Departamento de Programación
Facultad de Informática
Universidad Nacional del Comahue

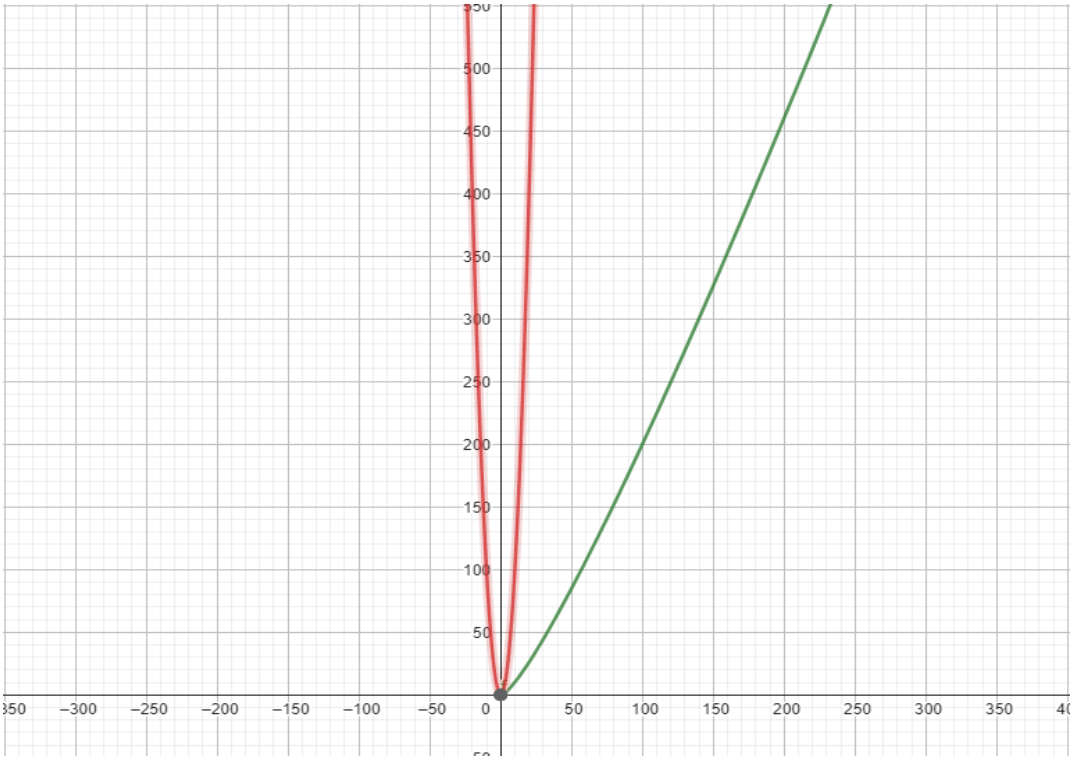
DESARROLLO DE ALGORITMOS
2do cuatrimestre 2022
Trabajo Práctico FINAL

Alumno: Leonardo David Galindez
DNI: 44237998
Legajo:FAI-3862
Carrera: Licenciatura en Sistemas de Información
Índice de referencia: 1
Archivos: estadiosMundiales.txt,Estadio.java,testEstadio.java

Que hace el Algoritmo:

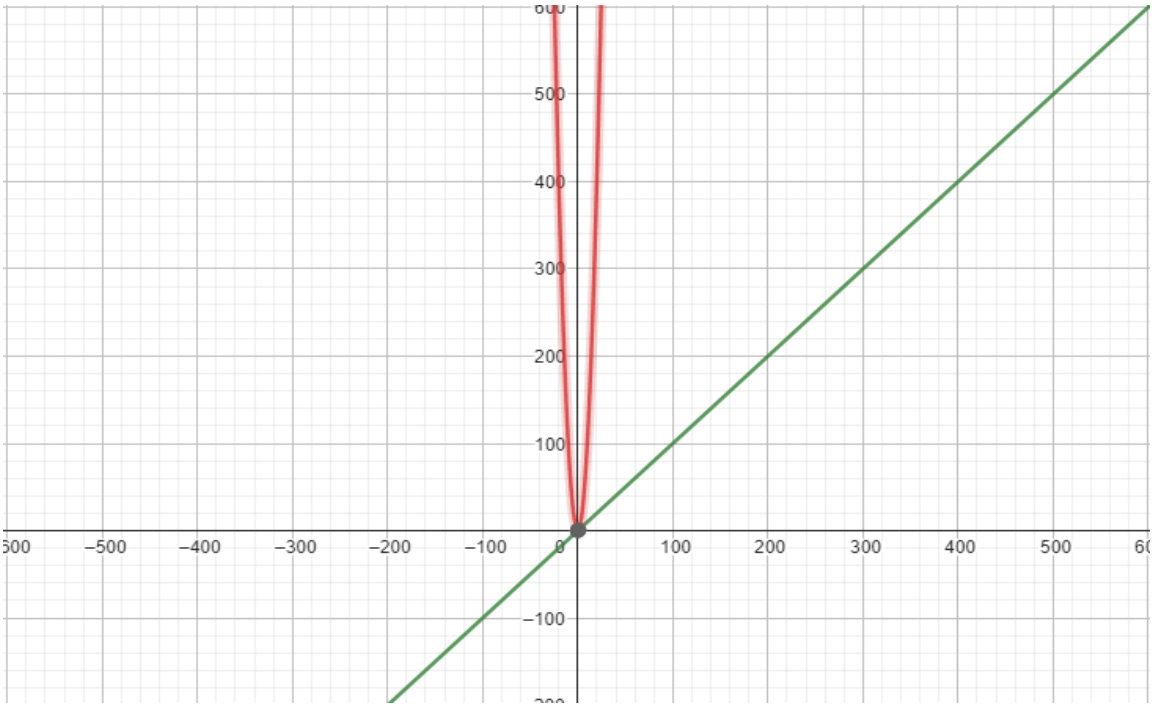
Este algoritmo al ejecutarlo lo primero que va a hacer es leer el archivo de la dirección dada y guardarlo en un arreglo, todo esto antes de mostrar el menú con las siguientes opciones, poder ordenar el arreglo por el método de ordenamiento quicksort por nombre de la ciudad con el método definido en la clase Estadio compareTo de forma ascendente o descendente a elección del usuario, también tenemos la posibilidad de ordenar mediante otro método que es el de inserción también con la mismas posibilidades de orden que el quicksort otro de las opciones que posee es la función de abreviatura que va a solicitarle al usuario el número del estadio que desee y esta función va a retornar el nombre del estadio sin vocales ni espacio y con la primer letra en mayúscula y por último está la opción de finalizar la ejecución.

Comparaciones de tiempo: Entre Inserción y quicksort



Quicksort:
Funcion Verde: mejor caso:O(n*log(n))

Función Naranja: peor caso:O(n*n)



Inserción:
Función verde mejor caso:O(n)
Función Naranja: Peor caso:O(n*n)

Tiempos y promedios:

Inserción:

Ascendiente con 100 Estadios:

1er	120300 ns
2da	125800 ns
3ra	228900 ns

Promedio: 158.333,33 ns

Descendiente con 100 Estadios:

1er	180700 ns

2da	233600 ns
3ra	227000 ns

Promedio: 213.766,66 ns

Ascendiente con 25 Estadios:

1er	79800 ns
2da	47200 ns
3ra	41500 ns

Promedio: 56.166,66 ns

Descendiente con 25 Estadios:

1er	38300 ns
2da	53700 ns
3ra	42600 ns

Promedio: 44.866,66 ns

Quicksort:

Ascendiente con 100 Estadios:

1er	31300 ns
2da	76200 ns
3ra	33200 ns

Promedio:46.900 ns

Descendiente con 100 Estadios:

1er	66500 ns
2da	64400 ns
3ra	69000 ns

Promedio:66.633,33 ns

Ascendiente con 25 Estadios:

1er	29900 ns

2da	23400 ns
3ra	22700 ns

Promedio:25.333,33 ns

Descendiente con 25 Estadios:

1er	31800 ns
2da	27500 ns
3ra	25200 ns

Promedio:28.166,66 ns

Análisis con 100 estadios:

El Método de ordenamiento quicksort ascendiente es de 3,375977185501066 ns veces más rápido que el método de ordenamiento de inserción.

El método de ordenamiento quicksort descendiente es de 3,208104112461436 ns veces más rápido que el método ordenamiento de inserción.

Análisis con 25 estadios:

El método de ordenamiento quicksort ascendiente es de 2,21710529172438 ns veces más rápido que el método ordenamiento de inserción.

El método de ordenamiento quicksort descendiente es de 1,592899548615278 ns veces más rápido que el método ordenamiento de inserción.

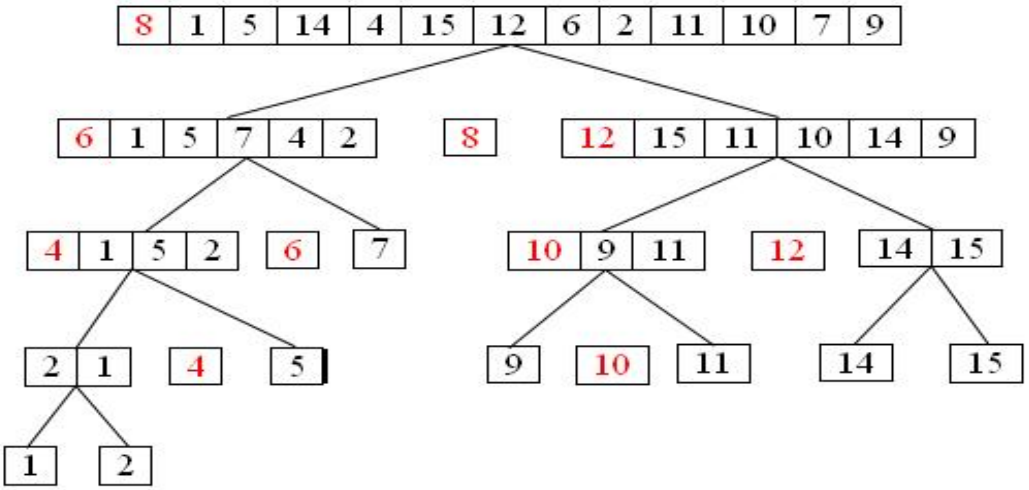
Conclusión

Como se ve en los datos de tiempo, el método de ordenamiento quicksort es más eficiente que el método de ordenamiento denominado inserción.

Si vemos las tablas de tiempos quicksort con 25 estadios y 100 estadios no hay mucha diferencia de tiempo esto por el tipo de eficiencia del método quicksort que es n*log(n) en cambio con las tablas de tiempo de inserción las diferencia de tiempo con 25 y 100 estadios son más visibles, por esto decimos que para números más grandes el método quicksort es más eficiente que el método de inserción.

***observación:**para lograr ver una gran diferencia de tiempo en el método quicksort tenemos que probar con números mucho más grandes.

Explicación del Método Ordenamiento quicksort:



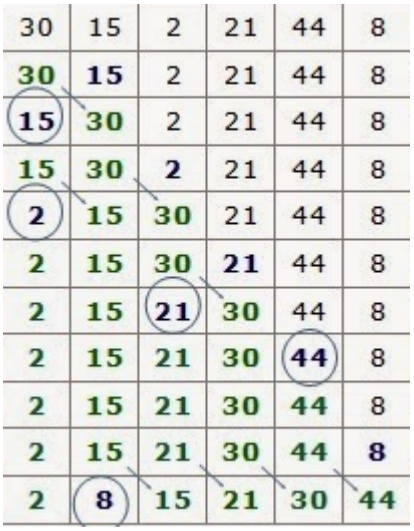
Juntando los elementos, el arreglo quedaría ordenado

1	2	4	5	6	7	8	9	10	11	12	14	15

Quicksort El algoritmo inicia asignando a las variables integradoras donde comienza y dónde termina el recorrido,luego elige un número del arreglo como pivote a partir de este valor vamos a poder separar los números mayores o menores que el pivote. Cuando termine de separar los mayores y menores el pivote se coloca en la posición del último número menor que el pivote encontrado, al finalizar este intercambio el pivote estará entre los números menores y mayores que el. Al tener separados estos números podemos realizar otra vez el ordenamiento quicksort, con el sub arreglo de menores que el valor del

pivote, solamente que sus parámetros serían de 0 a la posición del pivote -1 y luego lo mismo con el sub arreglo de los mayores que el pivote y sus parametro serían la posición del pivote +1 y longitud del arreglo -1. Así hasta que la cantidad de números mayores y menores sea 1 cuando suceda esto, tendríamos el arreglo ordenado.

Explicación del Método Ordenamiento Inserción:



Inserción: El algoritmo se inicia con los siguientes parámetros, el arreglo y una variable i que empieza en 1 esta se va a ir iterando hasta la longitud del arreglo -1 , luego el elemento de la posición i se guarda en una variable auxiliar y se compara con los elementos anteriores hasta la posición 0, si el elemento en la posición i es menor que alguno de los anteriores se intercambian por ese elemento mayor, si no es menor que ninguno se intercambia con elementos de la posición 0, repitiendo este método hasta que i sea igual a la longitud del arreglo -1.

UML TDA ESTADIO:

Estadio
- numero:int - nombre:String - ciudad:String - capacidad:int - mundial:String
Constructores + Ingrediente(int num) + Ingrediente(int num,String nom,String ciu,int capa,String anioM) Observadores + getNumero():int + getNombre():String + getCiudad():String + getCapacidad():int + getMundial():String + toString():String Comparadores + equals (Estadio nuevoEstadio):boolean + compareTo(Estadio nuevoEstadio):int Modificadores +setNombre(String nom) +setCiudad(String ciu) +setCapacidad(int capa) +setMundial(String anioM) + setMedidaGrama(double medG)

Pseudocódigo:

```
Algoritmo testEstadio()
    Entero rta, cantEstadios, numEstadio, pos <- 0, posEstadio, rtaSub;
    logico valor <- false
    long tiempoI, tiempoF, tiempoTotal;
    texto nomModificado
```

```
Estadio Estadios[] <-crear Estadio[1000000]//definimos un arreglo sobredimensionado
cantEstadios <- leerTxt(direccion, Estadios, 1)//metodo para leer el archivo
Estadio copiaEstadios[]<- new Estadio[cantEstadios]
REPETIR
   //Menu
   rta <- MostrarMenu()
   SEGUN (rta) HACER
       1:
         //Mostramos al arreglo de estadios original
         MostrarEstadios(Estadios, cantEstadios)
        termina
      2://quicksort
         rtaSub <- MostrarSubMenu();
         SEGUN(rtaSub) HACER
            1:
               copiarArreglo(Estadios, copiaEstadios)
               tiempoI <- 0;
               tiempoI <- IniciamaNanoTiempo()            //iniciamos el tiempo
               quicksortA(copiaEstadios, 0, cantEstadios - 1)
               tiempoF <- TerminaNanoTiempo()  //finalizamos el tiempo
               MostrarEstadios(copiaEstadios, cantEstadios)
               tiempoTotal <- tiempoF - tiempoI;          //tiempo total
               tiempoTotal <- tiempoTotal;    //pasamos de nanosegundos a segundos
              ESCRIBIR("tiempo quicksort Ascendiente:" + tiempoTotal);
              TERMINA
            2:
               copiarArreglo(Estadios, copiaEstadios)
               tiempoI <- 0;
               tiempoI <- IniciamaNanoTiempo() ;//modularizar
               quicksortD(copiaEstadios, 0, cantEstadios - 1);
              tiempoF <- TerminaNanoTiempo()
               MostrarEstadios(copiaEstadios, cantEstadios);
               tiempoTotal <- tiempoF - tiempoI;
               tiempoTotal <- tiempoTotal;
               Escribir("tiempo quicksort descendiente:" + tiempoTotal);
               TERMINA
            0:
               Escribir("Finalizado");
               TERMINA
            ParaTodoOtrocaso:
               Escribir("Error");
               TERMINA
         FINSEGUN
         TERMINA
      3://insercion
```

```
rtaSub <- MostrarSubMenu();
SEGUN(rtaSub) HACER
   1:
      copiarArreglo(Estadios, copiaEstadios);
      tiempoI <- 0;
      tiempoI <- IniciaNanoTiempo()          //tiempo inicial  cambiar a long
      insercionA(copiaEstadios, cantEstadios)
      tiempoF <- TerminaNanoTiempo()     //tiempo final
      MostrarEstadios(copiaEstadios, cantEstadios)
      tiempoTotal <- tiempoF - tiempoI;        //tiempo total
      tiempoTotal <- tiempoTotal;
      ESCRIBIR("tiempo insercion Ascendiente:" + tiempoTotal);
      TERMINA
   2:
      copiarArreglo(Estadios, copiaEstadios);
      tiempoI <- 0;
      tiempoI <- IniciaNanoTiempo()
      insercionD(copiaEstadios, cantEstadios)
      MostrarEstadios(copiaEstadios, cantEstadios)
      tiempoF <- TerminaNanoTiempo()
      tiempoTotal <- tiempoF - tiempoI
      tiempoTotal <- tiempoTotal
      ESCRIBIR("tiempo insercion descendiente:" + tiempoTotal);
      TERMINA
   0:
      ESCRIBIR("Finalizado")
      TERMINA;
   ParaTodoOtro:
      ESCRIBIR("Error")
      TERMINA
FINSEGUN
TERMINA
4://Abreviatura
REPETIR
   ESCRIBIR("Ingrese numero Estadio");
  LEER  numEstadio
MIENTRAS (numEstadio > cantEstadios OR numEstadio <= 0);
//solo puede  ingresar un numero valido
posEstadio <- posEstadio(Estadios, numEstadio, pos, cantEstadios)
nomModificado <- abreviatura(Estadios, posEstadio, pos)
nomModificado <- Mayuscula(nomModificado)
ESCRIBIR(Estadios[posEstadio].obtenerNombre());
ESCRIBIR(nomModificado);
TERMINA
0://Finalizar
```

```
            valor <- VERDADERO
            TERMINA
          ParaTodoOtro:
            ESCRIBIR("Error")
           TERMINA
        FINSEGUN
      MIENTRAS(NOTvalor)
  FIN ALGORITMO testEstadio

MODULO copiarArreglo(Estadio[] arr, Estadio[] copia) RETORNA VACIO
      Entero i
      PARA( i <- 0 HASTA  copia.length-1 PASO  1) HACER
        copia[i] <- arr[i]
      FIN PARA
FIN MODULO

  //Menus
  MODULO MostrarSubMenu() RETORNA ENTERO
    ENTERO rtaSub;
    //SubMenu
    ESCRIBIR("Ingrese orden ");
    ESCRIBIR("Ascendente---------1");
    ESCRIBIR("Descendiente-------2");
    ESCRIBIR("Finalizar----------0");
    LEER( rtaSub)
    retorna rtaSub
  FIN MODULO

  MODULO MostrarMenu() RETORNA ENTERO
    ENTERO rta
    ESCRIBIR("")
    //menu
    ESCRIBIR("Ordenado por numero de Estadio----------1");
    ESCRIBIR("Ordenamiento quicksort por Ciudad--------2");
    ESCRIBIR("Ordenamiento insercion por Ciudad--------3");
    ESCRIBIR("Abreviatura-----------------------------4");
    ESCRIBIR("Finalizar-------------------------------0");
    LEER (rta)
    retorna  rta
  FIN MODULO
  //Método para leer el archivo txt
  MODULO leerTxt(texto direccion,Estadio Estadios[],entero cantEstadios)RETORNA
ENTERO
    INTENTO HACER
```

```
        int j <- 0, cantAtributos <- 5;//Si agregamos mas atributos le asignamos más
posiciones al array
        String Atributos[] <- crear String[cantAtributos]
        BufferedReader bf <- new BufferedReader(new FileReader(direccion));//buffer lee
linea por linea
        texto line
        MIENTRAS ((linea = bf.readLine()) != null) HACER //repite mientras bf tiene datos
           ObtenerAtributos(linea, Atributos)  //obtenemos los atributos separados en un
arreglo
           CargarEstadio(Estadios, Atributos, j)//mandamos el arreglo atributos para cargarlo
a cada estadio
           j<-J+1 //j es la variable iteradora que la usamos en la cargar del estadio
           cantEstadios<-cantEstadios+1;//tenemos un control de la cantidad de estadios
que se cargaron al arreglo
        FIN MIENTRAS
        bf.cerrar();//cerramos el archivo
      1er  ERROR (FileNotFoundException ex) {//error de archivo no encontrado
        ESCRIBIR(ex.obtenerMessage() + "\nSignifica que el archivo del "
              + "que queriamos leer no existe.");
       2do  ERROR (IOException ex) {//error de permisos
        ESCRIBIR("Error leyendo o escribiendo en algun archivo.");
      TERMINA
      //podemos agregar otro catch Exception y mostrar cualquier error
      retorna cantEstadios - 1
   FIN INTENTO

   //separa la linea en atributos y lo guardo en un arreglo
   MODULO ObtenerAtributos(texto  linea, texto Atributos[]) RETORNA VACIO

      TEXTO atributo;
      Entero i <-  0, posIni <- 0, posEnd
      MIENTRAS(i < LONGITUD(Atributos)) {
        posEnd <- indiceDe(linea)f("|", posIni);
        atributo <- subCadena(linea,posIni, posEnd)
        Atributos[i] <- atributo;
        posIni <- posEnd + 1;
        i<-i+1
      FIN MIENTAS
   FIN MÓDULO

   MODULO  CargarEstadio(Estadio Estadios[], TEXTO Atributos[], int j) RETORNA VACIO
//le asigno los atributos al estadio
      TEXTO nombre, ciudad, mundial
      ENTERO numero, capacidad;
      numero <-  Integer.pasamosAInt(Atributos[0])//convertimos string a int
```

```
        nombre <- removerEspacios(Atributos[1]);
        ciudad <- removerEspacios(Atributos[2]);
        capacidad <- Integer.pasamosAInt(Atributos[3])//convertimos string a int
        mundial = removerEspacios(Atributos[4]);

        Estadio nuevoEstadio <-  crear Estadio(numero, nombre, ciudad, capacidad,
mundial)//constructor
        Estadios[j] <- nuevoEstadio//asignamos el objeto a al arreglo segun la posicion j
    FIN MODULO

   MODULO MostrarEstadios(Estadio Estadios[], int cantEstadios) RETORNA VACIO
       Entero i
        PARA (i <- 0 HASTA  cantEstadios-1 PASO 1) HACER
            ESCRIBIR(toString(Estadios[i]))
            ESCRIBIR("")//salto
        FIN PARA
    FIN MODULO

    //metodo para buscar la posicion del estadio
    MODULO  posEstadio(Estadio Estadios[], ENTERO numEstadio, ENTERO i, Entero
cantEstadios) RETORNA ENTERO
        ENTERO pos<- 0;
        SI(i < cantEstadios) entonces
            SI (numEstadio == getNumero(Estadios[i])) {
                pos <- i
            SINO
                pos<-posEstadio(Estadios, numEstadio, i + 1, cantEstadios)
            FINSI
        FINSI

        retorna pos
    FIN MODULO

    //metodo que retorna una cadena sin vocales y sin espacios
    MODULO  abreviatura(Estadio Estadios[], int numEstadio, int pos) RETORNA TEXTO
        TEXTO nomOficial, nomModificado <- ""
        nomOficial <- aMinuscula(getNombre(Estadios[numEstadio]))//pasamos el nombre al
minuscula para no tener problemas
        nomOficial <-removerEspacios(nomOficial)
        SI (pos <longitud(nomOficial) ENTONCES//buscamos las vocales y los espacios y los
saltamos
            SI (caracPos(pos,nomOficial) == 'a' ||caracPos(pos,nomOficial) == 'e' ||
caracPos(pos,nomOficial) == 'i'
                    ||caracPos(pos,nomOficial) == 'o' || caracPos(pos,nomOficial) == 'u'||
caracPos(pos,nomOficial) == ' ')
```

```
            nomModificado <- abreviatura(Estadios, numEstadio, pos + 1);
          SINO
            nomModificado <- nomModificado + caracPos(pos,nomOficial) +
abreviatura(Estadios, numEstadio, pos + 1);
          FINSI
        FINSI
        retorna  nomModificado
    FIN MODULO

    //metodo vuelve la primer letra de la cadena  mayuscula
    MODULO  Mayuscula(TEXTO cadena) RETORNA TEXTO
        TEXTO nuevaCadena <- ""
        caracter letra
        ENTERO i
        PARA( i <- 0; HASTA longitud(cadena)-1; i++) {//concatenamos una nueva cadena con
la primerletra mayuscula
          SI (i == 0) ENTONCES                    //separamos la primer letra
            letra <- caracEnPos(i,cadena)
            nuevaCadena <- "" + letra
            nuevaCadena <- aMayuscula(nuevaCadena);//con el metodo toUpperCase la
volvemos mayuscula
          SINO
            nuevaCadena <- nuevaCadena +caracEnPos(i,cadena)
          FIN SI
        FIN PARA
        retorna nuevaCadena
    FIN MODULO

    //Insercion
    //Ascendentemente
   MODULO  insercionA(Estadio Estadios[], ENTERO cantEstadios) RETORNA VACIO
       ENTERO i
        PARA ( i <- 1 HASTA cantEstadios-1 PASO 1) HACER
          reubicarA(Estadios, i)
        FIN PARA

    FIN MODULO

    MODULO  reubicarA(Estadio Estadios[], ENTERO i) RETORNA VACIO
        ENTERO j;
        Estadio auxEstadio<- crear  Estadio(0)
        auxEstadio <- Estadios[i]
        j <- i - 1;
        MIENTRAS (j >= 0 AND Estadios[j].CompareTo(auxEstadio) > 0) {
          Estadios[j + 1] <- Estadios[j]
```

```
        j<-J-1
    FIN MIENTRAS
    Estadios[j + 1] <-auxEstadio
 FIN MÓDULO

 //Descendentemente
MODULO  insercionD(Estadio Estadios[], int cantEstadios) RETORNA VACIO
        Entero i
    PARA ( i <-1 HASTA cantEstadios-1 PASO 1) {
        reubicarD(Estadios, i);
    FIN PARA
 FIN MODULO

 MODULO reubicarD(Estadio Estadios[], ENTERO i) RETORNA VACIO
    ENTERO j;
    Estadio auxEstadio<- crear Estadio(0);
    auxEstadio <- Estadios[i]
    j <- i - 1;
    MIENTRAS (j >= 0 AND  Estadios[j].CompareTo(auxEstadio) < 0) {
        Estadios[j + 1] <- Estadios[j];
        J<-J-1;
    }
    Estadios[j + 1] <- auxEstadio;
}

//Quicksort
// Ascendentemente
MODULO  quicksortA(Estadio Estadios[], int izq, int der) RETORNA VACIO
    Estadio pivoteEstadio <- crear  Estadio(0);
    Estadio auxEstadio <-  crear Estadio(0);
    pivoteEstadio <-  Estadios[(izq+der)/2]         //elegimos un elemento como pivote
    Entero i <- izq;                          //  i analiza de izquierda a derecha
    Entero j <- der;                           //  j analiza de derecha a izquierda

    MIENTRAS (i < j) HACER                          // mientras no se crucen las búsquedas
        MIENTRAS  (Estadios[i].CompareTo(pivoteEstadio) <= 0 AND  i < j)HACER
            i<-I+1;                          // busca elemento mayor que pivote
        FIN MIENTRAS
        MIENTRAS(Estadios[j].CompareTo(pivoteEstadio) > 0) HACER
            i<-I-1;                          // busca elemento menor que pivote
        FIN MIENTRAS
        SI (i < j) ENTONCES                     //si las iteradores no se cruzan
            auxEstadio <- Estadios[i]       // intercambia elementos
            Estadios[i] <- Estadios[j]
            Estadios[j] <- auxEstadio
```

```
        FINSI
      FIN MIENTRAS

      Estadios[izq] <- Estadios[j]              // se coloca el pivote de forma en donde los
elementos
      Estadios[j] <- pivoteEstadio;              // menores a su esten a su izquierda y los
mayores a su derecha
      //repetimos procesos con los subconjunto restantes
      SI (izq < j - 1) ENTONCES
        quicksortA(Estadios, izq, j - 1)       // ordenamos subarray izquierdo
      FIN SI

      SI (j + 1 < der) ENTONCES
        quicksortA(Estadios, j + 1, der)       // ordenamos subarray derecho
      FIN SI
   FIN MODULO

   //Descendentemente
   MODULO  quicksortD(Estadio Estadios[], ENTERO izq, ENTERO der) RETORNA VACIO
      Estadio pivoteEstadio <- crear Estadio(0);
      Estadio auxEstadio <- crear Estadio(0);
      pivoteEstadio <-Estadios[izq]
      int i <- izq;
      int j <- der;

      MIENTRAS (i < j) HACER
        MIENTRAS (Estadios[i].CompareTo(pivoteEstadio) >= 0 && i < j) HACER
          i<-i+1;
        FIN MIENTRAS
        MIENTRAS (Estadios[j].CompareTo(pivoteEstadio) < 0) HACER
          j<-j-1;
        FIN MIENTRAS
        SI(i < j) ENTONCES
          auxEstadio <- Estadios[i]
          Estadios[i] <- Estadios[j]
          Estadios[j] <- auxEstadio
        FINSI
      FIN MIENTRAS
      Estadios[izq] <- Estadios[j];
      Estadios[j] <- pivoteEstadio;

      SI (izq < j - 1) ENTONCES
        quicksortD(Estadios, izq, j - 1)
      FINSI
      SI(j + 1 < der) ENTONCES
```

```
        quicksortD(Estadios, j + 1, der);
    FIN SI
FIN MODULO
```