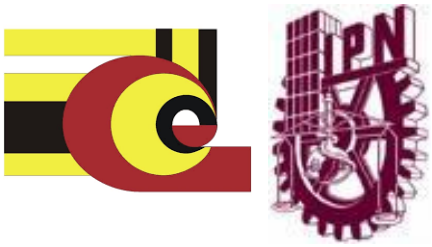
Ordenamiento rápido.



Diplomado de Java - Módulo I

Departamento de Diplomados y Extensión Profesional

Centro de Investigación en Computación

Instituto Politécnico Nacional

Modulo I – Ordenamiento rápido.

Profesor:

Alan Badillo Salas

Integrantes:

Erick Alvarez Barcena

becker7000@outlook.es

Fecha de entrega:

30 de abril de 2022

Algoritmo modificado.

Comenta cada línea el pseudocódigo explicando su funcionamiento en general y en particular.

OrdenamientoRapido.psc

```
1  //Este subproceso intercambia de posición dos valores de una lista.
2  //Se reciben tres parámetros: la lista, posición de un elemento
3  // y posición de otro elemento. No devuelve nada.
4  SubProceso  Intercambiar (lista_entrada, i, j)
5      Definir  temporal Como Entero; //Esta variable es auxiliar
6      //Ayuda a salvar uno de los valores para poder hacer el intercambio.
7      temporal ← lista_entrada[i];
8      //Una vez guardado en temporal el valor de lista_entrada[i] se modifica
9      //lista_entrada[i] asignandole el valor de lista_entrada[j]
10     lista_entrada[i] ← lista_entrada[j];
11     //Se asigna a lista_entrada[j] el valor antiguo de lista_entrada[i]
12     //que está a salvo en temporal.
13     lista_entrada[j] ← temporal;
14 FinSubProceso
15
16 //Esta función se encarga de devolver un indice entre l y r que será
17 //el
18 SUBPROCESO indice_menor ← Particion (lista_entrada, l, r)
19
20     //Se define la variable pivote que será encargada de guardar un
21     //valor de la lista, al inicio el pivote puede ser cualquier elemento.
22     Definir  pivote Como Entero;
23
24     //En este caso el pivote se toma como el último elemento.
25     //para así buscar los elemento que sean menores y luego los mayores a él.
26     //La partición de la lista se hace en función al valor del pivote.
27     pivote ← lista_entrada[r];
28
29     //Se imprime la forma en la que se va a partionar.
30     // l es la primera posición y r la última, pivote
31     // es el valor que está en la ultima posición
32     IMPRIMIR "Particion (" l " " pivote " " r ")";
33
34     //La variabe k será una variable de control para recorre la lista.
35     Definir k Como Entero;
36
37     //Se crea la variable indice_menor que será el indice de la lista
38     //que devuelva el primer elemento de la lista.
39     Definir indice_menor Como Entero;
40
41     indice_menor ← l - 1;
42
43     //Este ciclo incia en k que a su vez guarda el indice inicial
44     //se recorre toda la lista desde l hasta r-1.
45     PARA k ← l HASTA r - 1 CON PASO 1 HACER
```

```
46          SI lista_entrada[k] < pivote ENTONCES //si el elemento en la posición k
47              //es menor al pivote significa que están desordenados y se intercambian de posición.
48              //Estas dos lineas de código sólo tendrán efecto si k e indice_menor se vuelven disparejos.
49              indice_menor ← indice_menor + 1;
50              Intercambiar(lista_entrada, indice_menor, k);
51          FIN SI
52      FIN PARA
53
54      //Estas dos lineas de código están para hacer el intercambio final
55      //en dado caso de que k e indice_menor sean diferentes pero k ya haya superado el
56      //valor de r-1.
57      indice_menor ← indice_menor + 1;
58      Intercambiar(lista_entrada, indice_menor, r);
59
60  FIN SUBPROCESO
61
62  //Este subproceso recibe tres parámetros: la lista de entrada
63  // luego el inidice inicial y el indice final de la misma.
64  SubProceso  Ordenar (lista_entrada, l, r)
65
66      si l < r Entonces //Caso basede la funcíón recursiva.
67
```

```
68          //Se muestra en pantalla de donde a donde se va a ordenar.
69          Imprimir "ORDENAR (" l " " r ")";
70
71          //Se crea una variable para guardar el indice del pivote.
72          //que es el indice apartir del cual se particionará.
73          Definir indice_particion Como Entero;
74
75          //Tal indice lo decide otro subproceso llamado Partición.
76          indice_particion ← Particion(lista_entrada, l, r);
77
78          //Se muestra en pantalla que se hará una partición
79          //donde la variable indice_particion será un valor numérico entre l y r.
80          Imprimir l " " indice_particion " " r;
81          //Luego de obtener el indice de partición se
82          //envian dos lista a ordener en diferentes llamadas
83          //al subpreceso Ordenar.
84
85          //Primero la lista que hay desde 0 hasta un elemento
86          //antes del indice del pivote.
87          Ordenar(lista_entrada, l, indice_particion - 1);
88          //Después la lista que hay desde un elemento después
89          //al pivote hasta el último elemento.
90          Ordenar(lista_entrada, indice_particion + 1, r);
91          //Estas dos ultimas llamadas son llamadas recursivas.
92      FinSi
93
94  FinSubProceso
```

```
//Proceso general
Proceso OrdenamientoRapido

Definir  N Como Entero;; //Definimos una variable para el tamaño de la lista a ordenar.

Imprimir  "Dame el tamaño de la lista a ordenar: "; //Se muestra una instrucción en pantalla.
Leer N; //Se registra el valor de N a través del teclado.

Definir  lista Como Entero; //Se crea una variable del tipo de datos que guardaremos en la lista.
Dimension  lista[N]; //Se crea la lista de tamaño N.

Definir i Como Entero; //Variable de control para recorrer la lista desde 0 hasta N-1.

Para i ← 0 Hasta  N - 1 Con Paso  1 Hacer //Se crea una estructura de control para recorrer la lista.
    Imprimir  "Dame el valor de la lista en la posición ",i,": "; //Se muestra el elemento a registrar.
    Leer  lista[i]; //Se registra el valor del elemento i-ésimo de la lista.
FinPara //Al finalizar este ciclo tenemos la lista llena de elementos.

Ordenar(lista, 0, N - 1); //Se ordena la lista con la función predefinida Ordenar.
// Esta función recibe 3 parámetros: la lista a ordenar,
// la primera pocisión de la lista y la última posición.

//Se vuevle a imprimir en pantalla la lista para mostrar que ya está ordenada en forma creciente.
Para i ← 0 Hasta  N - 1 Con Paso  1 Hacer
    Imprimir "Valor en la posición ",i,": ",lista[i]; //Se formatea la salida para mejor vizualización.
FinPara

FinProceso
```

Algoritmo final

Modifica el pseudocódigo para imprimir la tabla de las listas con las siguientes columnas:

- Los valores de l y r
- La lista antes de la partición de l a r
- El índice de partición
- La lista antes de ordenar de l a indice_particion - 1
- La lista antes de ordenar de indice_particion + 1 a r
- La lista después de ordenar de indice_particion + 1 a r

OrdenamientoRapido.psc

```
1  //Este subproceso intercambia de posición dos valores de una lista.
2  //Se reciben tres parámetros: la lista, posición de un elemento
3  // y posición de otro elemento. No devuelve nada.
4  SubProceso  Intercambiar (lista_entrada, i, j)
5      Definir  temporal Como Entero; //Esta variable es auxiliar
6      //Ayuda a salvar uno de los valores para poder hacer el intercambio.
7      temporal ← lista_entrada[i];
8      //Una vez guardado en temporal el valor de lista_entrada[i] se modifica
9      //lista_entrada[i] asignandole el valor de lista_entrada[j]
10     lista_entrada[i] ← lista_entrada[j];
11     //Se asigna a lista_entrada[j] el valor antiguo de lista_entrada[i]
12     //que está a salvo en temporal.
13     lista_entrada[j] ← temporal;
14 FinSubProceso
15
16 //Esta función se encarga de devolver un indice entre l y r que será
17 //el
18 SUBPROCESO indice_menor ← Particion (lista_entrada, l, r)
19
20     //Se define la variable pivote que será encargada de guardar un
21     //valor de la lista, al inicio el pivote puede ser cualquier elemento.
22     Definir  pivote Como Entero;
23
24     //En este caso el pivote se toma como el último elemento.
25     //para así buscar los elemento que sean menores y luego los mayores a él.
26     //La partición de la lista se hace en función al valor del pivote.
27    pivote ← lista_entrada[r];
28
29    //Se imprime la forma en la que se va a partionar.
30    // l es la primera posición y r la última, pivote
31    // es el valor que está en la ultima posición
32    //IMPRIMIR "Particion (" l " " pivote " " r ")";
33
34    //La variabe k será una variable de control para recorre la lista.
35    Definir k Como Entero;
36
37    //Se crea la variable indice_menor que será el indice de la lista
38    //que devuelva el primer elemento de la lista.
39    Definir indice_menor Como Entero;
```

```
40
41      indice_menor ← l - 1;
42
43      //Este ciclo incia en k que a su vez guarda el indice inicial
44      //se recorre toda la lista desde l hasta r-1.
45      PARA k ← l HASTA r - 1 CON PASO 1 HACER
46          SI lista_entrada[k] < pivote ENTONCES //si el elemento en la posición k
47              //es menor al pivote significa que están desordenados y se intercambian de posición.
48              //Estas dos lineas de código sólo tendrán efecto si k e indice_menor se vuelven disparejos.
49              indice_menor ← indice_menor + 1;
50              Intercambiar(lista_entrada, indice_menor, k);
51          FIN SI
52      FIN PARA
53
54      //Estas dos lineas de código están para hacer el intercambio final
55      //en dado caso de que k e indice_menor sean diferentes pero k ya haya superado el
56      //valor de r-1.
57      indice_menor ← indice_menor + 1;
58      Intercambiar(lista_entrada, indice_menor, r);
59
60  FIN SUBPROCESO
61
62  //Este subproceso recibe tres parámetros: la lista de entrada
63  // luego el inidice inicial y el indice final de la misma.
64  SubProceso  Ordenar (lista_entrada, l, r)
65
```

```
66      si l < r Entonces //Caso basede la función recursiva.
67
68          //Se muestra en pantalla de donde a donde se va a ordenar.
69          //Imprimir "ORDENAR (" l " " r ")";
70
71          //Se crea una variable para guardar el indice del pivote.
72          //que es el indice apartir del cual se particionará.
73          Definir indice_particion Como Entero;
74
75          //Lista antes de la Particion de l a r.
76          Imprimir "Lista antes de la partición de l a r: ";
77          Definir i Como Entero;
78          Para i ← 0 Hasta  r-1 Con Paso  1 Hacer
79              Imprimir "Valor en la posición ",i,": ",lista_entrada[i]; //Se formatea la salida para mejor vizualizac
80          FinPara
81
82          //Tal indice lo decide otro subproceso llamado Partición.
83          indice_particion ← Particion(lista_entrada, l, r);
84          Imprimir "El indice de partición es: ",indice_particion;
85
86          //Se muestra en pantalla que se hará una partición
87          //donde la variable indice_particion será un valor numérico entre l y r.
88          //Imprimir l " " indice_particion " " r;
89          Imprimir  "l: ",l,", r: ",r;
```

```
90
91          Imprimir "Lista antes del ordenamiento de l a indice_partición-1: ";
92          Para i ← 0 Hasta  r-1 Con Paso  1 Hacer
93              Imprimir "Valor en la posición ",i,": ",lista_entrada[i]; //Se formatea la salida para mejor vizualizac
94          FinPara
95
96          //Luego de obtener el indice de partición se
97          //envian dos lista a ordener en diferentes llamadas al subpreceso Ordenar.
98          //Primero la lista que hay desde 0 hasta un elemento antes del indice del pivote.
99          Ordenar(lista_entrada, l, indice_particion - 1);
100         //Después la lista que hay desde un elemento después
101         //al pivote hasta el último elemento.
102         Ordenar(lista_entrada, indice_particion + 1, r);
103         //Estas dos ultimas llamadas son llamadas recursivas.
104
105         Imprimir "Lista después del ordenamiento de l a indice_partición+1: ";
106         Para i ← 0 Hasta  r-1 Con Paso  1 Hacer
107             Imprimir "Valor en la posición ",i,": ",lista_entrada[i]; //Se formatea la salida para mejor vizualizac
108         FinPara
109
110
111     FinSi
112
113 FinSubProceso
114
115 //Proceso general
116 Proceso OrdenamientoRapido
117
118 Definir  N Como Entero;; //Definimos una variable para el tamaño de la lista a ordenar.
119
120 Imprimir  "Dame el tamaño de la lista a ordenar: "; //Se muestra una instrucción en pantalla.
121 Leer N; //Se registra el valor de N a través del teclado.
122
123 Definir  lista Como Entero; //Se crea una variable del tipo de datos que guardaremos en la lista.
124 Dimension  lista[N]; //Se crea la lista de tamaño N.
125
126 Definir i Como Entero; //Variable de control para recorrer la lista desde 0 hasta N-1.
127
128 Para i ← 0 Hasta  N - 1 Con Paso  1 Hacer //Se crea una estructura de control para recorrer la lista.
129     Imprimir  "Dame el valor de la lista en la posición ",i,": "; //Se muestra el elemento a registrar.
130     Leer  lista[i]; //Se registra el valor del elemento i-ésimo de la lista.
131 FinPara //Al finalizar este ciclo tenemos la lista llena de elementos.
132
133 Ordenar(lista, 0, N - 1); //Se ordena la lista con la función predefinida Ordenar.
134 // Esta función recibe 3 parámetros: la lista a ordenar,
135 // la primera pocisión de la lista y la última posición.
136
137 //Se vuevle a imprimir en pantalla la lista para mostrar que ya está ordenada en forma creciente.
138 Para i ← 0 Hasta  N - 1 Con Paso  1 Hacer
139     Imprimir "Valor en la posición ",i,": ",lista[i]; //Se formatea la salida para mejor vizualización.
140 FinPara
141
142 FinProceso
```

[[Capturas de pantalla del funcionamiento]]

```
*** Ejecución Iniciada. ***
Dame el tamaño de la lista a ordenar:
> 5
Dame el valor de la lista en la posición 0:
> 7
Dame el valor de la lista en la posición 1:
> 1
Dame el valor de la lista en la posición 2:
> 6
Dame el valor de la lista en la posición 3:
> 5
Dame el valor de la lista en la posición 4:
> 9
Lista antes de la partición de l a r:
Valor en la posición 0: 7
Valor en la posición 1: 1
Valor en la posición 2: 6
Valor en la posición 3: 5
El indice de partición es: 4
l: 0, r: 4
Lista antes del ordenamiento de l a indice_partición-1:
Valor en la posición 0: 7
Valor en la posición 1: 1
Valor en la posición 2: 6
Valor en la posición 3: 5

Lista antes de la partición de l a r:
Valor en la posición 0: 7
Valor en la posición 1: 1
Valor en la posición 2: 6
El indice de partición es: 1
l: 0, r: 3
Lista antes del ordenamiento de l a indice_partición-1:
Valor en la posición 0: 1
Valor en la posición 1: 5
Valor en la posición 2: 6
Lista antes de la partición de l a r:
Valor en la posición 0: 1
Valor en la posición 1: 5
Valor en la posición 2: 6
El indice de partición es: 3
l: 2, r: 3
```

```
Lista antes del ordenamiento de l a indice_partición-1:
Valor en la posición 0: 1
Valor en la posición 1: 5
Valor en la posición 2: 6
Lista después del ordenamiento de l a indice_partición+1:
Valor en la posición 0: 1
Valor en la posición 1: 5
Valor en la posición 2: 6
Lista después del ordenamiento de l a indice_partición+1:
Valor en la posición 0: 1
Valor en la posición 1: 5
Valor en la posición 2: 6
Lista después del ordenamiento de l a indice_partición+1:
Valor en la posición 0: 1
Valor en la posición 1: 5
Valor en la posición 2: 6
Valor en la posición 3: 7
Valor en la posición 0: 1
Valor en la posición 1: 5
Valor en la posición 2: 6
Valor en la posición 3: 7
Valor en la posición 4: 9
*** Ejecución Finalizada. ***
```

Respuestas a preguntas.

Explica cómo funciona el algoritmo con tus propias palabras. Y responde las siguientes preguntas.

El algoritmo de ordenamiento rápido funciona con base en la técnica divide y vencerás, prácticamente divide en varias listas de números una lista grande para que así se pueda ordenar más rápido, se emplea una variable especial llamada **pivote** que bien puede ser cualquier elemento de la lista pero en caso de que se conozca un número cercano al promedio sería mejor elección como pivote, a partir de que se selecciona un valor de la lista como pivote se separa la lista grande en dos listas una lista izquierda que va a contener los elementos menores al **pivote** y una lista derecha que va a contener elementos mayores al **pivote** este mismo proceso se emplea en las sublistas izquierda y derecha, entonces así, se dice que se es un proceso recursivo, esto se emplea hasta que bajo un caso base la recursividad termina y la lista se ordena por completo.

1. ¿Cómo funciona el subproceso **Intercambiar (lista_entrada, i , j)**?

Este subproceso funciona apoyándose en una variable auxiliar que guarda el valor del elemento contenido en la posición **i** de la **lista(lista_entrada)**, se guarda para que no se pierda, ya que el valor de la posición **i** será reasignado, una vez guardado se puede modificar sin problema el valor en la posición **i**, entonces es aquí donde el valor contenido en la posición **j** se guarda, para después reasignar el valor en la posición **j** con el valor guardado en la variable auxiliar que como sabemos era el valor que anteriormente contenía la posición **i**.

2. ¿Qué representa **i**?
   Representa la una posición de la lista de entrada **(lista_entrada)**, particularmente una posición menor a la posición j, aunque eventualmente puede ser mayor.

3. ¿Qué representa **j**?
   Representa otra posición de la lista de entrada **(lista_entrada)**, diferente a **i**, particularmente es mayor, aunque eventualmente puede ser menor.

4. ¿Qué hace el subproceso sobre **lista_entrada**?
   Intercambiar dos de sus elementos de posición entre sí, al llevarse a cabo este intercambio se habrán quedado ordenados de forma creciente sobre la lista **(lista_entrada)** recorriendo la lista de izquierda a derecha.

5. ¿Cómo funciona el subproceso **indice_menor <- Particion(lista_entrada, l,r)**?
   Prácticamente busca intercambiar el elemento en la posición **l** con el elemento en la posición **r** si y sólo si están en desorden. Ayudándose de la función intercambiar. Al final la función retorna el valor del **indice_menor**, que servirá de guía para partir los siguientes ordenamientos.

6. ¿Qué representa **indice_menor**?
   Es una variable que guarda el índice de la lista que contiene el valor que funcionará como pivote en la siguiente partición.

7. ¿Qué representa **l**?
   Es una variable que guarda el indicie inicial de la lista de entrada a partir del cual se hará la partición.

8. ¿Qué representa **r**?
   Es una variable que representa el tamaño el índice final de la lista de entrada, siempre tiene que ser menor al tamaño de la lista completa.

9. ¿Qué hace el subproceso sobre **lista_entrada**?
   Envía pares de valores la lista de entrada al subproceso **intercambiar** para que los intercambie de posición, este envío lo hace sólo cuando las variables locales **k** e **indice_menor** son diferentes.

10. ¿Cómo funciona el subproceso **Ordenar(lista_entrada,l,r)**?
    En caso de que **l** sea menor que **r** pide a **Particion** le devuelva un índice de partición el cual servirá para dividir a la lista en dos, una lista izquierda y una lista derecha, las listas izquierda y derecha se mandan a ordenar de forma recursiva con éste mismo subproceso, aunque el subproceso sólo tendrá efecto si **l** es menor que **r**.

11. ¿Qué representa el **l**?
    Es una variable que guarda el valor del primer indicie de la lista de entra, la cual se va a separar su ordenamiento en dos sublistas

12. ¿Qué representa **indice_particion**?
    Será un valor entre **r** y **l** en el cuál está el elemento **Pivote** ese elemento no se envía a los siguientes ordenamientos, se omite, por lo tanto se envía la parte a la izquierda del **Pivote** de **l** a **indice_particion-1** y la parte a la derecha de **Pivote** de **indice_particion+1** a **r** a **Ordenar** por separado**.**

13. ¿Qué representa **r**?
    Es una variable que guarda el valor del índice final de la lista de entrada, sirve de guía para no salir del rango indicies posibles.

14. ¿Qué hace el subproceso sobre **lista_entrada**?
    Separar en dos ordenamientos a partir del **Pivote** prácticamente aplicando la técnica divide y vencerás.

Conclusión.

El ordenamiento rápido nos enseña que podemos separar un proceso en varios subprocesos tantos como sean necesarios para poder ejecutarlos más rápido pero esto también es a costa de procesamiento, ya que entre más separaciones mayor será la complejidad del algoritmo. Dado el número de recursiones en algún nivel determinado podríamos tener un volcado de memoria y aunque sea el ordenamiento más rápido que existe podría no ser muy útil al momento de intentar ordenar una lista que tenga más de 100,000 elementos. Una ventaja es que es corto, sencillo y elegante podríamos aplicar esta misma técnica a otros algoritmos que requieran separar varios procesos tan es el caso del algoritmo de las Torres de Hanói donde aplicamos recursividad dentro de un subproceso dos veces por cada recursión, esto va duplicando el número de procesos por cada que entre en una nueva llamada hasta llegar al caso base. Probablemente el algoritmo de ordenamiento rápido es aplicable en calculadoras sencillas donde tenemos que ordenar un conjunto pequeño, por ejemplo, para calcular la mediana de un conjunto de datos de entrada, primero se ordenarían los datos luego se buscaría cual es el elemento de en medio bajo dos condiciones si la cantidad de elementos es par, se suman los dos elementos centrales y se saca un promedio de ellos, entonces tenemos la mediana o si la cantidad de elementos es impar entonces directamente obtenemos el elemento de en medio, este y otros algoritmos pueden hacer uso de un ordenamiento rápido.