Escuela de Ingeniería Civil Informática
Facultad de Ingeniería

Estructuras de datos

Capítulo V: Algoritmos de ordenamiento

Fabián Riquelme Csori

fabian.riquelme@uv.cl

2017-II

Index

Problema de ordenamiento

- Sea un subconjunto de n elementos $X = \{e_1, \ldots, e_n\}$ de un conjunto referencial Y, $X \subseteq Y$.
- Para Y se define una relación de orden total $\leq$

Problema de ordenamiento

- Sea un subconjunto de n elementos $X = \{e_1, \ldots, e_n\}$ de un conjunto referencial Y, $X \subseteq Y$.
- Para Y se define una relación de orden total $\leq$
  - reflexiva:
  - anti-simétrica:
  - transitiva:

Problema de ordenamiento

- Sea un subconjunto de n elementos $X = \{e_1, \ldots, e_n\}$ de un conjunto referencial Y, $X \subseteq Y$.
- Para Y se define una relación de orden total $\leq$
  - reflexiva: $\forall e_i \in Y,\ e_i \leq e_i$
  - anti-simétrica:
  - transitiva:

Problema de ordenamiento

- Sea un subconjunto de n elementos $X = \{e_1, \ldots, e_n\}$ de un conjunto referencial Y, $X \subseteq Y$.
- Para Y se define una relación de orden total $\leq$
  - reflexiva: $\forall e_i \in Y,\ e_i \leq e_i$
  - anti-simétrica: $\forall e_i, e_j \in Y,\ e_i \leq e_j \wedge e_j \leq e_i \Rightarrow e_i = e_j$
  - transitiva:

Problema de ordenamiento

- Sea un subconjunto de n elementos $X = \{e_1, \ldots, e_n\}$ de un conjunto referencial Y, $X \subseteq Y$.
- Para Y se define una relación de orden total $\leq$
  - reflexiva: $\forall e_i \in Y,\ e_i \leq e_i$
  - anti-simétrica: $\forall e_i, e_j \in Y,\ e_i \leq e_j \wedge e_j \leq e_i \Rightarrow e_i = e_j$
  - transitiva: $\forall e_i, e_j, e_k \in Y,\ e_i \leq e_j \wedge e_j \leq e_k \Rightarrow e_i \leq e_k$

Problema de ordenamiento

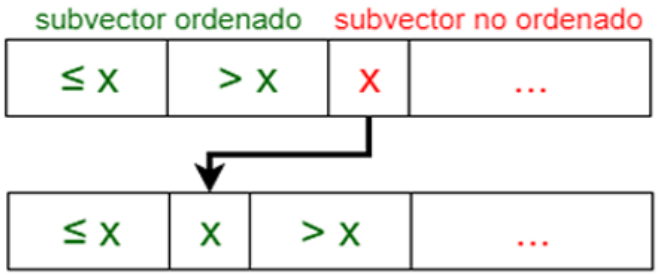
- Sea un subconjunto de n elementos $X = \{e_1, \ldots, e_n\}$ de un conjunto referencial Y, $X \subseteq Y$.
- Para Y se define una relación de orden total $\leq$
  - reflexiva: $\forall e_i \in Y,\ e_i \leq e_i$
  - anti-simétrica: $\forall e_i, e_j \in Y,\ e_i \leq e_j \wedge e_j \leq e_i \Rightarrow e_i = e_j$
  - transitiva: $\forall e_i, e_j, e_k \in Y,\ e_i \leq e_j \wedge e_j \leq e_k \Rightarrow e_i \leq e_k$
- Problema: dado $(X, \leq)$, mediante una serie de permutaciones σ encontrar una secuencia $X < e_{\sigma(1)}, \ldots, e_{\sigma(n)} >$ que verifique $e_{\sigma(1)} \leq \ldots \leq e_{\sigma(n)}$.

Aplicaciones

- Compresión de datos
- Computación gráfica
- Planificación de tareas
- Balanceado de carga en computación paralela
- Bioinformática
- Simulación (sistemas de partículas)
- Gestión de cadenas de suministro
- Sistemas de recomendaciones
- etc. etc. etc.

Ordenamiento por inserción

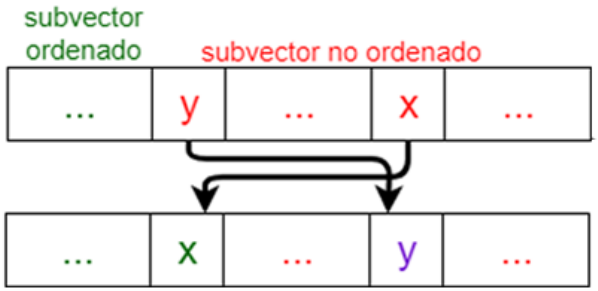
- En cada iteración, se inserta un elemento del subvector no ordenado en la posición correcta dentro del subvector ordenado.



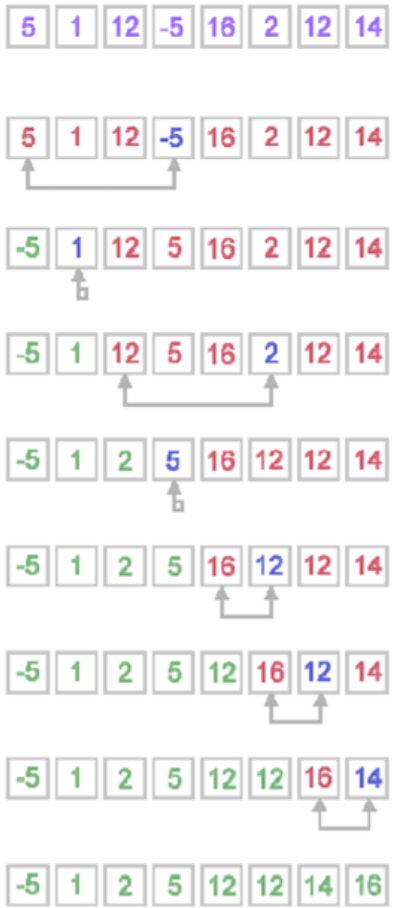

```
void insertionSort (double v[])
{
  double tmp;
  int i, j;
  int N = v.length;
  for (i=1; i<N; i++) {
      tmp = v[i];
      for (j=i; (j>0) && (tmp<v[j-1]); j--)
          v[j] = v[j-1];
      v[j] = tmp;
  }
}
```

Ordenamiento por selección

- En cada iteración, se intercambia el menor elemento del subvector no ordenado con el primer elemento de dicho subvector.




```
void selectionSort (double v[])
{
  double tmp;
  int i, j, pos_min;
  int N = v.length;
  for (i=0; i<N-1; i++) {
      // Menor elemento del vector v[i..N-1]
      pos_min = i;
      for (j=i+1; j<N; j++)
          if (v[j]<v[pos_min])
             pos_min = j;
      // Coloca el mínimo en v[i]
      tmp = v[i];
      v[i] = v[pos_min];
      v[pos_min] = tmp;
  }
}
```



Ordenamiento de burbuja

- En cada iteración se compara cada elemento con el siguiente, intercambiándolos de posición si están en el orden equivocado.

  El proceso se repite tantas veces como sea necesario.

```
void bubbleSort (double v[])
{
  double tmp;
  int i,j;
  int N = v.length;

  for (i = 0; i < N - 1; i++)
      for (j = N - 1; j > i; j--)
          if (v[j] < v[j-1]) {
              tmp = v[j];
              v[j] = v[j-1];
              v[j-1] = tmp;
           }
}
```

Ordenamiento de burbuja (centinela)

- Este algoritmo se puede mejorar para cuando el vector está casi ordenado. Mediante un centinela se termina cuando en una iteración del bucle interno no se produce intercambio.

```
void bubbleSort(double[] v)
{
  boolean swapped = true;
  int i, j = 0;
  double tmp;
  while (swapped) {
        swapped = false;
        j++;
        for (i = 0; i < v.length - j; i++)
            if (v[i] > v[i + 1]) {
               tmp = v[i];
               v[i] = v[i + 1];
               v[i + 1] = tmp;
               swapped = true;
            }
  }
}
```

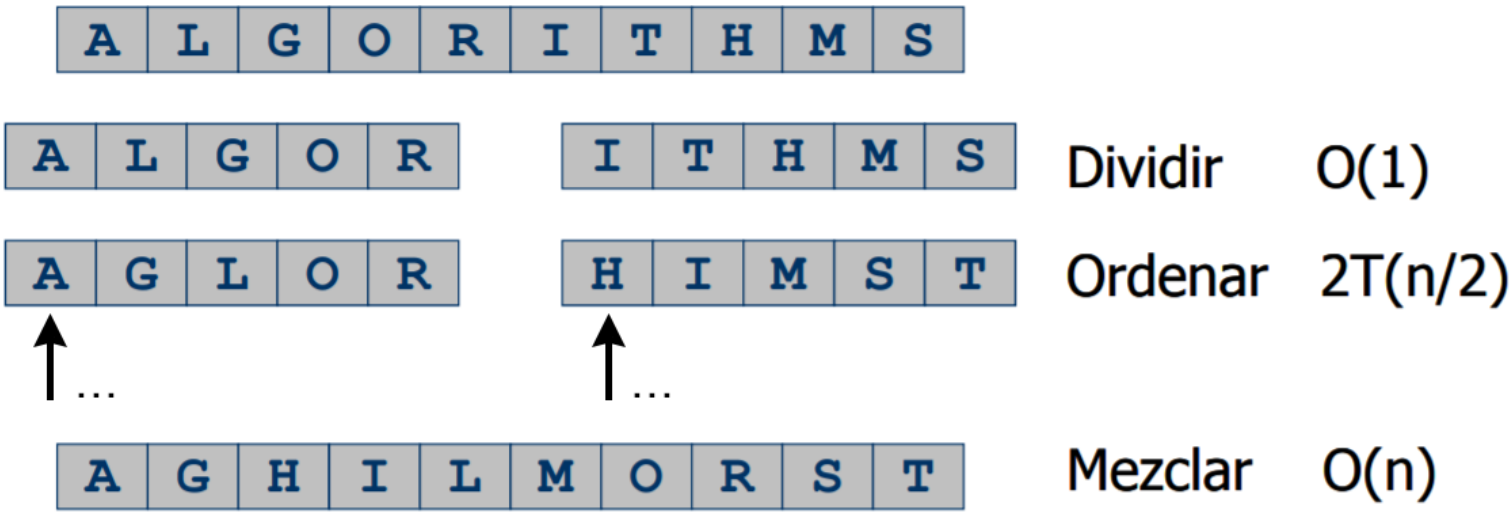
Ejercicios

- Ordenar "en papel" el siguiente vector utilizando los tres métodos anteriores: [8, 5, 2, 6, 9, 3, 1, 4, 0, 7].
- Implementar los pseudocódigos anteriores y comprobar sus resultados del ejercicio anterior.
- Incluir un contador de iteraciones en sus algoritmos y comprobar cuál requiere menos iteraciones de los tres.

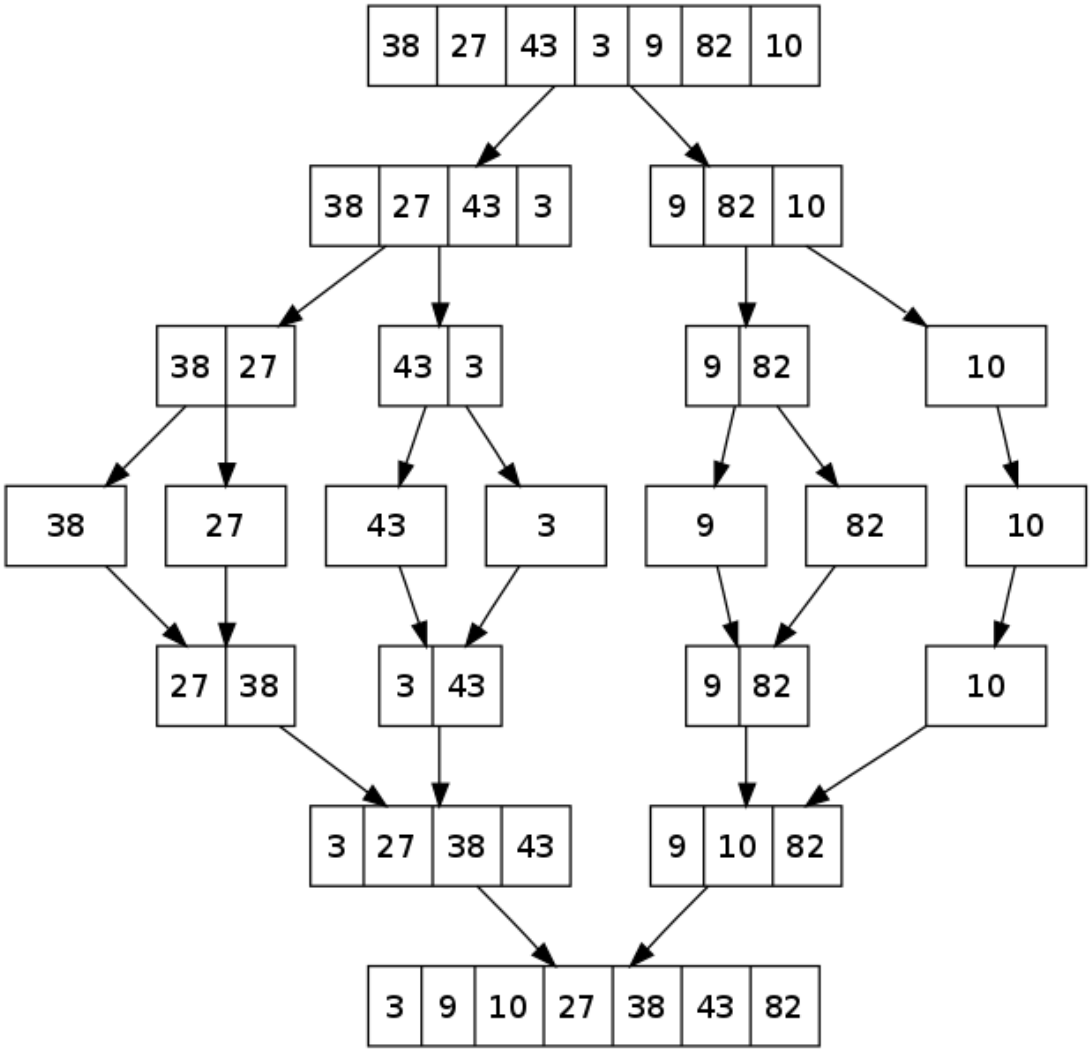
Ordenamiento por mezcla (*Merge sort*)

- Usa la estrategia divide y vencerás. Se divide el vector en dos mitades; se ordena recursivamente cada mitad (usando el mismo método),

  y se van combinando las dos mitades ordenadas.



Ordenamiento por mezcla (*Merge sort*)



Ordenamiento rápido (*Quicksort*)

- Se toma un elemento arbitrario del vector (pivote p). El vector se particiona tal que a izquierda queda todo $\leq p$ y a derecha $\geq p$. Se ordena por separado cada zona delimitada por pivote.

```
void quicksort (double v[], int izda, int dcha)
{
  int pivote;  // Posición del pivote
  if (izda<dcha) {
     pivote = partir (v, izda, dcha);
     quicksort (v, izda, pivote-1);
     quicksort (v, pivote+1, dcha);
  }
}
```

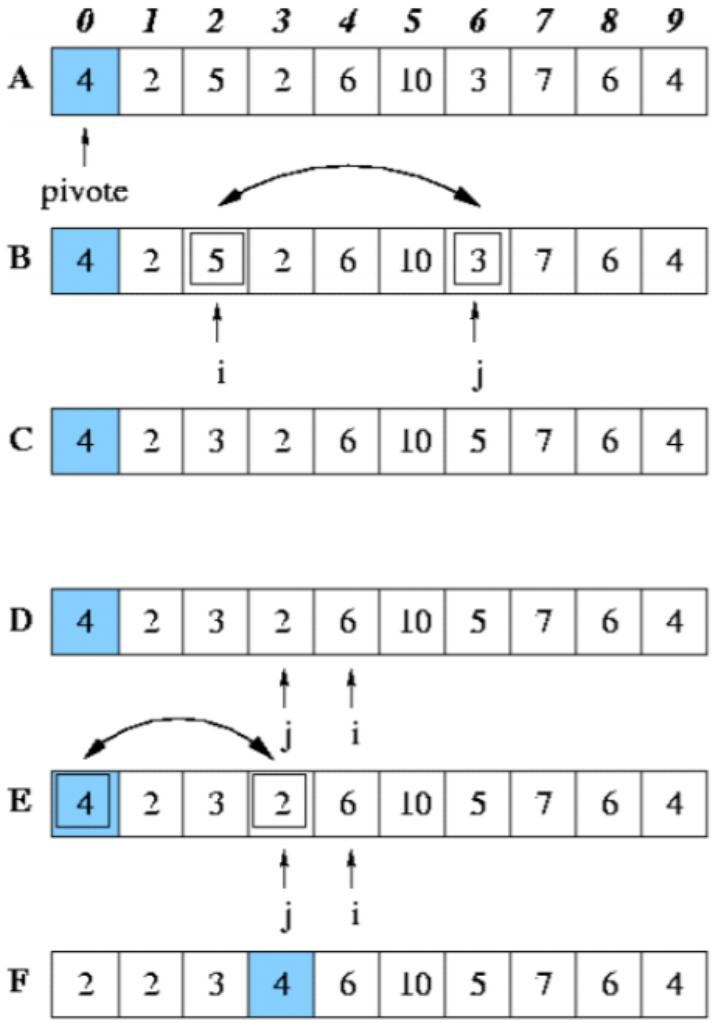
Para ejecutar:

```
quicksort (vector, 0, vector.length-1);
```

Ordenamiento rápido (*Quicksort*)

Para la partición:

- Recorrer el vector de izq a der hasta encontrar un elemento en posición i tal que $v[i] > p$.
- Recorrer el vector de der a izq hasta encontrar un elemento en posición j tal que $v[j] < p$.
- Intercambiar los elementos ubicados en i y j, de modo que $v[i] < p < v[j]$.



Ordenamiento rápido (*Quicksort*)

```
int partir (double v[],
      int primero, int ultimo)
{
  // Valor del pivote
  double pivote = v[primero];

  // Variable auxiliar
  double temporal;

  int izda = primero+1;
  int dcha = ultimo;

  do {  // Pivotear…

    while ((izda<=dcha)
          && (v[izda]<=pivote))
          izda++;
    while ((izda<=dcha)
          && (v[dcha]>pivote))
          dcha--;

    if (izda < dcha) {
       temporal = v[izda];
       v[izda] = v[dcha];
       v[dcha] = temporal;
       dcha--;
       izda++;
    }

  } while (izda <= dcha);

  // Colocar el pivote en su sitio
  temporal = v[primero];
  v[primero] = v[dcha];
  v[dcha] = temporal;

  // Posición del pivote
  return dcha;
}
```

Para conjuntos grandes es conveniente elegir el pivote en el medio.

Mejoras del ordenamiento de inserción

- Heapsort o algoritmo por montículos
  - Basado en la construcción de un árbol parcialmente ordenado (heap).
- Shellsort u ordenamiento de Shell
  - Compara elementos separados por varias posiciones y, en varias en varias pasadas, de saltos cada vez menores, ordena el vector.

Otros algoritmos de ordenamiento

- Bucket sort o Bin sort
- Radix sort
- Histogram sort
- Counting sort, ultra sort o math sort
- Tally sort
- Pigeonhole sort
- Bead sort
- Timsort
- Smoothsort
- Stupid sort
- etc.

Ejercicios

- Ordenar "en papel" el siguiente vector utilizando Merge Sort y Quicksort: [8, 5, 2, 6, 9, 3, 1, 4, 0, 7].
  - Para Quicksort, probar con el pivote en v[0] y v[5].
- Implementar los pseudocódigos anteriores y comprobar sus resultados del ejercicio anterior.
- Incluir un contador de iteraciones en sus algoritmos y comprobar cuál requiere menos iteraciones de los tres.

Comparación de algoritmos

- Comparación de tiempos de ejecución:
  https://www.youtube.com/watch?v=ZZuD6iUe3Pc
- Comparación de complejidad computacional:
  https://en.wikipedia.org/wiki/Sorting_algorithm
  - ¿A qué se refiere la columna "Stable" en las tablas?
  - ¿Cuál es el algoritmo más malo (sin lugar a dudas)?
  - ¿Cuáles son los mejores algoritmos dependiendo de cada criterio?

Bibliografía recomendada

- Weiss, M., Estructura de datos y algoritmos, Addison-Wesley, 1995.
- Aho, Hopcroft y Ullman, Estructuras de datos y algoritmos, Addison-Wesley, 1988.
- National Institute of Standards and Technology, Dictionary of Algorithms and Data Structures `https://xlinux.nist.gov/dads/`

Recursos

- Wikimedia Commons.
- Algoritmos de ordenamiento: Análisis y diseño de algoritmos. DECSAI - Departamento de Ciencias de la Computación e I.A., Universidad de Granada.