



**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: 1

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. # ORDENAMIENTO POR MEZCLA
2.
3. SUBPROCESO Mezclar (lista_entrada, l, m, r)
4.
5.     IMPRIMIR "MEZCLAR (" l " " m " " r ")";
6.
7.     DEFINIR Nl COMO ENTERO;
8.     DEFINIR Nr COMO ENTERO;
9.
10.    Nl <- m - 1;
11.    Nr <- r - m + 1;
12.
13.    IMPRIMIR Nl " " Nr;
14.
15.    DEFINIR Ml COMO ENTERO;
16.    DEFINIR Mr COMO ENTERO;
17.
18.    Ml <- Nl;
19.    Mr <- Nr;
20.
21.    SI Ml <= 0 ENTONCES
22.        Ml <- 1;
23.    FIN SI
24.
25.    SI Mr <= 0 ENTONCES
26.        Mr <- 1;
27.    FIN SI
28.
29.    DEFINIR lista_izquierda COMO ENTERO;
30.    DIMENSION lista_izquierda[Ml];
31.
32.    DEFINIR lista_derecha COMO ENTERO;
33.    DIMENSION lista_derecha[Mr];
34.
35.    DEFINIR i COMO ENTERO;
36.    DEFINIR j COMO ENTERO;
37.
38.    PARA i <- 0 HASTA Ml - 1 CON PASO 1 HACER
39.        lista_izquierda[i] <- lista_entrada[l + i];
40.    FIN PARA
41.
42.    PARA j <- 0 HASTA Mr - 1 CON PASO 1 HACER
43.        lista_derecha[j] <- lista_entrada[m + j];
44.    FIN PARA
45.
46.    DEFINIR k COMO ENTERO;
47.
48.    i <- 0;
```

```

49.   j <- 0;
50.   k <- 1;
51.
52.   MIENTRAS i < Nl Y j < Nr Y k <= r HACER
53.       SI lista_izquierda[i] <= lista_derecha[j] ENTONCES
54.           lista_entrada[k] = lista_izquierda[i];
55.           i <- i + 1;
56.       SINO
57.           lista_entrada[k] = lista_derecha[j];
58.           j <- j + 1;
59.       FIN SI
60.       k <- k + 1;
61.   FIN MIENTRAS
62.
63.   MIENTRAS i < Nl Y k <= r HACER
64.       lista_entrada[k] = lista_izquierda[i];
65.       i <- i + 1;
66.       k <- k + 1;
67.   FIN MIENTRAS
68.
69.   MIENTRAS j < Nr Y k <= r HACER
70.       lista_entrada[k] = lista_derecha[j];
71.       j <- j + 1;
72.       k <- k + 1;
73.   FIN MIENTRAS
74.
75. FIN SUBPROCESO
76.
77. SUBPROCESO Ordenar ( lista_entrada, l, r )
78.
79.   SI l < r ENTONCES
80.
81.       DEFINIR m COMO ENTERO;
82.
83.       m <- Redon( (l + r + 1) / 2 );
84.
85.       Imprimir "L: " l " M: " m " R: " r;
86.
87.       Ordenar(lista_entrada, l, m - 1);
88.
89.       Ordenar(lista_entrada, m, r);
90.
91.       Mezclar(lista_entrada, l, m, r);
92.
93.   FIN SI
94.
95. FIN SUBPROCESO
96.
97. DEFINIR N COMO ENTERO;
98.
99. IMPRIMIR "Dame el tamaño de la lista a ordenar:";
100.   LEER N;
101.
102.   DEFINIR lista COMO ENTERO;
103.   DIMENSION lista[N];
104.
105.   DEFINIR i COMO ENTERO;
106.
107.   PARA i <- 0 HASTA N - 1 CON PASO 1 HACER
108.       IMPRIMIR "Dame el valor de la lista en la posición " i ":";
109.       LEER lista[i];

```

```

110.      FIN PARA
111.
112.      Ordenar(lista, 0, N - 1);
113.
114.      PARA i <- 0 HASTA N - 1 CON PASO 1 HACER
115.          IMPRIMIR lista[i];
116.      FIN PARA

```

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

```

1. SUBPROCESO Mezclar (lista_entrada, l, m, r)// función Mezclar
2.
3.      IMPRIMIR "MEZCLAR (" l " " m " " r ")"; //l= inicio de la lista izquierda,
      valor del index medio de la lista, longitud de la lista -1
4.
5.      DEFINIR Nl COMO ENTERO;
6.      DEFINIR Nr COMO ENTERO;
7.
8.      Nl <- m - 1; // índice de inicio en la lista izquierda
9.      Nr <- r - m + 1; // índice de inicio en la lista derecha, en caso de que la
      lista tenga un numero impar de elementos le agrega un valor de la lista izq para
      hacer su longitud un numero entero y poder dividir la lista en dos partes
10.
11.     IMPRIMIR "Nl: " Nl " ,Nr: " Nr;
12.
13.     DEFINIR Ml COMO ENTERO;
14.     DEFINIR Mr COMO ENTERO;
15.
16.     Ml <- Nl; // variables para iterar sobre la lista sin tener que modificar los
      valores verdaderos
17.     Mr <- Nr; //
18.
19.     SI Ml <= 0 ENTONCES
20.         Ml <- 1;
21.     FIN SI
22.
23.     SI Mr <= 0 ENTONCES
24.         Mr <- 1;
25.     FIN SI
26.
27.     DEFINIR lista_izquierda COMO ENTERO; // creación de las listas con los tamaños
      calculados en la líneas 8 y 9
28.     DIMENSION lista_izquierda[Ml];
29.
30.     DEFINIR lista_derecha COMO ENTERO;
31.     DIMENSION lista_derecha[Mr];
32.
33.     DEFINIR i COMO ENTERO;
34.     DEFINIR j COMO ENTERO;
35.
36.     PARA i <- 0 HASTA Nl - 1 CON PASO 1 HACER // asignación de valores de la lista
      de entrada, los valores almacenados en esta lista van del índice[0] al índice[n]
37.         lista_izquierda[i] <- lista_entrada[l + i];
38.     FIN PARA
39.
40.     PARA j <- 0 HASTA Nr - 1 CON PASO 1 HACER// asignación de valores de la lista
      de entrada, los valores almacenados en esta lista van del índice[n/2+1] al índice[n-
      1]

```

```

41.     lista_derecha[j] <- lista_entrada[m + j];
42.     FIN PARA
43.
44.     DEFINIR k COMO ENTERO;
45.
46.     i <- 0;
47.     j <- 0;
48.     k <- 1;
49.
50.     MIENTRAS i < Nl Y j < Nr Y k <= r HACER //método burbuja comparando los
valores entre ambas lista para separar los valores menores y superiores
51.         SI lista_izquierda[i] <= lista_derecha[j] ENTONCES
52.             lista_entrada[k] = lista_izquierda[i];
53.             i <- i + 1;
54.         SINO
55.             lista_entrada[k] = lista_derecha[j]; // asignación de los valores
menores a la lista de entrada
56.             j <- j + 1;
57.         FIN SI
58.         k <- k + 1;
59.     FIN MIENTRAS
60.
61.     MIENTRAS i < Nl Y k <= r HACER
62.         lista_entrada[k] = lista_izquierda[i]; // asignación de los valores mayores
a la lista de entrada
63.         i <- i + 1;
64.         k <- k + 1;
65.     FIN MIENTRAS
66.
67.     MIENTRAS j < Nr Y k <= r HACER
68.         lista_entrada[k] = lista_derecha[j]; //concatenación de las listas creadas
con los valores ordenados
69.         j <- j + 1;
70.         k <- k + 1;
71.     FIN MIENTRAS
72.
73. FIN SUBPROCESO
74.
75. SUBPROCESO Ordenar ( lista_entrada, l, r )// r=N-1; r indica el índice de la mitad
de la lista, l es el valor inicial de la lista izquierda
76.
77.     SI l < r ENTONCES
78.
79.         DEFINIR m COMO ENTERO;
80.
81.         m <- Redon( (l + r + 1) / 2 ); // m es el valor que se encuentra en medio
entre el índice de inicio y el índice de final ayuda a dividir la lista y organizar
los valores de menor a mayor por cada iteración de la lista
82.         // de entrada
83.
84.         Imprimir "L: " l " M: " m " R: " r;
85.
86.         Ordenar(lista_entrada, l, m - 1); // recursividad de la lista para orden
los valores de la lista[0] a lista[n/2-1]
87.
88.         Ordenar(lista_entrada, m, r); // recursividad de la lista para orden los
valores de la lista[n] a lista[n-1]
89.         imprimir "lista de entrada despues del segundo ordenar, antes de
mezclar\n";
90.         definir jj Como Entero;
91.         jj<-0;

```

```

92.      mientras jj<=r Hacer
93.          imprimir "list[" jj "]: " lista_entrada[jj];
94.          jj<-jj+1;
95.      FinMientras
96.      Mezclar(lista_entrada, l, m, r);
97.      imprimir "lista entrada despues de mezclar\n";
98.      jj<-0;
99.      mientras jj<=r Hacer
100.          imprimir "list[" jj "]: " lista_entrada[jj];
101.          jj<-jj+1;
102.      FinMientras
103.      FIN SI
104.
105.      FIN SUBPROCESO
106.      Proceso practica_1
107.          DEFINIR N COMO ENTERO;
108.
109.          IMPRIMIR "Dame el tamaño de la lista a ordenar:";
110.          LEER N;
111.
112.          DEFINIR lista COMO ENTERO;
113.          DIMENSION lista[N];
114.
115.          DEFINIR i COMO ENTERO;
116.
117.          PARA i <- 0 HASTA N - 1 CON PASO 1 HACER
118.              IMPRIMIR "Dame el valor de la lista en la posición " i ":";
119.              LEER lista[i];
120.          FIN PARA
121.
122.          imprimir "N-1: " N-1;
123.
124.          Ordenar(lista, 0, N - 1);
125.
126.          PARA i <- 0 HASTA N - 1 CON PASO 1 HACER
127.              IMPRIMIR lista[i];
128.          FIN PARA
129.      FinProceso

```

2) Modifica el pseudocódigo para imprimir la tabla de las listas

```
1. SUBPROCESO Mezclar (lista_entrada, l, m, r)// función Mezclar
2.
3.     IMPRIMIR "MEZCLAR (" l " " m " " r ")"; //l= inicio de la lista izquierda,
    valor del index medio de la lista, longitud de la lista -1
4.
5.     DEFINIR Nl COMO ENTERO;
6.     DEFINIR Nr COMO ENTERO;
7.
8.     Nl <- m - 1; // index de inicio en la lista izquierda
9.     Nr <- r - m + 1; // index de inicio en la lista derecha, en caso de que la
    lista tenga un numero impar de elementos le agrega un valor de la lista izq para
    hacer su longitud un numero entero y poder dividir la lista en dos partes
10.
11.    IMPRIMIR "Nl: " Nl " ,Nr: " Nr;
12.
13.    DEFINIR Ml COMO ENTERO;
14.    DEFINIR Mr COMO ENTERO;
15.
16.    Ml <- Nl; // variables para iterar sobre la lista sin tener que modificar los
    valores verdades
17.    Mr <- Nr; //
18.
19.    SI Ml <= 0 ENTONCES
20.        Ml <- 1;
21.    FIN SI
22.
23.    SI Mr <= 0 ENTONCES
24.        Mr <- 1;
25.    FIN SI
26.
27.    DEFINIR lista_izquierda COMO ENTERO; // creacion de las listas con los tamaños
    calculador en la lineas 8 y 9
28.    DIMENSION lista_izquierda[Ml];
29.
30.    DEFINIR lista_derecha COMO ENTERO;
31.    DIMENSION lista_derecha[Mr];
32.
33.    DEFINIR i COMO ENTERO;
34.    DEFINIR j COMO ENTERO;
35.
36.    PARA i <- 0 HASTA Nl - 1 CON PASO 1 HACER // asignacion de valores de la lista
    de entrada, los valores almacenados en esta lista van del index[0] al index[n]
37.        lista_izquierda[i] <- lista_entrada[l + i];
38.    FIN PARA
39.
40.    PARA j <- 0 HASTA Nr - 1 CON PASO 1 HACER// asignacion de valores de la lista
    de entrada, los valores almacenados en esta lista van del index[n/2+1] al index[n-
    1]
41.        lista_derecha[j] <- lista_entrada[m + j];
42.    FIN PARA
43.
44.    DEFINIR k COMO ENTERO;
45.
46.    i <- 0;
47.    j <- 0;
48.    k <- 1;
49.
```

```

50.     MIENTRAS i < Nl Y j < Nr Y k <= r HACER //me todo burbuja comparando los
valores entre ambas lista para se parar los valores menores y superiores
51.         SI lista_izquierda[i] <= lista_derecha[j] ENTONCES
52.             lista_entrada[k] = lista_izquierda[i];
53.             i <- i + 1;
54.         SINO
55.             lista_entrada[k] = lista_derecha[j]; // asignacion de los valores
menores a la lista de entrada
56.             j <- j + 1;
57.         FIN SI
58.         k <- k + 1;
59.     FIN MIENTRAS
60.
61.     MIENTRAS i < Nl Y k <= r HACER
62.         lista_entrada[k] = lista_izquierda[i]; // asignacion de los valores mayores
a la lista de entrada
63.         i <- i + 1;
64.         k <- k + 1;
65.     FIN MIENTRAS
66.
67.     MIENTRAS j < Nr Y k <= r HACER
68.         lista_entrada[k] = lista_derecha[j]; //concatenacion de las listas creadas
con los valores ordenados
69.         j <- j + 1;
70.         k <- k + 1;
71.     FIN MIENTRAS
72.
73. FIN SUBPROCESO
74.
75. SUBPROCESO Ordenar ( lista_entrada, l, r )// r=N-1; r indca el indice de la mitad
de la lista, l es el valor inicial de la lista izquierda
76.
77.     SI l < r ENTONCES
78.
79.         DEFINIR m COMO ENTERO;
80.
81.         m <- Redon( (l + r + 1) / 2 ); // m es el valor que se ecuentra en medio
entre el index de inicio y el index de final ayuda a dividir la lista y organizar
los valores de meno a mayor por cada iteración de la lista
82.         // de entrada
83.
84.         Imprimir "L: " l " M: " m " R: " r;
85.         definir jj Como Entero;
86.         jj<-0;
87.         imprimir "antes del primer ordenar (l a m-1)";
88.         mientras jj<=m Hacer
89.             imprimir "list[" jj "]: " lista_entrada[jj];
90.             jj<-jj+1;
91.         FinMientras
92.         Ordenar(lista_entrada, l, m - 1); // recursividad de la lista para ordenn
los valores de la lista[0] a lista[n/2-1]
93.         imprimir "lista de entrada antes del segundo ordenar, (m a r)";
94.         jj<-0;
95.         mientras jj<=r Hacer
96.             imprimir "list[" jj "]: " lista_entrada[jj];
97.             jj<-jj+1;
98.         FinMientras
99.         Ordenar(lista_entrada, m, r); // recursividad de la lista para ordenn los
valores de la lista[n] a lista[n-1]
100.         imprimir "lista entrada antes de mezclar\n";
101.         jj<-0;

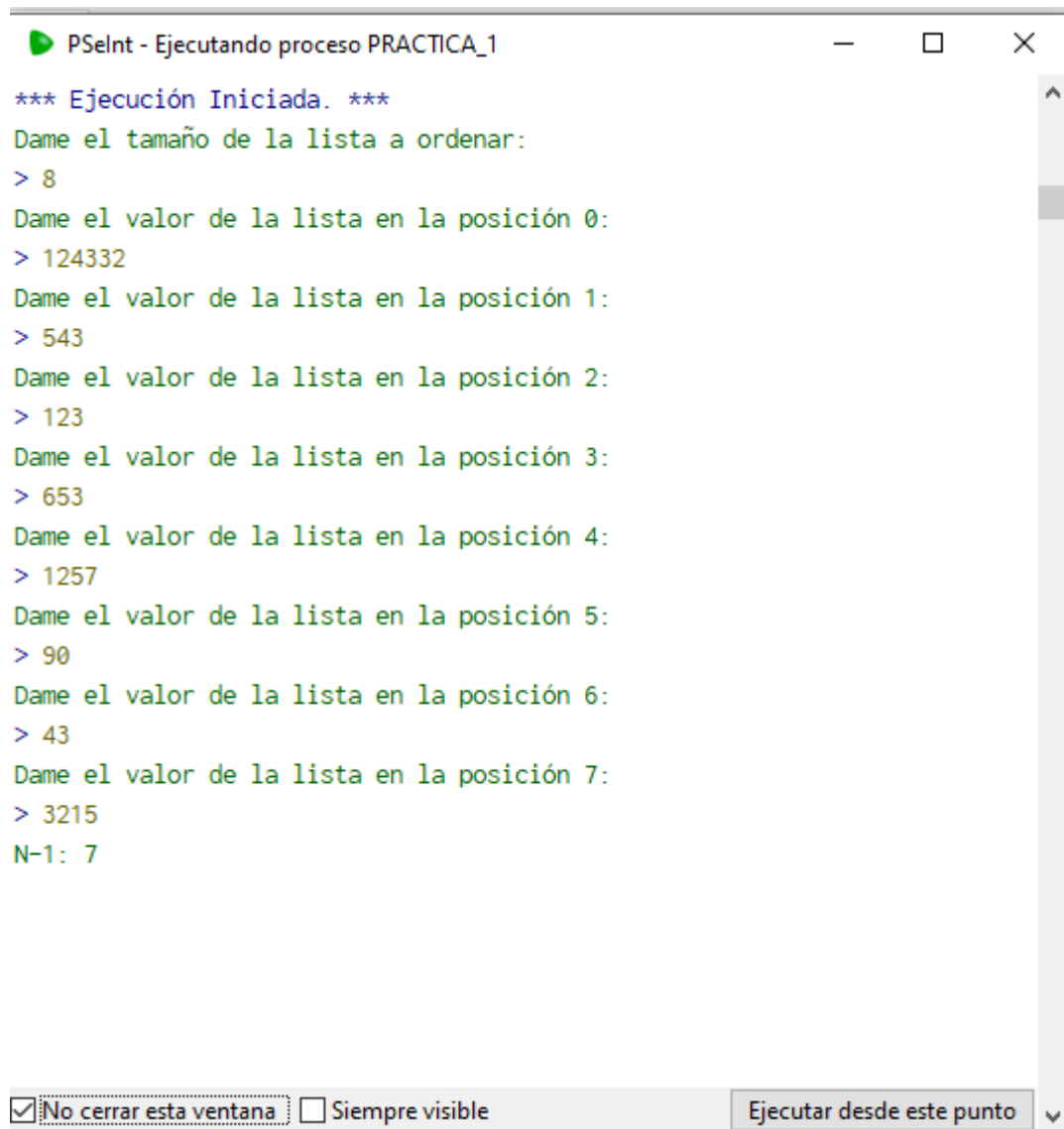
```



```

102.          mientras jj<=r Hacer
103.              imprimir "list[" jj "]: " lista_entrada[jj];
104.              jj<-jj+1;
105.          FinMientras
106.          Mezclar(lista_entrada, l, m, r);
107.          imprimir "lista entrada despues de mezclar\n";
108.          jj<-0;
109.          mientras jj<=r Hacer
110.              imprimir "list[" jj "]: " lista_entrada[jj];
111.              jj<-jj+1;
112.          FinMientras
113.      FIN SI
114.
115.  FIN SUBPROCESO
116.  Proceso practica_1
117.      DEFINIR N COMO ENTERO;
118.
119.      IMPRIMIR "Dame el tamaño de la lista a ordenar:";
120.      LEER N;
121.
122.      DEFINIR lista COMO ENTERO;
123.      DIMENSION lista[N];
124.
125.      DEFINIR i COMO ENTERO;
126.
127.      PARA i <- 0 HASTA N - 1 CON PASO 1 HACER
128.          IMPRIMIR "Dame el valor de la lista en la posición " i ":";
129.          LEER lista[i];
130.      FIN PARA
131.
132.      imprimir "N-1: " N-1;
133.
134.      Ordenar(lista, 0, N - 1);
135.
136.      PARA i <- 0 HASTA N - 1 CON PASO 1 HACER
137.          IMPRIMIR lista[i];
138.      FIN PARA
139.  FinProceso

```



```
*** Ejecución Iniciada. ***
Dame el tamaño de la lista a ordenar:
> 8
Dame el valor de la lista en la posición 0:
> 124332
Dame el valor de la lista en la posición 1:
> 543
Dame el valor de la lista en la posición 2:
> 123
Dame el valor de la lista en la posición 3:
> 653
Dame el valor de la lista en la posición 4:
> 1257
Dame el valor de la lista en la posición 5:
> 90
Dame el valor de la lista en la posición 6:
> 43
Dame el valor de la lista en la posición 7:
> 3215
N-1: 7
```

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

Ilustración 1- declaración del tamaño de la lista y asignación de valores.

```
PSeInt - Ejecutando proceso PRACTICA_1
N=1: 7
L: 0 M: 4 R: 7
antes del primer ordenar (l a m-1)
list[0]: 124332
list[1]: 543
list[2]: 123
list[3]: 653
list[4]: 1257
L: 0 M: 2 R: 3
antes del primer ordenar (l a m-1)
list[0]: 124332
list[1]: 543
list[2]: 123
L: 0 M: 1 R: 1
antes del primer ordenar (l a m-1)
list[0]: 124332
list[1]: 543
lista de entrada antes del segundo ordenar, (m a r)
list[0]: 124332
list[1]: 543
lista entrada antes de mezclar\n
list[0]: 124332
list[1]: 543
MEZCLAR (0 1 1)
N1: 1 ,Nr: 1
lista entrada despues de mezclar\n
list[0]: 543
list[1]: 124332
lista de entrada antes del segundo ordenar, (m a r)
list[0]: 543
list[1]: 124332
list[2]: 123
```

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

Ilustración 2 impresión de las listas antes de ser ordenadas.

```

PSeInt - Ejecutando proceso PRACTICA_1

list[7]: 3215
MEZCLAR (0 4 7)
N1: 4 ,Nr: 4
lista entrada despues de mezclar\n
list[0]: 43
list[1]: 90
list[2]: 123
list[3]: 543
list[4]: 653
list[5]: 1257
list[6]: 3215
list[7]: 124332
lista ordenada
43
90
123
543
653
1257
3215
124332
*** Ejecución Finalizada. ***

☒ No cerrar esta ventana ☐ Siempre visible Reiniciar

```

Ilustración 3 última iteración sobre las lista e impresión de la lista final

3) Tablas de listas antes y después de cada cambio

Lista entrada antes y después de ser ordenada

Antes	después
124332	43
543	90
123	123
654	543
1257	653
90	1257
43	3215
3215	124332

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

El algoritmo utilizado se puede ser dividido en dos partes, en la función “Mezclar” y la función “Ordenar”. La función “mezclar” recibe 3 parámetros los cuales son la lista de entrada, el valor “ L ”, “ m ”, “ r ” los cuales se usan para determinar el valor inicial, el valor medió y el valor final de la longitud de la lista, con la ayuda de estos valores se crean dos listas “lista_izquierda” y “lista_derecha” estas

dos listas se utilizan para tomar los valores de la lista de entrada y separarlos en dos secciones, los valores antes lista_entrada[0] a lista_entrada[m] y de lista_entrada[m+1] a lista_entrada[r] estos rangos de valores se asignan a las lista_izquierda y lista_derecha respectivamente, posteriormente se hace uso del método burbuja para poder ordenar los valores de las listas der e izq.

En la función “ordenar” también se toman como parámetros tres valores, en este caso son “Lista_entrada, l, r” donde “L” es el valor inicial de la lista, “r” es el valor de la longitud de la lista al cual se le resta una unidad debido a que la numeración de las listas empieza en 0, para poder iterar sobre todos los valores de la lista de entrada se utiliza una variable de control “m” la cual se le asigna el valor medío de la longitud de la lista de entrada, después se hace uso de un proceso recursivo llamando a la misma función para que se lecciones los valores de menor a mayor por cada elemento de las lista, primero se ordenan los valores de 0 a m y después de m+1 a r-1, una vez que fueron seleccionados los valores se pasan a la función “mezclar” para ordenarlos de menor a mayor y después retornar la lista de entrada.

Conclusiones

En esta práctica se pudo apreciar la practicidad que tiene el uso de pseudo códigos antes de empezar la programación de un programa en cualquier lenguaje, esto se debe a que el uso del pseudo código ayudo a facilitar la comprensión el valor de cada variable utilizado en el método de ordenamiento empleado en la práctica.