Ordenamiento en tiempo lineal y Estadísticas de orden

Agustín J. González

ELO320: Estructura de Datos y Algoritmos

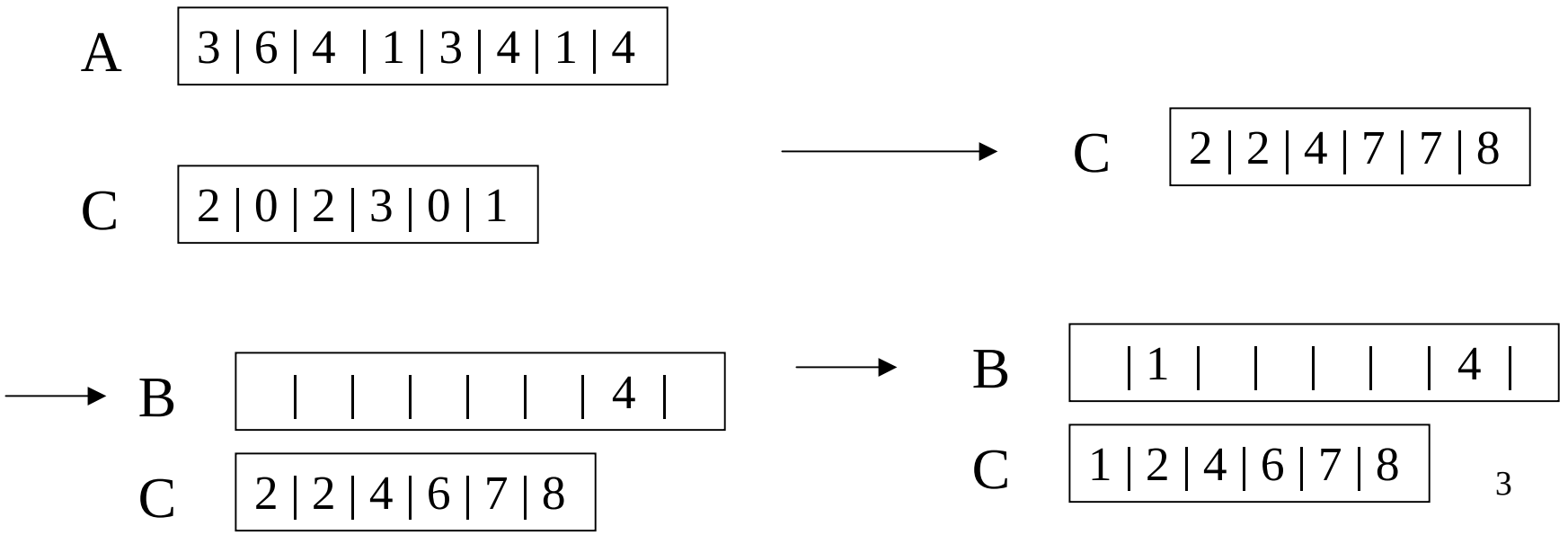
1er. Sem 2002

Idea

- Hasta ahora los algoritmos vistos se basan en la comparación de números para obtener el orden.
- Se puede probar que los algoritmos basados en esta técnica tienen como cota inferior un costo $\Theta(n \lg n)$.
- Exploraremos dos algoritmos: CountingSort y RadixSort.
- Estos algoritmos suponen un rango acotado para la entrada y logran hacer un ordenamiento en tiempo cercano a lineal.

CountingSort

- Asume que cada uno de los n elementos a ordenar es un entero en el rango 1 a k , k ≤ n.
- La idea es determinar, para cada entrada x, el número de elementos menor que x. Así es posible ubicar x directamente en la posición dentro del arreglo.
- Se A[1..n] el arreglo de entrada. CountingSort utiliza un arreglo C[1..k] y genera el resultado en otro arreglo, digamos B[1..n].



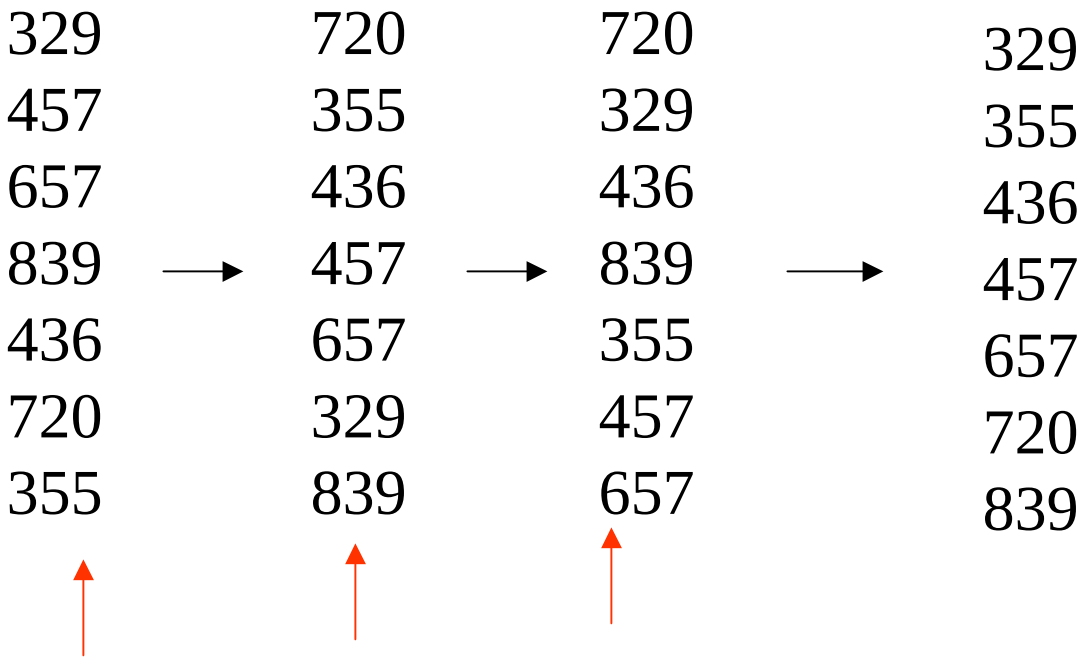
Algoritmo

```
CountingSort(A,B,k) {
        for (i=1;  i< = k;  i++)                                        Θ(k)
                C[i] = 0;
        for (j=1;  j< = Largo(A);  j++)                                 Θ(n)
                C[A[j]]++;
        /* hasta aquí C [i] contiene el número de elementos igual a i*/
         for (i=2;  i< = k;  i++)                                       Θ(k)
                C[i]+=C[i-1];
        /* hasta aquí C [i] contiene el número de elementos menor
           que o igual a i*/
        for ( j= Largo (A); j > 0; j--) {                               Θ(n)
                B[C[A[j]]] = A[j];
                C[A[j]] --;
        }                                                               ---------
}                                                                       Θ(n+k)

                                                                        = Θ(n), k<n
```

Radix Sort

- La idea es ordenar los números dígito por dígito.



- Se ordena desde el menos al más significativo.
- Cada vez se aplica CountingSort.
- Para ordenar d dígitos se toma un tiempo $\Theta(dn+dk)$

Algoritmo Radix sort

- RadixSort(A, d) {
  for (i=1; i <= d; i++)
  use un ordenamiento estable para ordenar arreglo A sobre digito i;
  }

- Un algoritmo de **ordenamiento es estable** si el orden de elementos iguales es preservado.
- Cuando se usa CountingSort el costo en tiempo es $\Theta(dn+dk)$. Si k es acotado y d también, esto conduce a un tiempo $\Theta(n)$.
- Desgraciadamente countingSort requiere espacios de memoria adicionales al requerido para mantener los datos a ordenar. Por ello, si la capacidad de memoria es un factor importante, quicksort es preferible.
- Otra característica interesante de los algoritmos de ordenamiento es si el ordenamiento es el el lugar o requiere memoria adicional. Cómo es heapsort? Quicksort? Insertion sort? Mergesort?

Medianas y Estadísticas de Orden

Agustín J. González

ELO320: Estructura de Datos y Algoritmos

1 er. Sem. 2002

Conceptos

- La estadística de orden i-ésimo de un conjunto de n elementos es el elemento i-ésimo más pequeño.
- El mínimo es la estadística de primer orden.
- La mediana es el punto en la “mitad del camino”
- La mediana se ubicas en $i=\lfloor (n+1)/2 \rfloor$ e $i=\lceil (n+1)/2 \rceil$
- Problema de selección:
  Entrada: un conjunto de n números y un número i $1 <= i <= n$
  Salida: El elemento x tal que es mayor que exactamente otros i-1.

Mínimo y máximo

- Mínimo(A)

```
min = A[1];
for (i=2; i<=length(A); i++)
        if (min > A[i] )
                min = A[i];
```

- Puede ser hecho en tiempo $\Theta(n)$.
- **Problema de selección**: Basta usar una forma adaptada de Randomized_Quicksort. En ésta sólo nos preocupamos de donde se encuentra la estadística del orden que buscamos.
- Randomized_Select(A, p,r,i)

```
if (p==r)
        return A[p];
q = Randomized_Partition(A,p,r);
k = q -p+1;
if (i <= k)
        return Randomized_Select(A,p,q,i);
else
        return Randomized_Select(A, q+1, r, i-k);
```

- El tiempo de este algoritmo es $\Theta(n)$ en promedio y su peor caso es $\Theta(n^2)$.