



Instituto Politécnico Nacional
Centro de Investigación en Computo



Alumno
RODRÍGUEZ ARRENDONDO JACOBO
jacobo.rodriquez499@gmail.com

Diplomado en lenguaje Java

Facilitador: Alan Badillo Salas

Módulo: 1

Práctica: 2

Fecha de entrega: 25 de abril de 2022

Introducción

En esta práctica se aborda los temas de pseudo códigos, algoritmos de ordenamiento, al igual que la implementación de ciclos lógicos e instancias condicionales para obtener una mejor comprensión de la lógica de programación.

Desarrollo

A partir del siguiente pseudo código para el ordenamiento de datos numéricos se realizó un análisis de su funcionamiento para poder responder las siguientes preguntas.

- 1) Comenta cada línea del pseudo código explicando su funcionamiento en general y en particular

```
1.
2.  SUBPROCESO Intercambiar (lista_entrada, i, j) //función para cambiar la posición
   de los valores en una
3.    //lista, las variables j e i tienen la función de ser los índices
4.    // de los valores que se quieren cambiar en la lista de entrada
5.
6.    DEFINIR temporal COMO ENTERO; // "temporal" es una variable que se utiliza
   para guardar el valor de la lista
7.    //de entrada en la posición "i" para que este no se pierda al momento de
   sustituir el valor de la lista de entrada
8.    // con índice i, posteriormente el valor de esta variable se le asigna al
   espacio de memoria de la lista de entrada
9.    //en la posición "j"
10.
11.    temporal <- lista_entrada[i]; // se guarda el valor que se encuentra en la
   posición i en temporal
12.    lista_entrada[i] <- lista_entrada[j]; // se le asigna el valor que se encuentra
   en "j"
13.    lista_entrada[j] <- temporal; // se cambia el valor que está en la posición "j"
   por el valor de temporal
14.
15. FIN SUBPROCESO
16.
17. SUBPROCESO indice_menor <- Particion (lista_entrada, l, r) // función que retorna
   un número entero el cual es el valor
18.    // la variable "r" es el último elemento de la lista y "l" una variable de
   control la cual actúa como primer elemento de la lista
19.    // para ayudarnos a iterar sobre la lista
20.    DEFINIR pivote COMO ENTERO;
21.
22.    pivote <- lista_entrada[r]; //pivote es el dato que se encuentra en la última
   localidad de memoria de la lista
23.
24.    IMPRIMIR "Particion (L: " l " pivote: " pivote " r: " r ")";
25.
26.    DEFINIR k COMO ENTERO;
27.
28.    DEFINIR indice_menor COMO ENTERO; //primera posición de la lista
29.
30.    indice_menor <- l - 1;
31.    imprimir "indice_menor: " indice_menor;
```

```

32.     PARA k <- 1 HASTA r - 1 CON PASO 1 HACER// ciclo for para poner todos los
    números menores al ultimo elementos de la lista
33.         SI lista_entrada[k] < pivote ENTONCES//antes de el
34.             indice_menor <- indice_menor + 1;
35.             Intercambiar(lista_entrada, indice_menor, k);
36.         FIN SI
37.     FIN PARA
38.
39.     indice_menor <- indice_menor + 1;
40.
41.     Intercambiar(lista_entrada, indice_menor, r);
42.
43. FIN SUBPROCESO
44.
45. SUBPROCESO Ordenar (lista_entrada, l, r)//funcion recursiva que parte la lista en
    dos secciones para poder Ordenar
46.     //ambas partes de la lista sin la necesidad de crear dos listas temporales
47.     //r es la longitud de la lista, l es la posicion en la que se quiere iniciar el
    conteo de elementos en la lista
48.     definir opc Como Entero;
49.     opc<-0;
50.     //////////////////////////////////
51.     Imprimir "lista antes de la partición [0]";
52.     Mientras opc<=r Hacer
53.         imprimir"lst[" opc ":" lista_entrada[opc];
54.         opc<-opc+1;
55.     FinMientras
56.     opc<-0;
57.     //////////////////////////////////
58.     SI l < r ENTONCES
59.
60.         IMPRIMIR "ORDENAR (L:" l " r:" r ")";
61.
62.         DEFINIR indice_particion COMO ENTERO;
63.         indice_particion <- Particion(lista_entrada, l, r);
64.         IMPRIMIR "L: " l " indice de partición: " indice_particion " r:" r;
65.
66.         //////////////////////////////////
67.         Imprimir "lista antes de ordner de l a indice_particion-1 [1]";
68.         opc<-l;
69.         Mientras opc<=indice_particion-1 Hacer
70.             imprimir"lst[" opc ":" lista_entrada[opc];
71.             opc<-opc+1;
72.         FinMientras
73.         opc<-0;
74.         //////////////////////////////////
75.         Ordenar(lista_entrada, l, indice_particion - 1);
76.         Imprimir "lista antes de ordner de l a indice_particion+1 [2]";
77.         opc<-indice_particion+1;
78.         Mientras opc<=r Hacer
79.             imprimir"lst[" opc ":" lista_entrada[opc];
80.             opc<-opc+1;
81.         FinMientras
82.         opc<-0;
83.         //////////////////////////////////
84.         Ordenar(lista_entrada, indice_particion + 1, r);
85.         //////////////////////////////////
86.         Imprimir "lista después de ordner de l a indice_particion+1 [3]";
87.         opc<-indice_particion+1;
88.         Mientras opc<=r Hacer
89.             imprimir"lst[" opc ":" lista_entrada[opc];

```

```

90.         opc<-opc+1;
91.     FinMientras
92.     opc<-0;
93.     //////////////////////////////////
94.     FIN SI
95.
96. FIN SUBPROCESO
97.
98. Proceso ordenamiento_rapido
99.     DEFINIR N COMO ENTERO;
100.
101.         IMPRIMIR "Dame el tamaño de la lista a ordenar:";
102.         LEER N;
103.
104.         DEFINIR lista COMO ENTERO;
105.         DIMENSION lista[N];
106.
107.         DEFINIR i COMO ENTERO;
108.
109.         PARA i <- 0 HASTA N - 1 CON PASO 1 HACER
110.             IMPRIMIR "Dame el valor de la lista en la posición " i ":";
111.             LEER lista[i];
112.         FIN PARA
113.
114.         Ordenar(lista, 0, N - 1);
115.
116.         PARA i <- 0 HASTA N - 1 CON PASO 1 HACER
117.             IMPRIMIR lista[i];
118.         FIN PARA
119.     FinProceso

```

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

valores de l y r

```
PSeInt - Ejecutando proceso ORDENAMIENTO_RAPIDO

lista antes de la partición [0]
lst[0]:3
lst[1]:2
lst[2]:1
lst[3]:6
lst[4]:5
lst[5]:4
lst[6]:9
ORDENAR (L:0 r:7)
Particion (L: 0 pivote: 7 r: 7)
indice_menor: -1
L: 0 indice de partición: 6 r:7
lista antes de ordner de l a indice_particion-1 [1]
lst[0]:3
lst[1]:2
lst[2]:1
lst[3]:6
lst[4]:5
lst[5]:4
lista antes de la partición [0]
lst[0]:3
lst[1]:2
lst[2]:1
lst[3]:6
lst[4]:5
ORDENAR (L:0 r:5)

☒ No cerrar esta ventana ☐ Siempre visible
```

Ilustración 1 valores de L y R en cada iteración


La lista antes de la partición de l a r

```
PSeInt - Ejecutando proceso ORDENAMIENTO_RAPIDO

Dame el valor de la lista en la posición 7:
> 7
lista antes de la partición [0]
lst[0]:3
lst[1]:2
lst[2]:1
lst[3]:6
lst[4]:5
lst[5]:4
lst[6]:9
lst[7]:7
```

Ilustración 2 lista antes de la partición - ciclo 1

El índice de partición



```
PSeInt - Ejecutando proceso ORDENAMIENTO_RAPIDO

ORDENAR (L:0 r:7)
Particion (L: 0 pivote: 8 r: 7)
indice_menor: -1
L: 0 índice de partición: 6 r:7
lista antes de ordner de l a indice_particion-1 [1]
lst[0]:3
lst[1]:2
lst[2]:1
lst[3]:6
lst[4]:5
lst[5]:4
lista antes de la partición [0]
lst[0]:3
lst[1]:2
lst[2]:1
lst[3]:6
lst[4]:5
lst[5]:4
ORDENAR (L:0 r:5)
Particion (L: 0 pivote: 4 r: 5)
indice_menor: -1
L: 0 índice de partición: 3 r:5
lista antes de ordner de l a indice_particion-1 [1]
lst[0]:3
lst[1]:2
lst[2]:1
lista antes de la partición [0]
lst[0]:3
lst[1]:2
```

☒ No cerrar esta ventana ☐ Siempre visible Ejecutar desde este punto

Ilustración 3 índice de partición de cada ciclo

La lista antes de ordenar de l a indice_particion - 1

```
PSelnt - Ejecutando proceso ORDENAMIENTO_RAPIDO
ORDENAR (L:0 r:7)
Particion (L: 0 pivote: 8 r: 7)
indice_menor: -1
L: 0 indice de partición: 6 r:7
lista antes de ordner de l a indice_particion-1 [1]
lst[0]:3
lst[1]:2
lst[2]:1
lst[3]:6
lst[4]:5
lst[5]:4
```

Ilustración 4 lista antes de ordenar de l a indice_particion - 1

lista antes de ordenar de indice_particion + 1 a r y después de ordenar de indice_particion + 1 a r

```
PSelnt - Ejecutando proceso ORDENAMIENTO_RAPIDO
L: 0 indice de partición: 0 r:2
lista antes de ordner de l a indice_particion-1 [1]
lista antes de la partición [0]
lista antes de ordner de l a indice_particion+1 [2]
lst[1]:2
lst[2]:3
lista antes de la partición [0]
lst[0]:1
lst[1]:2
lst[2]:3
ORDENAR (L:1 r:2)
Particion (L: 1 pivote: 3 r: 2)
indice_menor: 0
L: 1 indice de partición: 2 r:2
lista antes de ordner de l a indice_particion-1 [1]
lst[1]:2
lista antes de la partición [0]
lst[0]:1
lst[1]:2
lista antes de ordner de l a indice_particion+1 [2]
lista antes de la partición [0]
lst[0]:1
lst[1]:2
lst[2]:3
lista después de ordner de l a indice_particion+1 [3]
lista después de ordner de l a indice_particion+1 [3]
lst[1]:2
lst[2]:3
```

Explica cómo funciona el algoritmo con tus propias palabras.

El código de esta práctica este compuesto por tres funciones principales: “intercambiar, partición y ordenar”, estas funciones se encargan de implementar el algoritmo quick sort haciendo uso de una sola lista de datos y recursividad

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

Este subproceso sirve para cambiar la posición de los valores en una lista, las variables j e i tienen la función de ser los índices de los valores que se quieren cambiar en la lista de entrada, la variable "temporal" es una variable que se utiliza para guardar el valor de la lista de entrada en la posición "i" para que este no se pierda al momento de sustituir el valor de la lista de entrada con índice i,

posteriormente el valor de esta variable se le asigna al espacio de memoria de la lista de entrada en la posición "j".

Una vez realizado esto se guarda el valor que se encuentra en la posición "i" en temporal, después se le asigna el valor que se encuentra en "j" y por último se cambia el valor que está en la posición "j" por el valor de temporal.

¿Cómo funciona el subproceso `indice_menor <- Particion(lista_entrada, l, r)`?

Este subproceso retorna un número entero, la variable "r" es el último elemento de la lista y "l" una variable de control la cual actúa como primer elemento de la lista para ayudarnos a iterar sobre la lista, la variable "pivote" es el dato que se encuentra en la última localidad de memoria de la lista, la variable "indice_menor" es utilizada para saber en que para saber en qué posición de memoria se encuentra el ultimo valor ordenado en la lista

¿Cómo funciona el subproceso `Ordenar (lista_entrada, l, r)`?

Este subproceso es recursivo que parte la lista en dos secciones para poder Ordenar ambas partes de la lista sin la necesidad de crear dos listas temporales "r" es la longitud de la lista, "l" es la posición en la que se quiere iniciar el conteo de elementos en la lista e "índice de partición" es una variable que indica en que índice de la lista se debe de dividir para poder trabajar en ambas partes de la lista de una manera simultanea.