



Tecnológico de Monterrey

Análisis y diseño de Algoritmos

Proyecto N°1

Campus Santa Fe

Enrique Lira Martínez A01023351

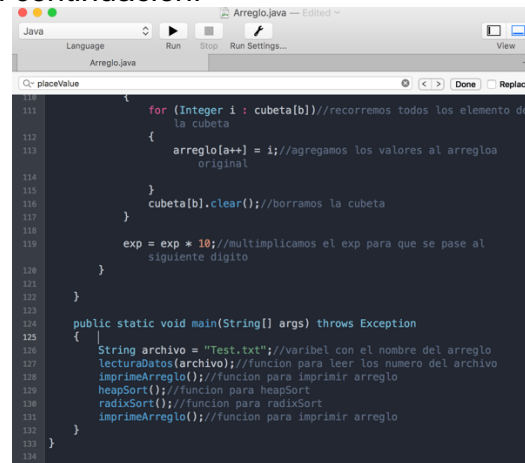
Profesor: Dr. Víctor Manuel de la Cueva H

12 de febrero de 2018

MANUAL DEL USUARIO

CONFIGURACIÓN

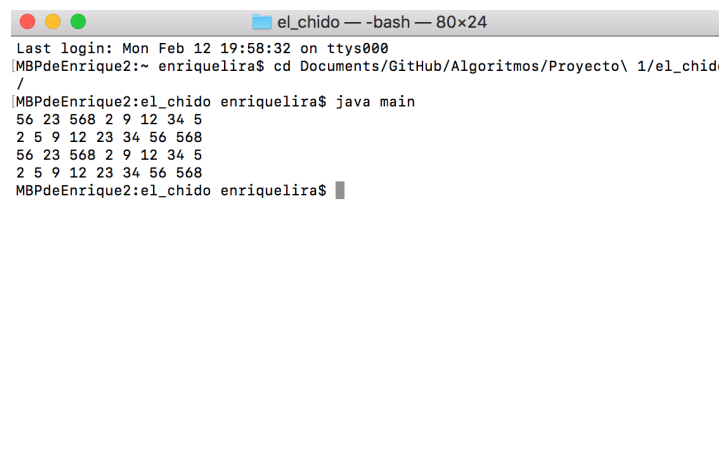
1. El usuario debe contar con un IDE o un editor de texto en caso de que el usuario tenga un IDE.
 - Si el usuario tiene un IDE deberá de hacer los siguientes pasos, el usuario deberá solamente abrir el archivo en su IDE y correr el programa con el arreglo.java descrito a continuación.



```
Arreglo.java -- Edited
Java
Language
Run
Stop
Run Settings...
View

C:\placeValue
110
111     for (Integer i : cubeta[b])//recorremos todos los elemento de
112     {
113         arreglo[a++] = i;//agregamos los valores al arreglo
114         original
115     }
116     cubeta[b].clear();//borramos la cubeta
117 }
118
119     exp = exp * 10;//multiplicamos el exp para que se pase al
120     siguiente digito
121 }
122 }
123
124 public static void main(String[] args) throws Exception
125 {
126     String archivo = "Test.txt";//varibel con el nombre del arreglo
127     lecturaDatos(archivo);//funcion para leer los numero del archivo
128     imprimeArreglo();//funcion para imprimir arreglo
129     heapSort();//funcion para heapSort
130     radixSort();//funcion para radixSort
131     imprimeArreglo();//funcion para imprimir arreglo
132 }
133 }
134 }
```

- 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.



```
el_chido -- -bash -- 80x24
Last login: Mon Feb 12 19:58:32 on ttys000
[MBPdeEnrique2:~ enrique1ira$ cd Documents/GitHub/Algoritmos/Proyecto\ 1/el_chido
/]
[MBPdeEnrique2:el_chido enrique1ira$ java main
56 23 568 2 9 12 34 5
2 5 9 12 23 34 56 568
56 23 568 2 9 12 34 5
2 5 9 12 23 34 56 568
MBPdeEnrique2:el_chido enrique1ira$ ]
```

- 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.

```
public static void main(String[] args) throws Exception
```

2. Primero debemos definir una clase de la siguiente manera

```
Arreglo objeto = new Arreglo();
```

3. 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)

```
objeto.lecturaDatos("Test.txt");
```

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

```
objeto.heapSort();  
objeto.radixSort();
```

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

```
objeto.imprimeArreglo();
```

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*i+1,hijoD=2*i+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 diferente 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 elementos
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[(elemento / exp )%10].add
    if (Maxfind se encuentra )
        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 establece 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.

```
public void lecturaDatos(String archivo) throws Exception
{
    File file = new File(archivo); //cremos una variable file
    BufferedReader br = new BufferedReader(new FileReader(file)); //cremos un buffer para poder leer el archivo
    String st; //cremos una variable para guardar las lineas del archivo
    int i=0; //variable para el contador
    while ((st = br.readLine()) != null) //mientras la funcion readLine() no encuentre una linea sin caracteres
    {
        if(i==0) //if para leer la primera linea del archivo
        {
            n=Integer.parseInt(st); //guardamos el primer valor del archivo en la variable n la cual es el tamaño del
            arreglo y utilizamos a Integer.parseInt para pasar de un string a un int
            arreglo = new int[n]; //creamos un arreglo con el tamaño definido anteriormente
        }
        else
        {
            arreglo[i-1] = Integer.parseInt(st); //pasamos los valores de cada linea al arreglo
        }
        i++; //sumamos uno al contador
    }
}
```

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

```
public void MAX_heapify(int arr[], int n, int i)
{
    int raiz = i; // Declaramos la variable raiz como el nodo raiz
    int izquierdo = 2*i + 1; // hijo izquierdo
    int derecho = 2*i + 2; // hijo derecho

    if (izquierdo < n && arr[izquierdo] > arr[raiz]) //si el hijo izquierdo es mas grande que la raiz
        raiz = izquierdo; //cambiamos el valor raiz por el del hijo izquierdo

    if (derecho < n && arr[derecho] > arr[raiz]) //si el hijo derecho es mas grande que la raiz
        raiz = derecho; //cambiamos el valor raiz por el del hijo derecho

    if (raiz != i) //si el valor de la raiz es diferente al que teniamos al incio
    {
        int temp = arr[i]; //realizamos el cambio de variable y subimos el valor que es mas grande
        arr[i] = arr[raiz];
        arr[raiz] = temp;

        MAX_heapify(arr, n, raiz); //hacemos una funcion recursiva para poder acomodar todos los elementos
    }
}
```

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

```
// Ordena los elemento del arreglo de la clase usando el algoritmo Heap Sort.
public void heapSort()
{
    // Construimos el heap
    for (int i = n / 2 - 1; i >= 0; i--)
    {
        MAX_heapify(arreglo, n, i);
    }

    // Los movemos para poder reducirlo
    for (int i=n-1; i>=0; i--)
    {
        // Movmeos la raiz hasta el ultimo elemento
        int temp = arreglo[0];
        arreglo[0] = arreglo[i];
        arreglo[i] = temp;

        // se vulvea llamar la funcion para reducirlo
        MAX_heapify(arreglo, i, 0);
    }
}
```

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 dígito al terminar de recorrer el arreglo , recorreremos 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

```

public void radixSort()
{
    ArrayList<Integer> cubeta[] = new ArrayList[10]; // creamos un arrayList para guardar los elemntos
    for (int count = 0; count < cubeta.length; count++) //vamos de 0 a 9
    {
        cubeta[count] = new ArrayList<>(); //dentro de cada posicion volvemos a crear un arrayList
    }

    boolean findMax = false; //variable para indicar el si se encontro elemento con mayor digitos
    int exp = 1; //variable para sacar cada digito
    while (!findMax) //mientras no sea false
    {
        findMax = true;
        for (Integer element : arreglo) //recorremos todo el arreglo
        {
            cubeta[(element / exp) % 10].add(element); //agregamos los valores en la posicion del digito que se este analizando
            if (findMax && ((element / exp) % 10) > 0) //mientras el elemento del arreglo no sea 0 no se va a cerrar el ciclo
            {
                findMax = false;
            }
        }

        int a = 0; //variable para un contador
        for (int b = 0; b < 10; b++) //recorremos las cubetas creadas
        {
            for (Integer i : cubeta[b]) //recorremos todos los elemento de la cubeta
            {
                arreglo[a++] = i; //agregamos los valores al arreglo original
            }
            cubeta[b].clear(); //borramos la cubeta
        }

        exp = exp * 10; //multiplicamos el exp para que se pase al siguiente digito
    }
}

```

Función imprimirPila

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.

```

// Imprime el arreglo
public void imprimeArreglo()
{
    for (int i=0; i<n; ++i) //ciclo que pasa por todos los elementos del arreglo
        System.out.print(arreglo[i]+" "); //imprime cada elemento
    System.out.println(); //salto de linea
}

```