

**Análisis y diseño de Algoritmos**

Proyecto Nº1

Campus Santa Fe

Enrique Lira Martínez A01023351

Profesor: Dr. Víctor Manuel de la Cueva H

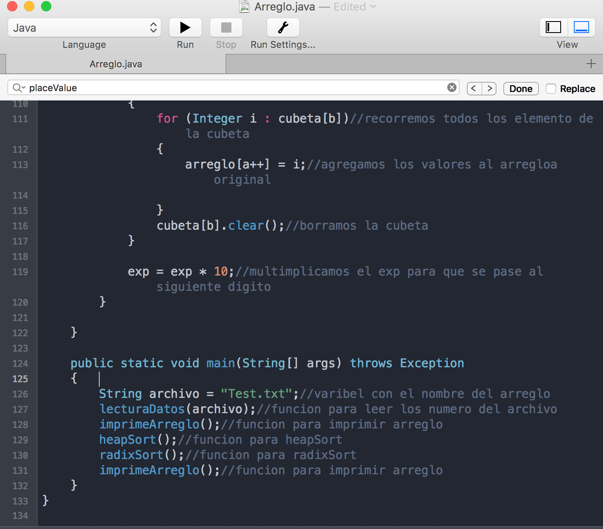
12 de febrero de 2018

**MANUAL DEL USUARIO**

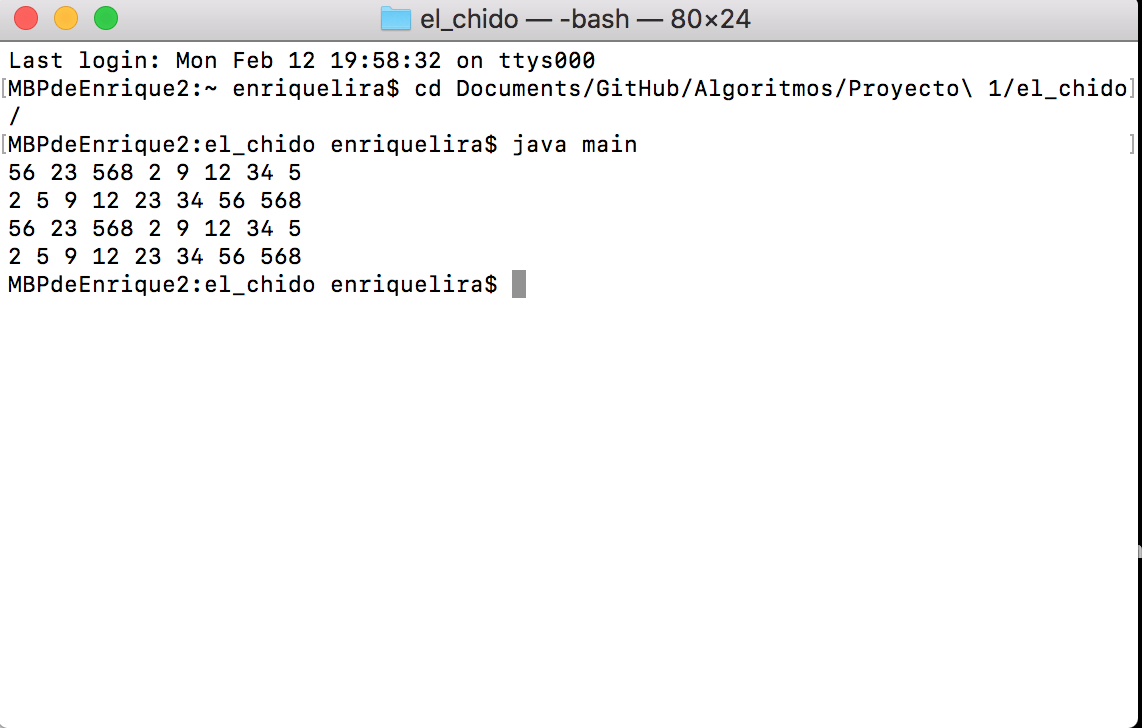
**CONFIGURACIÓN**

1. El usuario debe contar con un IDE o un editor de texto en caso de que el usuario tenga un IDE.

* Si el usuario tiene un IDE deberá de hacer los siguientes pasos, el usuario deberá solamente abrir el archivo en su IDE y correr el programa con el arreglo.java descrito a continuación.



* Si el usuario tiene un editor de texto deberá de hacer los siguientes pasos, el usuario deberá solamente su terminal y deberá de buscar el archivo (Ej. cd Desktop/carpeta……) el usuario deberá de saber dónde se ubica el archivo.



* Después de haber hecho el programa main usted deberá de correr el programa en la terminal con el comando java “nombre del archivo” y dará enter

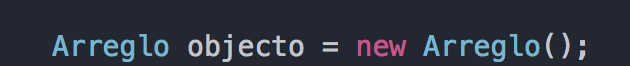
**ESTRUCTURA GENERAL**

**Estructura del main propuesto**

1. Al crear su función main usted deberá poner un throws Exception ,esto se debe a que si no encuentra un archivo regresara un error ,continuación habrá un ejemplo.



1. Primero debemos definir una clase de la siguiente manera

.

1. El usuario tendrá que definirá el nombre del archivo lo podrá poner en una variable como se muestra a continuación o directamente en la función lecturaDatos (Nota : dentro de esta función se crea el arreglo con el tamaño especificado en el archivo de texto)



1. A continuación, se podrán hacer uso de la función radi sort o heap sort de la siguientes maneras





1. Si se quiere imprimirá el arreglo se deber usar la siguiente función



**ALGORITMO**

Se crearon siguientes la funciones:

* void lecturaDatos(String archivo)
* void imprimeArreglo()
* void heapSort()
* void MAX\_heapify(int arr[], int n, int i)
* void radixSort()

cada función tiene un funcionamiento diferente que a continuación se describirá su algoritmo

* lecturaDatos(String archivo)

Paso 1: Inicio

Paso 2: se inicializa un objeto File y un buffer para leer los datos

Paso 3:while (readline ¡= NULL)

Si (contador = 0)

Tamaño = readline

Else

Arreglo[contador-1]=readline

I++

Paso 4: Fin

* heapSort()

Paso 1:inicio

Paso 2:for (tamaño/2-1 hasta que i=0 i--)

MAX\_heapify(arreglo, tamaño, i)

Paso 3:for (tamaño-1 hasta que i>=0 i--)

swap = arreglo[0];

arreglo[0] = arreglo[i];

arreglo[i] = swap;

MAX\_heapify(arreglo, i, 0)

* void MAX\_heapify(int arr[], int n, int i)

paso 1:incio

Paso 2:root=i ,hijoI=2\*1+1,hijoD=2\*1+2

Paso 2:si (hijoI < tamaño y arreglo[hijoI]>arreglo [root]

Root = hijoI

Paso 4: si (hijoD < tamaño y arreglo[hijoD]>arreglo [root]

Root = hijoD

Paso 5:si (root es difernte de i )

swap = arreglo[i];

arreglo[i] = arreglo[root];

arreglo[root] = swap;

MAX\_heapify(arreglo,tamaño, root)

Paso 3:Final

* void radixSort()

paso 1:inicio

Paso 2:arrayList cubeta con 10 elmentos

Paso 3:for (0 hasta el tamaño de arreglo)

Cubeta[i]=arraylist

Paso 4: while (¡findmax)

for (0 hasta que no haya elementos en el arreglo)

cubeta[(elemnto /exp )%10].add

if (Maxfind se encuntra )

maxfind cambia a false

paso 5: for (0 hasta 10)

for (0 hasta el tamaño de arreglo)

arreglo=i

limpiamos la cubeta

Paso 3:Final

**Descripción Técnica**

**Función lecturaDatos(String archivo)**

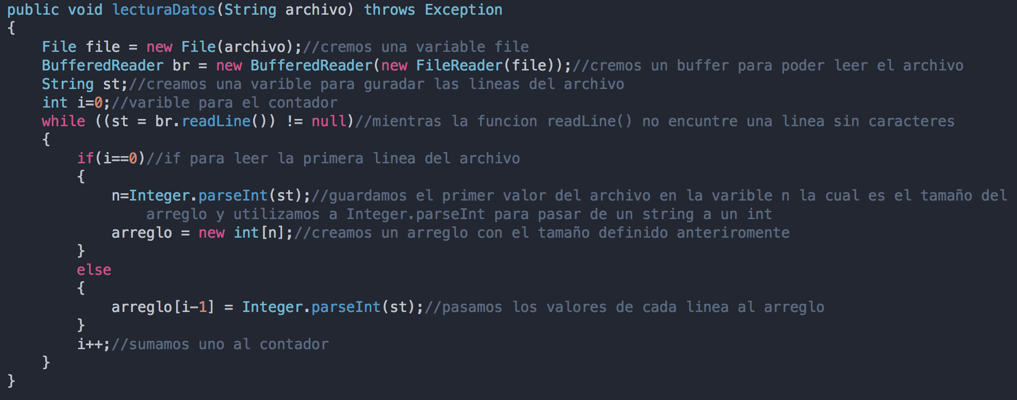
**Argumentos:** la función tiene como parámetro un string el cual será el nombre del archivo

**Variables:**

* file = archivo del texto
* n = tamaño del arreglo (se estable con relación al primer dato ingresado en el archivo de texto )
* int arreglo []

**Lógica:**

La función recibe el nombre del archivo y este lo leerá línea por línea, para poder ingresarlo a un arreglo primero debemos de crear el arreglo , y primero se realizan if con un contador si es 0 la primera línea será el tamaño del arreglo y se creara el arreglo si no se van a ir agregando los datos al arreglo creado.

****

**Función MAX\_heapify**

**Argumentos:** la función recibe el arreglo ,el tamaño y la posición del nodo padre

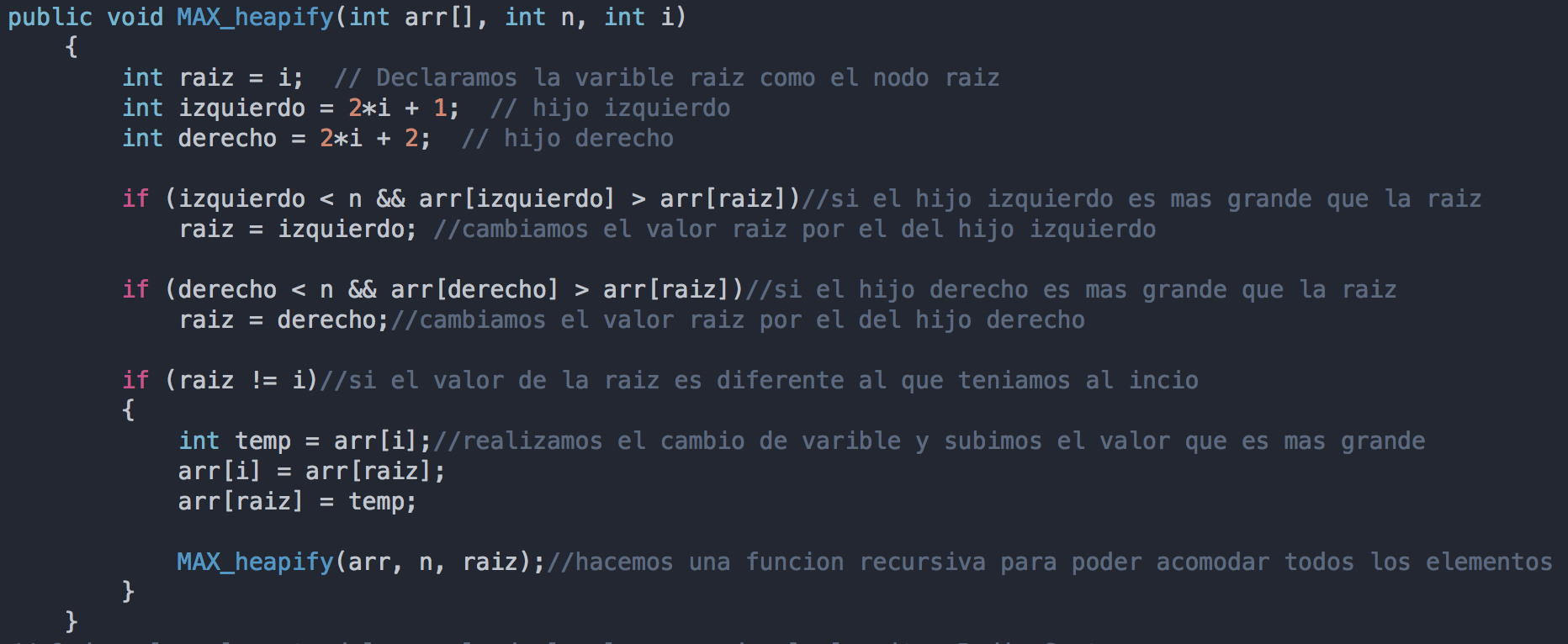
**Variables:**

* arreglo
* tamaño
* posición del padre

**Lógica:**

La función recibe el arreglo que se desea ordenar y lo primero que hara será crear los nodos que seria la raíz y los hijos izquierdo y derecho ,se realizan dos condiciones ,la primera es si el hijo izquierdo es mas grande que la raíz ,si es cierto entonces se cambara el valor de la raíz y viceversa con el hijo derecho

Si el valor de root fue cambiado por los hijos este deberá de realizar un swap y entrar en la función haciendo una recursión hasta que quede un su lugar apropiado

****

**void heapSort()**

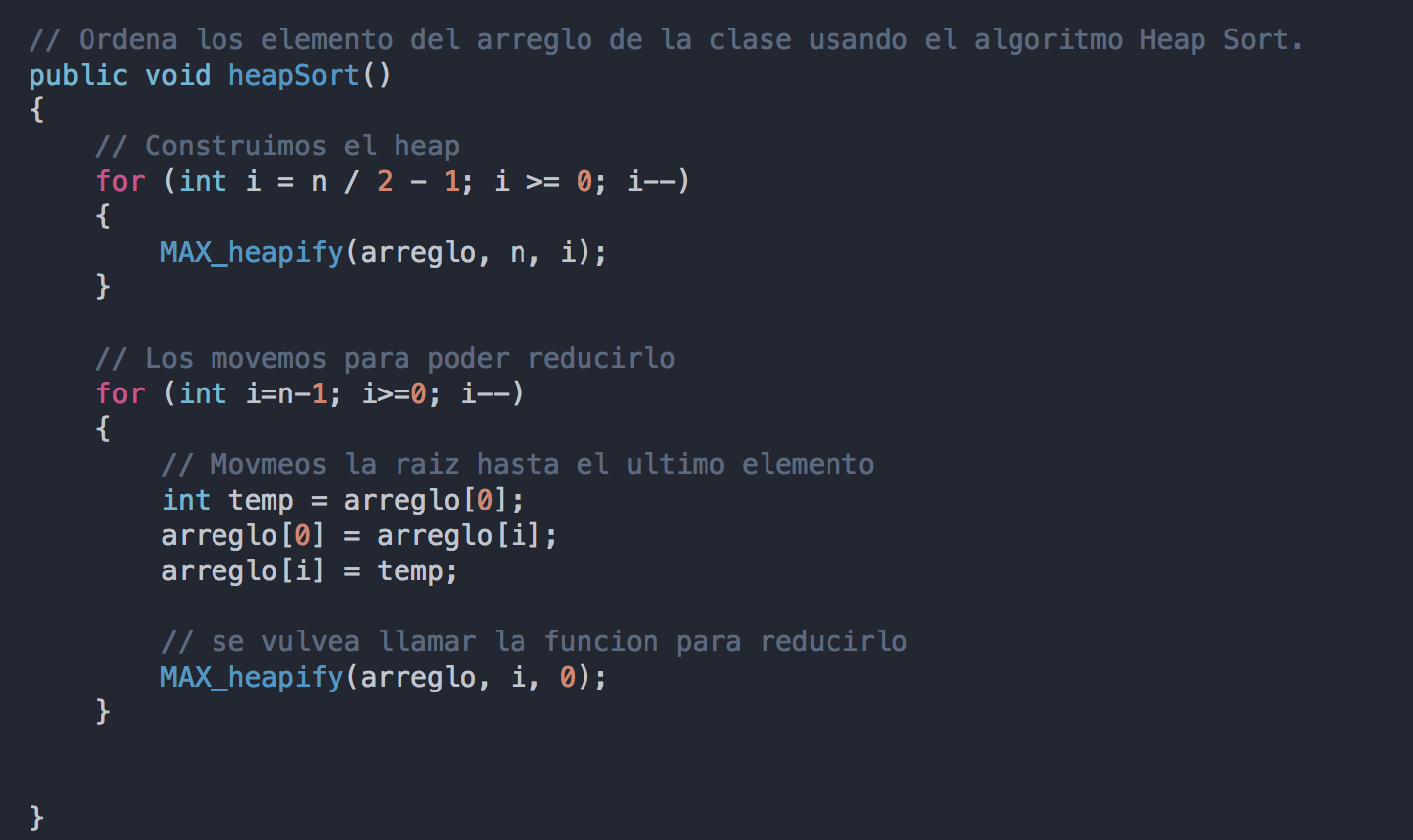
**Argumentos:** No recibe valores

**Variables:**

* arreglo :los elementos del archivo
* n =tamaño del arreglo

**Lógica:**

Primero construimos el heap y lo llenamos de de la mitad del arreglo , después reducimos el arreglo y volvemos llamar a la funcino MAX\_heapify para poder reducirlo

****

**Función void radixSort()**

**Argumentos:** No recibe valores

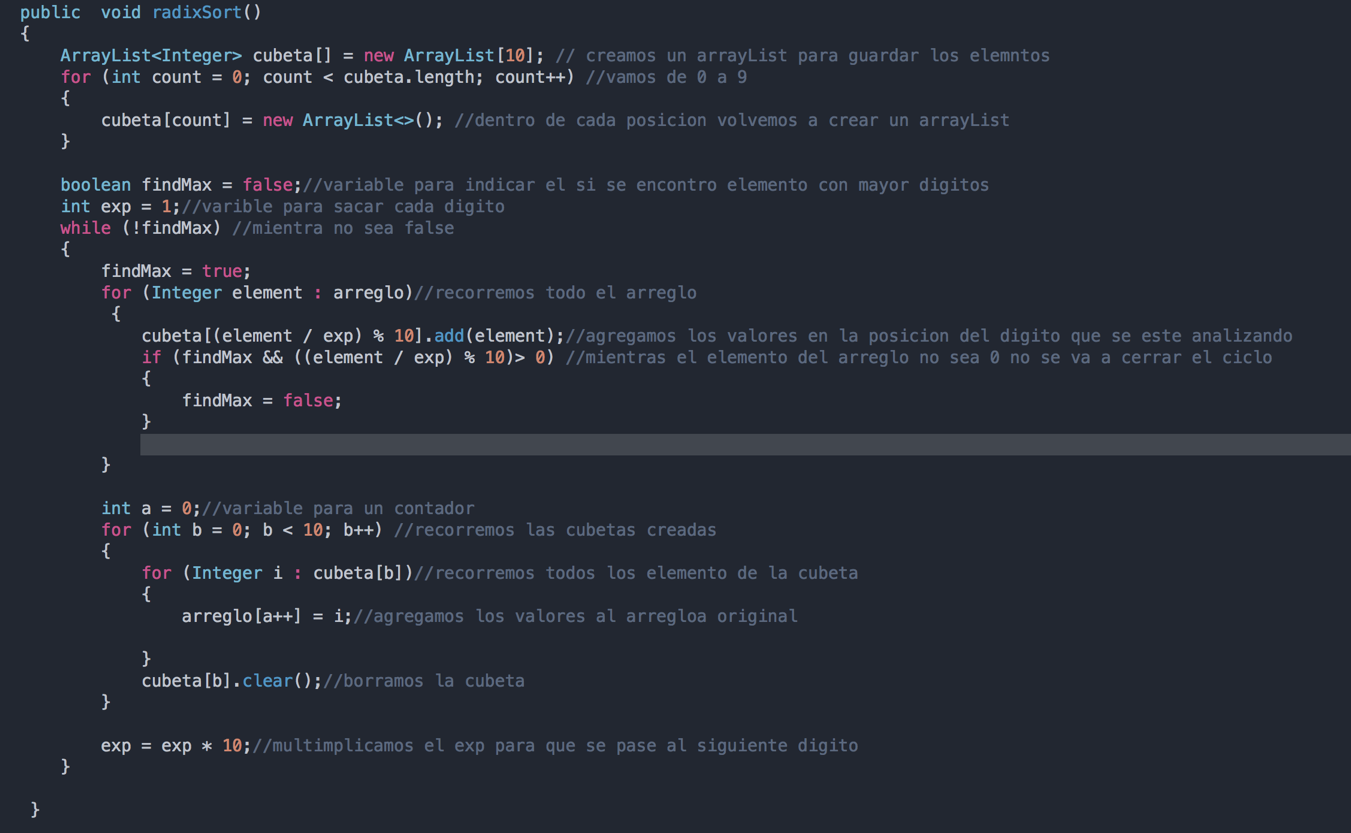
**Variables:**

arreglo :los elementos del archivo

n =tamaño del arreglo

**Lógica:**

Lo primero que se realizara en esta función es la creación de los arrayList , lo primero será definir un arraylist de 10 elementos y dentro de ellos crearemos un arraylist por cada posición ,después se sacra los dígitos de cada número del arreglo previamente creado ,el ciclo while servirá para poder definir si se a encontrado el elemento con mas dígitos y dentro de este while se irán guardando los elementos el arreglo en el arrayList dependiendo del digito al terminar de recorrer el arreglo , recorrernos las cubetas y guardemos los elementos creados en la variable arreglo y al final se multiplicara el exponente por 10 para sacar el siguiente elemento

****

**Función imprimrirPila**

**Argumentos:** No recibe valores

**Variables:**

* tamaño: Es la posición del ultimo valor de la pila

**Lógica:**

Lo que se hace esta función es imprimir el valor actual que contiene el tope de

nuestro arreglo.

