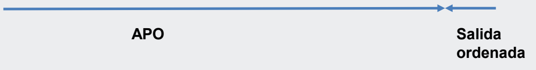
1	2	3	4	5	6	7	8	9	10
9	5	9	6	8	9	10	10	18	3

31

Heapsort

i

1	2	3	4	5	6	7	8	9	10
5	6	9	9	8	9	10	10	18	3



32

Heapsort: Método de desplazar un elemento

```
DesplazaElemento(Primero, Ultimo: tipo entero);
Comienzo
      Actual ← Primero;
      mientras Actual <= (Ultimo div 2) hacer
            Si ultimo = 2*Actual  entonces                    //Actual tiene un hijo
                  Si  V[Actual].clave  > V[2*Actual].clave  entonces
                    Intercambia(V[Actual], V[2*Actual])
                       fin si
                  Actual ← Ultimo
            Sino                                              // Actual tiene dos hijos
                  Menor ← MenorHijo(2*Actual, 2*Actual+1)     //indice del menor
                       Si  V[Actual].clave  > V[Menor].clave entonces
                             Intercambia(V[Actual], V[Menor])
                             Actual ← Menor
                       Sino  Actual ← Ultimo
                       Fin si
            Fin si
      Fin mientras
Fin
```


33

Heapsort – armar el heap

- En este ejemplo estamos partiendo de un heap ya armado en el vector.
- Pero el vector inicialmente está desordenado, el heap hay que armarlo.
- Para ello se usará también la operación "DesplazaElemento".
- Sea el vector inicial:

1	2	3	4	5	6	7	8	9	10
5	6	10	18	8	9	9	10	3	9

Algoritmos y Estructuras de Datos 34

34

Heapsort – armar el heap

1	2	3	4	5	6	7	8	9	10
5	6	10	18	8	9	9	10	3	9

nodos internos — hojas

- Se observa que todos los elementos de la posición 6 en adelante son hojas.
- Por lo tanto basta con desplazar el elemento de la posición 5 al lugar que le corresponde, luego 4, y así sucesivamente hasta llegar al 1.
- Esta operación es más eficiente que si se fueran insertando de a uno los elementos en un heap vacío.

Algoritmos y Estructuras de Datos 35

35

Heapsort – armar el heap

1	2	3	4	5	6	7	8	9	10
5	6	10	18	8	9	9	10	3	9
5	6	10	18	8	9	9	10	3	9
5	6	10	3	8	9	9	10	18	9
5	6	9	3	8	9	10	10	18	9
5	3	9	6	8	9	10	10	18	9
3	5	9	6	8	9	10	10	18	9

Algoritmos y Estructuras de Datos 36

36

Algoritmo de Heapsort

```
Comienzo
   Desde i = N div 2 hasta 1 hacer
      DesplazaElemento(i, N);
   Fin desde

   Desde i = N hasta 2 hacer
      Intercambia(V[1], V[i])
      DesplazaElemento(1, i-1);
   Fin desde
Fin
```

Arma un árbol parcialmente ordenado a partir de un vector desordenado V

Para cada valor de i, quita la raíz y la intercambia con esa posición i, que es al final de la lista de salida. Luego desplaza el elemento de la raíz hasta el lugar que le corresponde. En cada paso, el árbol va de la posición 1 hasta la i-1; de la i hasta la N está la secuencia de salida ordenada.

37

Algoritmo de Heapsort: análisis del método

```
Comienzo
   Desde i = N div 2 hasta 1 hacer
      DesplazaElemento(i, N);           O(logN)   } O(N*logN)
   Fin desde                                                  +
   Desde i = N hasta 2 hacer
      Intercambia(V[1], V[i])           O(1)      } O(N*logN)
      DesplazaElemento(1, i-1)          O(logN)
   Fin desde
Fin
                                                  ____________
                                                   O(N*logN)
```


38

Heapsort

- Ordenar 44, 28, 15, 45, 74, 23, 17, 59

1	2	3	4	5	6	7	8	i
44	28	15	45	74	23	17	59	4
44	28	15	45	74	23	17	59	3
44	28	15	45	74	23	17	59	2
15	28	17	45	74	23	44	59	1
17	28	23	45	74	59	44	15	8
23	28	44	45	74	59	17	15	7
28	45	44	59	74	23	17	15	6
44	45	74	59	28	23	17	15	5
45	59	74	44	28	23	17	15	4
59	74	45	44	28	23	17	15	3
74	59	45	44	28	23	17	15	2

39

Cuenta por distribución.

- Aplicable en caso de que existan claves iguales, y cuando están en el ámbito **u<=K_j<=v**, donde (**v-u**) es pequeño.
- Supongamos que todas las claves están entre 1 y 100.
- En la primer pasada contamos cuántas ocurrencias de cada clave hay, en la segunda pasada movemos los elementos al lugar apropiado en el área de salida.
- todas las claves iguales para **u<= K_j <= v**
- clasifica los registros **R_1,..,R_N**, usando tabla auxiliar **Cuenta[u], ..., Cuenta[v].**
- Al final del algoritmo los elementos se mueven al área de salida **S_1,..., S_N**, en el orden deseado.

40

Cuenta por distribución.

```
CuentaPorDistribución
begin
 for i = u to v hacer Cuenta[i] = 0
 for j = 1 to N incrementar Cuenta[Kj]
 for i = u +1 to v
    Cuenta[i] = Cuenta[i] + Cuenta[i-1]
 for j = N downto 1 hacer
     i = Cuenta[Kj]
     S[i ] = R[j ]
     Cuenta[Kj] = i-1
end
```

Ejercicio:

Simular el algoritmo de Cuenta para el conjunto de 8 elementos:

2, 5, 3, 0, 2, 3, 0, 3

41

R	1	2	3	4	5	6	7	8
	2_1	5	3_1	0_1	2_2	3_2	0_2	3_3

u = 0 v = 5

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr>
<tr><td>Cuenta</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td></td><td>||</td><td></td><td>||</td><td>|||</td><td></td><td>|</td></tr>
<tr><td></td><td>2</td><td>0</td><td>2</td><td>3</td><td>0</td><td>1</td></tr>
<tr><td></td><td>2</td><td>2</td><td>4</td><td>7</td><td>7</td><td>8</td></tr>
<tr><td></td><td>1</td><td></td><td>3</td><td>6</td><td></td><td>7</td></tr>
<tr><td></td><td>0</td><td></td><td>2</td><td>5</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>4</td><td></td><td></td></tr>
</table>

42

```
for j = N downto 1 hacer
   i = Cuenta[Kj]
   S[i ] = R[j ]
   Cuenta[Kj] = i-1
end
```

R	1	2	3	4	5	6	7	8
	2_1	5	3_1	0_1	2_2	3_2	0_2	3_3

Cuenta

0	1	2	3	4	5
2	**2**	**4**	**7**	**7**	**8**
1		3	6		7
0		2	5		
			4		

s	1	2	3	4	5	6	7	8
	0_1	0_2	2_1	2_2	3_1	3_2	3_3	5

j	K_j	i = Cuenta[Kj]	S[i] = R[j]
8	3_3	7 - 6	3_3
7	$\mathbf{0_2}$	2 - 1	$\mathbf{0_2}$
6	$\mathbf{3_2}$	6 - 5	S[6]=R[6]
5			
4			
3			
2			
1			

43

Binsort (clasificación por urnas).

- Los algoritmos de clasificación ya vistos tienen una cota mínima de O(n*log n).
- Si se conoce el rango de los valores de las claves se puede pasar a O(n).
- Ej: Si el tipo de clave es entero con rango 1..n, y existen n claves diferentes, entonces es posible diseñar un algoritmo de clasificación de orden n.
- Ver animación en

https://www.cs.usfca.edu/~galles/visualization/CountingSort.html

44

Binsort (trivial)

– Solución con dos arreglos: el original **A** y otro como área de salida **B**.

```
for i = 1 to n do
   B[A[i]] := A[i];
```

– Solución con una sola área.

```
for i = 1 to n do
   while A[i] <> i do
     intercambia(A[i], A[A[i]]);
```

Ejemplo:

3	1	4	10	5	9	2	6	7	8

45

Binsort

- Problema con los duplicados: en el algoritmo anterior asumimos que no existían.
- Para manejarlos, podemos hacer que el arreglo B contenga cabezales a listas en las cuales se almacenan los registros con claves duplicadas.
- Se pueden usar punteros al último de cada lista, y éste a su vez puede apuntar al primer elemento de la lista siguiente, para poder recorrer todo el conjunto sin tener que utilizar las cabeceras.

46

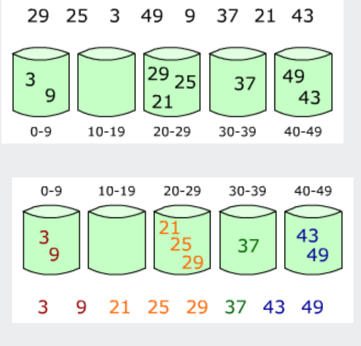
Binsort – algoritmo conceptual

- **BINSORT(*A*) o BUCKETSORT(A)**
- ***n*** *largo del array* [*A*]
- **m** sub-arrays **B**
- **para** *i* de 1 a *n*
  - insertar ***A*[*i*]** en la lista ***B*** *correspondiente*
- **para** *i* = 1 to *m*
  - ordenar lista **B[i]** por insercion (u otro método)
  - concatenar las listas **B[1]**, **B[2]**, . . , **B[m]** en orden

Ver también en:
http://es.wikipedia.org/wiki/Bucket_sort

47

BINSORT



48

Algoritmo de Binsort con listas

```
    Binsort (entrada, m)
        urnas := new array de m listas vacías
(1)     for i = 1 to n
(2)         insertar entrada[i] en urnas[DMS(entrada[i]).clave]
(3)     for i = 0 to m-1
(4)         Ordenar(urnas[i])
        salida := Concatenar(urnas[0]… urnas[m-1])
        devolver salida
```

- Orden:
  - Las sentencias (1) y (2) llevan tienen un tiempo de ejecución de O(n)
  - Las (3) y (4) O(m), donde m es el número de claves diferentes.
  - El total del algoritmo es de O(m+n).
  - Animación en https://www.cs.usfca.edu/~galles/visualization/BucketSort.html

49

Radix o Clasificación por residuos.

- Asumimos un TipoClave constituido por k elementos, $f_1, f_2, .., f_k$, de tipos t_1, $t_2, ..., t_k$.
- Se desea clasificar los registros en orden lexicográfico.
- Ejemplos:

```
type
  TipoClave = record
              dia : 1..31;
              mes: 1..12;
              año: 1900..1999;
  end;
```

```
type
  TipoClave = array[1..10] of char;
```


50

Algoritmo de Radix

- La idea clave para la ordenación por RADIX es
  - hacer una ordenación de todos los registros por BINSORT, primero sobre f_k, el "dígito menos significativo"
  - Luego concatenar las urnas, con el menor valor primero
  - Hacer BINSORT sobre f_{k-1},
  - Y asi sucesivamente para los restantes campos de la clave
- Al insertar en las urnas, asegurarse de hacerlo al final y no al principio de la lista.
- En general, luego de aplicar binsort sobre f_k, f_{k-1},...,f_i, los registros aparecerán en orden lexicográfico si la clave consiste en los campos f_i,...,f_k

51

Algoritmo de Radix

- Se clasifica la lista A de n registros con claves que consisten de k campos. El procedimiento usa k arreglos B_i de tipo array[t_i] of TipoLista.

procedure RadixSort;

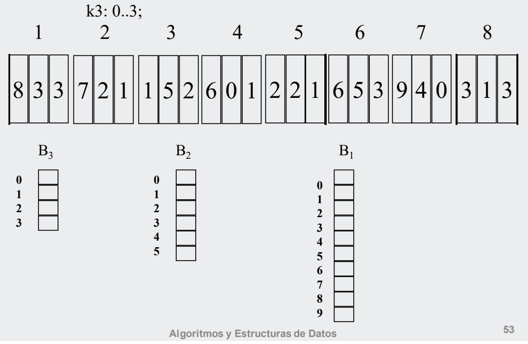
(1) **para** i := k **hasta** 1
(2) **para** cada valor **v** de tipo $\mathbf{t_i}$ **Vaciar**(B_i[v]); {limpiar las urnas}
(3) **para** cada registro **R** de **A**
(4) **mover** R desde **A** hasta el final de la urna B_i[v];
(5) **para** cada valor **v** de tipo $\mathbf{t_i}$, de menor a mayor, **hacer**
(6) **concatena** $\mathbf{B_i[v]}$ en el extremo de **A**.
end;
end;

52

Ejemplo de Radix

TipoClave = record
k1: 0..9;
k2: 0..5;
k3: 0..3;

A : array[1..n] of TipoClave



53

Radix sort

type
TipoClave = record
dia : 1..31;
mes: 1..12;
año: 1990..1999;
end;

- Ordenar historias con las siguientes claves:
  1. **15/12/97**
  2. **2/8/96**
  3. **3/10/99**
  4. **20/4/90**
  5. **23/5/94**
  6. **22/7/96**
  7. **25/4/90**
  8. **18/9/95**

Ver otros ejemplos en
http://en.wikipedia.org/wiki/Radix_sort
http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Sort/Radix/

Animación en
https://www.cs.usfca.edu/~galles/visualization/RadixSort.html

54

Algoritmo de Radix

- Se clasifica la lista A de n registros con claves que consisten de k campos. El procedimiento usa k arreglos B_i de tipo array[t_i] of TipoLista.

```
      procedure RadixSort;

(1)     para i := k hasta 1
(2)       para cada valor v de tipo t_i  Vaciar(B_i[v]);  {limpiar las urnas}
(3)       para cada registro R de A
(4)         mover R desde  A hasta el final de la urna B_i[v];
(5)       para cada valor v de tipo t_i , de menor a mayor,  hacer
(6)         concatena B_i[v] en el extremo de A.
        end;
      end;
```


55

Análisis de algoritmo de Radix.

- El ciclo de la línea (2) tarda un tiempo O(s_i), donde s_i es el número de valores diferentes del tipo t_i.
- El ciclo de las líneas (3) y (4) lleva un tiempo O(n).
- El ciclo de las líneas (5) y (6) lleva un tiempo O(*si*).
- El tiempo total entonces es :

$$\sum_{i=1}^{k} O(s_i + n) = O(k * n + \sum_{i=1}^{k} s_i) = O(n + \sum_{i=1}^{k} s_i)$$

56

Preguntas y Ejercicios

- ¿Qué es la clasificación interna y cuáles son sus objetivos? ¿Cómo se pueden categorizar los distintos métodos de ordenación existentes? Explique las características de cada grupo.
- Defina "ÁRBOL PARCIALMENTE ORDENADO". Ilustre con un ejemplo gráfico
- ¿Cuál es el orden de ejecución del PEOR caso del algoritmo de HEAPSORT? Indique cuál es el caso.
- Dado el siguiente conjunto de datos, proceda a clasificarlo utilizando el algoritmo de HEAPSORT, mostrando la evolución del vector en cada iteración.

22- 11- 44- 55- 88- 77- 33- 01

57