Universidad de San Buenaventura

Facultad ingeniería de sistemas

Taller 4 Corte 3

Análisis de algoritmos

Presenta:

Juan Felipe Hurtado Villani

Cristian Apraez

Samuel Martínez

Punto 1:

Para comenzar con este punto, primero planteare la teoría y luego hare un pequeño ejemplo antes del desarrollo del punto:

The stooge sort es un algoritmo recursivo de clasificación u ordenamiento, definido de la siguiente manera:

Paso 1: si el valor en el índice 0 es mayor que el valor en el último índice, cámbielos.

Paso 2: recursivamente,

a) Stooge clasifica las 2/3 iniciales de la matriz.

b) Stooge clasifica los últimos 2/3 de la matriz.

c) Stooge clasifica las 2/3 iniciales nuevamente para confirmar.

Ejemplo:

Entrada: 2 4 5 3 1

Salida: 1 2 3 4 5

Explicación:

Inicialmente, intercambie 2 y 1 siguiendo el paso 1 anterior.

1 4 5 3 2

Ahora, ordena de forma recursiva 2/3 de los elementos iniciales.

1 4 5 3 2

1 3 4 5 2

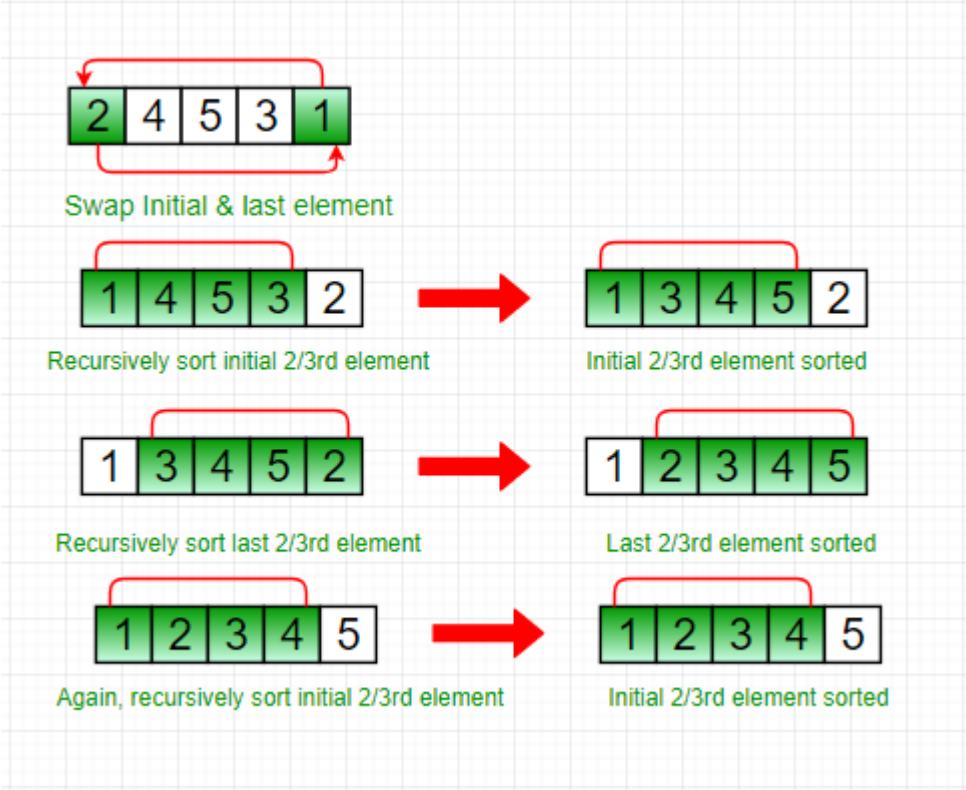
Luego, ordena de forma recursiva los últimos 2/3 de los elementos.

1 3 4 5 2

1 2 3 4 5

Nuevamente, ordene las 2/3 iniciales de los elementos para confirmar que los datos finales están ordenados.

1 2 3 4 5



Explicado este ejemplo, pasaremos al código:

```python
# Programa en Python para implementar el ordenamiento de Stooge

def stoogesort(arr, l, h):
    if l >= h:
        return


    # El primer elemento es menor que el ultimo, cambialos
    if arr[l]>arr[h]:
        t = arr[l]
        arr[l] = arr[h]
        arr[h] = t

    # Si hay más de 2 elementos en
    # la matriz
    if h-l + 1 > 2:
        t = (int)((h-l + 1)/3)

        # Ordene de forma recursiva los primeros 2/3 elementos
        stoogesort(arr, l, (h-t))

        # Ordena recursivamente los últimos 2/3 elementos
        stoogesort(arr, l + t, (h))

        # Ordene de forma recursiva los primeros 2/3 elementos
# nuevamente para confirmar
        stoogesort(arr, l, (h-t))


# derivador de la función de ordenamiento
arr = [2, 4, 5, 3, 1]
n = len(arr)

stoogesort(arr, 0, n-1)

for i in range(0, n):
    print(arr[i], end = ' ')

```

```
PS C:\Users\Juan Hurtado\Dropbox\Mi PC (DESKTOP-CU6IIF6)\Desktop>  c:
c:\Users\Juan Hurtado\.vscode\extensions\ms-python.python-2021.10.136
plement stooge sor.py'
1 2 3 4 5
PS C:\Users\Juan Hurtado\Dropbox\Mi PC (DESKTOP-CU6IIF6)\Desktop>
```

Ahora con los datos que nos piden, haremos dos ejemplos:

```
# derivador de la función de ordenamiento
arr = [2, 4, 5, 3, 1, 10, 5, 9, 8, 15]
n = len(arr)
```

```
plement stooge sor.py'
1 2 3 4 5 5 8 9 10 15
PS C:\Users\Juan Hurtado\Dropbox\Mi PC (DESKTOP-CU6IIF6)\Desktop>
```

```
# derivador de la función de ordenamiento
arr = [2, 4, 5, 3, 1, 10, 5, 9, 8, 15, 1, 20, 1050, 105, 522, 139, 589, 965, 845, 1552, 19657, 3241354, 654654, 1324874, 7261, 3748646, 321387, 87968, 53487, 377431, 387315]
n = len(arr)
```

```
PS C:\Users\Juan Hurtado\Dropbox\Mi PC (DESKTOP-CU6IIF6)\Desktop>  c:; cd 'c:\Users\Juan Hurtado\Dropbox\Mi PC (DESKTOP-CU6IIF6)\Desktop';
c:\Users\Juan Hurtado\.vscode\extensions\ms-python.python-2021.10.1365161279\pythonFiles\lib\python\debugpy\launcher' '57702' '--' 'c:\Use
plement stooge sor.py'
1 1 2 3 4 5 5 8 9 10 15 20 105 139 522 589 845 965 1050 1552 7261 19657 53487 87968 321387 377431 387315 654654 1324874 3241354 3748646
PS C:\Users\Juan Hurtado\Dropbox\Mi PC (DESKTOP-CU6IIF6)\Desktop>
```

La complejidad del tiempo de ejecución del stooge sort se puede escribir como,

T (n) = 3T (3n / 2) +? (1)

La solución de la recurrencia anterior es $O\left(n^{\left(\frac{log^3}{log^{1.5}}\right)}\right) = O\ (n^{2.709})$, por lo tanto, es más lenta que incluso el bubble sort (n ^ 2).

Punto 2:

Para el divide y vencerás tenemos que:

1. Definir el tamaño del vector
2. Nuestro componente “elVector” es un arreglo de N elementos de tipo entero.
3. Cargamos el vector con valores aleatorios.
4. Imprimimos los valores del vector.
5. Llamamos al método “hallarModa” e imprimimos el valor devuelto por el mismo.

```java
public class DivideYVencerasModa {
    Run | Debug
    public static void main (String [] args) {
        System.out.println ("Busqueda de la moda en un vector de N elementos.\n");
        System.out.println ("Aplicando tecnicas de divide y venceras.\n");
        System.out.println ("============================================================\n");
        int N = 50;                                 //Define el tamaño del vector
        int [] elVector = new int [N];              //elVector es un arreglo de N elementos de tipo enteros
        cargarVector(elVector);                     //Cargamos el vector con valores aleatorios
        imprimirVector(elVector);                   //Imprimimos los valores del vector
        /*Llamamos al metodo hallarModa e imprimimos el valor devuelto*/
        System.out.println ("\n\nEl valor que mas se repite es: " + hallarModa (elVector, 0 , elVector.length-1));
    }
```

```java
    /**
     * Metodo para hallar la moda en un vector de enteros
     * Utilizando la tecnica Divide y Venceras
     * @param a
     * @param prim
     * @param ult
     * @return
     */
    public static int hallarModa (int a[], int prim, int ult) {
        int i, frec, maxfrec, moda;
        if (prim == ult) return a[prim];
        moda = a[prim];
        maxfrec = Frecuencia(a, a[prim], prim, ult);
        for (i = prim + 1; i<=ult; i++) {
            frec = Frecuencia (a, a[i], i, ult);
            if (frec > maxfrec) {
                maxfrec = frec;
                moda = a[i];
            }
        }
        return moda;
    }
```

```java
    /**
     * Metodo para calcular el numero de veces que se repite un elemento
     * Utilizado por el metodo hallarModa
     * @param a
     * @param p
     * @param prim
     * @param ult
     * @return
     */
    public static int Frecuencia (int a[], int p, int prim, int ult) {
        int i, suma;
        if (prim > ult) return 0;
        suma = 0;
        for (i = prim; i<= ult; i++)
            if(a[i] == p)
                suma++;
        return suma;
    }
```

```java
    /**
     * Metodo para cargar el vector con valores int aleatorios
     * @param a
     */
    public static void cargarVector (int a[]) {
        Random r = new Random();                        //Variable para el numero generado aleatoriamente
        for (int i = 0; i< a.length; i++){      //Cargamos el vector con valores aleatorios
            a [i] = (r.nextInt(100));           //Genera y carga valores aleatorios en elVector
        }
    }
    /**
     * Metodo para imprimir el vector
     * @param a
     */
    public static void imprimirVector (int a[]) {
        System.out.println("\nEl vector es:");
        for (int i = 0; i < a.length; i++) {
            System.out.printf( "%d " , (a[i]) );
        }
    }
}
```

```
Busqueda de la moda en un vector de N elementos.

Aplicando tecnicas de divide y venceras.

============================================================


El vector es:
75 7 12 2 69 27 70 18 46 17 99 40 12 6 31 18 52 8 15 52 43 14 39 31 96 74 67 26 85 35 99 33 14 53 6
4 40 9 64 71 45 96 80 45 54 94 53 79 26 20 85

El valor que mas se repite es: 12
```

```
Busqueda de la moda en un vector de N elementos.

Aplicando tecnicas de divide y venceras.

============================================================


El vector es:
75 99 84 73 66 24 91 47 81 70 51 34 62 40 2 91 66 55 65 35 30 71 33 68 29 48 4 83 40 93 44 78 14 84 11 23 83 84 68 41 83 9 50 15 61 51 94 53 14 14

El valor que mas se repite es: 84
```

La ecuación de recurrencia es la siguiente:

$$T(n) <= \begin{cases} O(1) \text{ si } n = 1 & \text{//if (prim == ult) return a[prim];} \\ T(n/2) + O(n) & \end{cases}$$

La complejidad de la función Frecuencia es O(n), entonces la complejidad para calcular la moda está en orden de O(n^2)

Punto 3:

```python
import time
import numpy as np

start_time = time.time()*1000
# MERGE SORT
def mergesort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]
        mergesort(L)
        mergesort(R)

        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1


def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()


arr = []
for x in range(500):
    arr.append(np.random.randint(1,50))

print("Given array is", end="\n")
printList(arr)
mergesort(arr)
print("Sorted array is: ", end="\n")
printList(arr)
end_time=time.time()*1000

print("--- %s seconds ---" % (end_time - start_time))
```

```python
import time
import numpy as np

start_time = time.time()*1000

def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()

def insertionSort(arr):
    for i in range(1, len(arr)):

        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

arr = []
for x in range(500):
    arr.append(np.random.randint(1,50))

print("ANTES")
printList(arr)
insertionSort(arr)
print("DESPUES")
printList(arr)

end_time=time.time()*1000

print("--- %s seconds ---" % (end_time - start_time))
```

MERGE-SORT

Entrada	Tiempo Real (seg)	Complejidad(θ(nLog(n)))	Constantes
10	0.0	33.21928	0.0
10	0.0	33.21928	0.0

10	0.0	33.21928	0.0
50	0.0	282.19281	0.0
50	0.0	282.19281	0.0
50	0.0	282.19281	0.0
100	8.001953125	664.38562	0.012044139
100	0.0	664.38562	0.0
100	0.0	664.38562	0.0
500	7.998046875	4482.89214	0.001784126
500	8.062744140625	4482.89214	0.001798558
500	8.1982421875	4482.89214	0,001828784

INSERTION-SORT

Entrada	Tiempo Real (seg)	Complejidad(O(n^2))	Constantes
10	0.0	100	0.0
10	0.0	100	0.0

10	0.99951171875	100	0.00999511
50	0.992431640625	2500	0.000396972
50	1.0029296875	2500	0.000401178
50	0.995361328125	2500	0.000398144
100	1.09716796875	10000	0.000109716
100	2.0	10000	0.0002
100	0.997802734375	10000	0.0000997802
500	16.125244140625	250000	0.00006450097
500	17.06494140625	250000	0.00006825976
500	15.08056640625	250000	0.000060322265