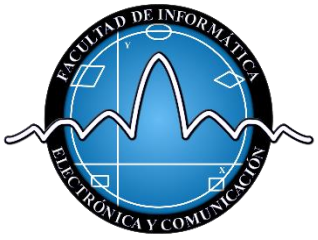


Universidad de Panamá

Facultad de Informática Electrónica y Comunicación

Escuela de Ingeniería en Informática

Materia:

Computabilidad y Complejidad de Algoritmos

Tema:

Divide y Vencerás

Integrantes:

Kaiser Obaldía 8-898-703

Yeny Ortega 8-923-1263

Profesor:

Ayax Mendoza

Fecha de Entrega: 17 de noviembre del 2020

Desarrollo

Desarrollar el problema de divide y vencerás

- Búsqueda Binaria

```
#*****************************************************************************************************
#Funcion de busqueda binaria
def binarySearch(arr,low,high,x):
    if low<=high:
        #encontrar el índice medio de la matriz
        mid=int( (low+high)/2  )

        #comprobando si la x está en el medio o no
        if arr[mid]==x:
            return mid
        elif arr[mid]>x:
            #si x is mayor que la mitad del elemento
            #entonces regresa mid+1 del indice e ignore los elementos de la izquierda
            #Llamando a la funcion de busqueda binaria nuevamente
            return binarySearch(arr,low,mid-1,x)
        else:
            return binarySearch(arr,mid+1,high,x)
    else:
        return -1
#Fin de la funcion
#*****************************************************************************************************

#*****************************************************************************************************
```

Código búsqueda binaria de forma recursiva

```
while opcion != 0:
    if opcion == 1:
        #Resolver el primer problema usando el algoritmo de busqueda binaria
        arr=[10,12,14,19,45,50,55,56,59,60,39]
        size=len(arr)-1
        print(size)
        # tenemos búsqueda de x
        x=50
        #Llamada de función
        result = binarySearch(arr,0,size,x)
        #imprimir el resultado
        if result != -1:
            print("El elemento x se encuentra en el indice %d",result)
        else:
            print("El elemento no se encuentra en el array")
```

Arreglo de prueba

Aplicando el algoritmo de búsqueda binaria lo que hicimos fue utilizar un array para encontrar la posición del elemento dentro del array, ignorando las demás posiciones y elementos de nuestra posición objetiva. Pasos de la función del programa:

1. Creamos una condición que nos permitiera saber si el array de prueba está vacío o no. Si hay algo en el array se divide la posición menor (0) con la posición mayor (10) dando como resultado la mitad del array, y si no el array estaría vacío.
2. Comprobamos si la posición a la mitad del elemento coincide con la posición objetiva, si coincide se imprime esa posición. Si es mayor que la mitad del elemento entonces se regresará todo lo que sigue de la posición objetivo, y aplicando el algoritmo se ignorara los elementos a la izquierda. En caso que la mitad del elemento sea menor será lo contrario.
3. Luego que tenemos nuestro array de prueba y la mitad del array procedemos a imprimir el tamaño total del arrray y definir nuestro elemento de búsqueda (x).
4. Si se coloca como elemento de búsqueda alguna que no se encuentre dentro del array el programa no mostrará el elemento encontrado.

- Método Quicksort

```python
#Inicio de la Funcion quickSort
def partition(array, start, end):
    pivot = array[start]
    low = start + 1
    high = end

    while True:
        # Si el valor actual que estamos viendo es mayor que el pivote
        # está en el lugar correcto (lado derecho del pivote) y podemos movernos hacia la izquierda,
        # al siguiente elemento.
        # También debemos asegurarnos de no haber superado el puntero bajo, ya que
        # indica que ya hemos movido todos los elementos a su lado correcto del pivote
        while low <= high and array[high] >= pivot:
            high = high - 1

        # Proceso opuesto al anterior
        while low <= high and array[low] <= pivot:
            low = low + 1


        # Encontramos un valor para alto y bajo que está fuera de servicio
        # o bajo es más alto que alto, en cuyo caso salimos del ciclo
        if low <= high:
            array[low], array[high] = array[high], array[low]
            #El bucle continua
        else:
            # Salimos del bucle
            break

    array[start], array[high] = array[high], array[start]

    return high
```

Función de partición

```python
def quick_sort(array, start, end):
    if start >= end:
        return

    p = partition(array, start, end)
    quick_sort(array, start, p-1)
    quick_sort(array, p+1, end)
```

Código de quickSort

```python
elif opcion == 2:
    #Resolver el segundo problema usando el algoritmo de Quicksort
    array = [29,99,27,41,66,28,44,78,87,19,31,76,58,88,83,97,12,21,44]
    quick_sort(array, 0, len(array) - 1)
    print("Array ordenado: ", array)
```

Arreglo de prueba

Aplicando el algoritmo de Quicksort lo que realizamos fue utilizar un arreglo con diferentes elementos (números enteros) y ordenarlos de menor a mayor elemento.

Los pasos que hicimos fueron los siguientes:

1. Primero creamos nuestra función de partición, en esta función manejaremos las variable de pivote que será nuestro elemento moderador. Dentro del código se comenta un poco más de como es el funcionamiento de partition.
2. Luego que tenemos nuestro pivote, con la función Quicksort realizaremos el ordenamiento por cada elemento dentro del array. Es decir que el Quicksort realizará los movimientos tanto izquierda como derecha para ir moviendo los elementos de menor a mayor.
3. Y por último creamos nuestro arreglo de prueba para someterlo al algoritmo creado.

- Encontrar el elemento máximo en un array

```
#*************************************************************************
#Inicio de la Función de encuentra el maximo elemento de un array
def arraymax():
    numeros = [3, 5, 10, 8, 5, 45, 55, 100, 200, 2, 7, 8]
    mayor = 0

    for n in numeros:
     if n > mayor:
         mayor = n
    print(mayor)

#*************************************************************************
```

Código de encontrar el elemento máximo de un array

```
elif opcion == 3:
    #Resolver el tercer problema de encontrar el maximo de un arreglo
    arraymax()
```

Arreglo de prueba

En este punto, creamos un array, después creamos la variable mayor donde se guardará el número mayor. Después hacemos un ciclo for para que recorra el array y por último imprimimos el array para saber cuál fue el elemento mayor.

- Merge-Sort

```python
#Inicio de la Función merge_sort
def merge_sort(lista):

    """
    Lo primero que se ve en el psudocódigo es un if que
    comprueba la longitud de la lista. Si es menor que 2, 1 o 0, se devuelve la lista.
    ¿Por que? Ya esta ordenada.
    """
    if len(lista) < 2:
        return lista

    # De lo contrario, se divide en 2
    else:
        middle = len(lista) // 2
        right = merge_sort(lista[:middle])
        left = merge_sort(lista[middle:])
        return merge(right, left)

# Función merge
def merge(lista1, lista2):
    """
    merge se encargara de intercalar los elementos de las dos
    divisiones.
    """
    i, j = 0, 0 # Variables de incremento
    result = [] # Lista de resultado
            else:
                result.append(lista2[j])
                j += 1

        # Agregamos los resultados a la lista
        result += lista1[i:]
        result += lista2[j:]

        # Retornamos el resultados
        return result
    #*************************************************************************
```

Código de Merge_sort

```python
elif opcion == 4:
    #Resolver el cuarto problema usando el algoritmo de Merge-sort
    # Prueba del algoritmo
    lista = [31,3,88,1,4,2,42]
    merge_sort_result = merge_sort(lista)
    print(merge_sort_result)
```

Arreglo de prueba

Aplicando el algoritmo de Merge-sort pudimos ordenar el array de prueba como se muestra en la imagen. Este algoritmo es parecido al Quicksort, sin embargo, este algoritmo lo que hace es dividir a la mitad un arreglo, para ir trabajándolo según las indicaciones que le hemos dado, en este caso será ordenar los elementos del arreglo de menor a mayor valor.

Los pasos parar armar el algoritmo en código fueron los siguientes:

1. Creamos una función de merge_sort (dentro del código se explica un poco más la función de esta) para poder separar a la mitad o por dos arrays el array de prueba. Primero se verifica que ya no esté ordenada, si lo está no se hace nada, y si no se pasa al siguiente paso que es dividir el array de prueba en dos, en nuestro programa lo llamaremos right y left.

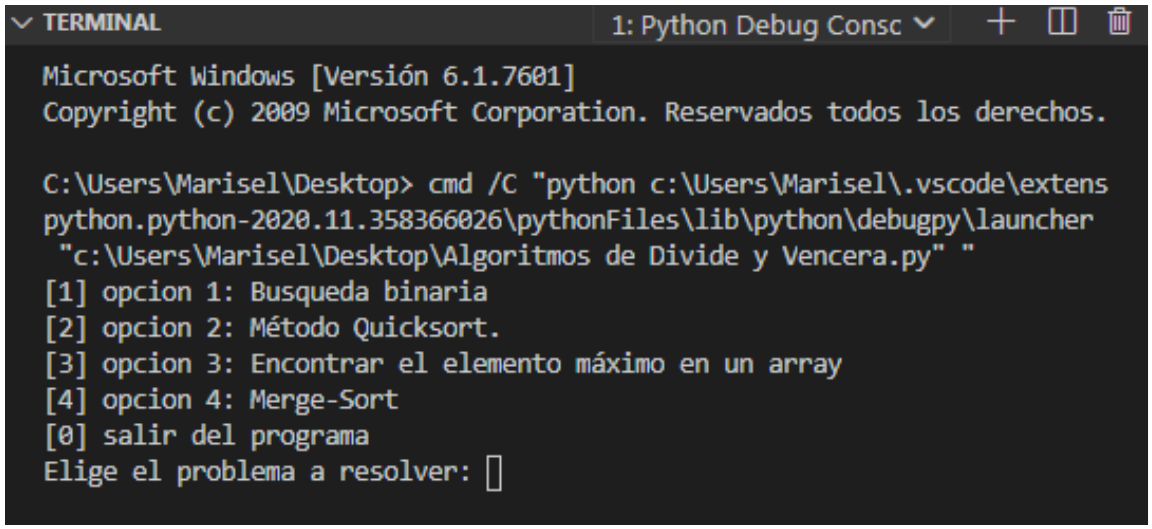
2. Ahora creamos una función llamada merge que se encarga de intercalar los elementos de menor a mayor u ordenado, para ambas listas creadas.

3. Con la función append podemos ir intercalando la lista 1 en la lista 2 (los elementos) o al revés.

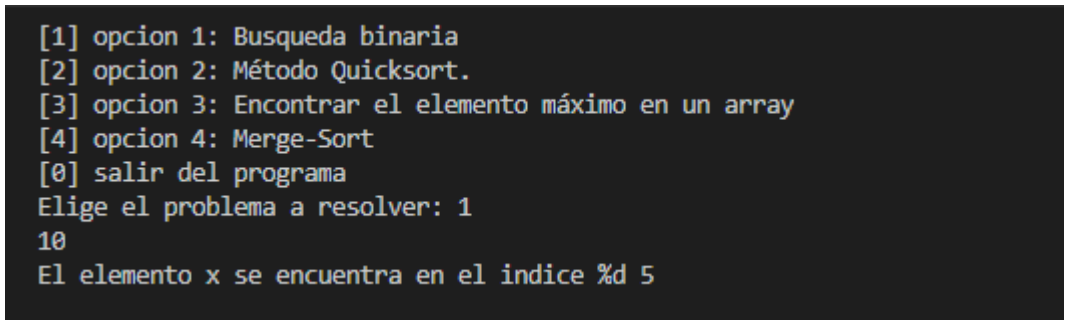
4. Luego procedemos a crear nuestro array de prueba, para llamar la función de merge_sort con la lista de resultados.

Resultados

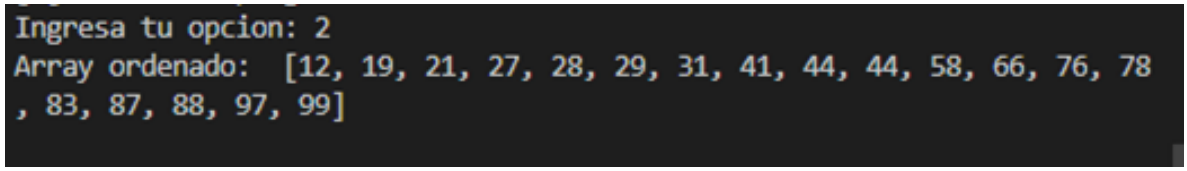
- Menú de soluciones de problemas



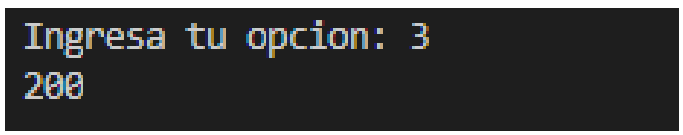
- Opción 1: Búsqueda binaria



- Opción 2: método Quicksort



- Opción 3: Elemento máximo de un arreglo



- Opción 4 : Merge_Sort

```
Ingresa tu opcion: 4
[1, 2, 3, 4, 31, 42, 88]
```