



## Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

# Laboratorios de computación salas A y B

*Profesor:* M.I. MARCO ANTONIO MARTINEZ

*Asignatura:* ESTRUCTURA DE DATOS Y ALGORITMOS

*Grupo:* 17

*No de Práctica(s):* 11

ISLAS ESPINO JESÚS ABRAHAM

*Integrante(s):*

*No. de Equipo de  
cómputo empleado:* 23

*No. de Lista o Brigada:*

*Semestre:* 2020-2

*Fecha de entrega:* 2020

*Observaciones:*

**CALIFICACIÓN:** \_\_\_\_\_

# **Práctica 11**

## **Estrategias para la construcción de algoritmos**

### **Objetivo**

El objetivo de esta guía es implementar, al menos, dos enfoques de diseño (estrategias) de algoritmos y analizar las implicaciones de cada uno de ellos. construir algoritmos (fuerza bruta, algoritmo ávido, bottom-up, top-down, divide y vencerás, etc).

### **INTRODUCCIÓN**

A través de los años, los científicos de la computación han identificado diversas técnicas generales que a menudo producen algoritmos eficientes para la resolución de muchas clases de problemas.

En la actualidad el ser humano se enfrenta a distintas situaciones: de aprendizaje, de retroalimentación y en muchas ocasiones dificultades que con la experiencia o por la elección de la alternativa apropiada, va dando solución.

Esta práctica se presentaran algunas de las técnicas más importantes como son: (fuerza bruta, algoritmo ávido, bottom-up, top-down, divide y vencerás,

Para resolver un problema se pueden desarrollar diversos algoritmos, existen en ocasiones múltiples soluciones, pero dentro de ellas existen las que son más eficientes y es aquí donde la habilidad del desarrollador juega un papel importante. Por lo que la práctica continua contribuye a la mejora del desarrollo.

# DESARROLLO

## FUERZA BRUTA

```
Simbolo del sistema - python
C:\Users\Alex>python
Python 2.7.18 (v2.7.18:8d21aa21f2, Apr 20 2020, 13:25:05) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from string import ascii_letters, digits
>>> from itertools import product
>>> caracteres = ascii_letters+digits
>>> def buscador(con):
...     archivo = open("combinaciones.txt", "w")
...     if 3<= len(con) <= 4:
...         for i in range(3,5):
...             for comb in product(caracteres, repeat = i):
...                 prueba = "".join(comb)
...                 archivo.write( prueba + "\n" )
...                 if prueba == con:
...                     print('Tu contraseña es {}'.format(prueba))
...                     archivo.close()
...                     break
...     else:
...         print('Ingresa una contraseña que contenga de 3 a 4 caracteres')
...
>>> from time import time
>>> t0 = time()
>>> con = 'H014'
>>> buscador(con)
Tu contraseña es H014
>>> print("Tiempos de ejecución {}".format(round(time()-t0, 6)))
File "<stdin>", line 1
    print("Tiempos de ejecución {}".format(round(time()-t0, 6)))
    ^
SyntaxError: invalid syntax
>>> print("Tiempos de ejecución {}".format(round(time()-t0, 6)))
Tiempos de ejecución 140.31
>>> con = 'H014'
>>>
>>>
>>>
>>> buscador(con)
Tu contraseña es H014
>>>
>>> print("Tiempos de ejecución {}".format(round(time()-t0, 6)))
Tiempos de ejecución 192.881
>>> _
```

## AVIDO

```
Simbolo del sistema - python
Microsoft Windows [Versión 10.0.18362.778]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Alex>python
Python 2.7.18 (v2.7.18:8d21aa21f2, Apr 20 2020, 13:25:05) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> def cambio(cantidad, denominaciones):
...     resultado = []
...     while (cantidad > 0):
...         if (cantidad >= denominaciones[0]):
...             num = cantidad // denominaciones[0]
...             cantidad = cantidad - (num * denominaciones[0])
...             resultado.append([denominaciones[0], num])
...             denominaciones = denominaciones[1:]
...     return resultado
...
>>> print(cambio(1000, [500, 200, 100, 50, 20, 5, 1]))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cambio' is not defined
>>> print(cambio(1000, [500, 200, 100, 50, 20, 5, 1]))
[[500, 2]]
>>> print(cambio(500, [500, 200, 100, 50, 20, 5, 1]))
[[500, 1]]
>>> print(cambio(300, [50, 20, 5, 1]))
[[50, 6]]
>>> print(cambio(200, [5]))
[[5, 40]]
>>> print(cambio(300, [50, 20, 5, 1]))
[[50, 6]]
>>> print(cambio(98, [50, 20, 5, 1]))
[[50, 1], [20, 2], [5, 1], [1, 3]]
>>> print(cambio(98, [5, 20, 1, 50]))
[[5, 19], [1, 3]]
>>> _
```

## DIVIDE Y VENCERAS

```
CS: Símbolo del sistema - python
Microsoft Windows [Versión 10.0.18362.778]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Alex>python
Python 2.7.18 (v2.7.18:8d21aa21f2, Apr 20 2020, 13:25:05) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> def quicksort(lista):
...     quicksort_aux(lista,0,len(lista)-1)
...
>>> def quicksort_aux(lista,inicio, fin):
...     if inicio < fin:
...         pivote = particion(lista, inicio, fin)
...         quicksort_aux(lista, inicio, pivote-1)
...         quicksort_aux(lista, pivote+1, fin)
...
>>> def particion(lista, inicio, fin):
...     pivote = lista[inicio]
...     print("Valor del pivote {}".format(pivote))
...     izquierda = inicio+1
...     derecha = fin
...     print("Indice izquierdo {}".format(izquierda))
...     print("Indice derecho {}".format(derecha))
...     bandera = False
...     while not bandera:
...         while izquierda <= derecha and lista[izquierda] <= pivote:
...             izquierda = izquierda + 1
...         while lista[derecha] >= pivote and derecha >= izquierda:
...             derecha = derecha -1
...         if derecha < izquierda:
...             bandera= True
...         else:
...             temp=lista[izquierda]
...             lista[izquierda]=lista[derecha]
...             lista[derecha]=temp
...     print(lista)
...     temp=lista[inicio]
...     lista[inicio]=lista[derecha]
...     lista[derecha]=temp
...     return derecha
...
>>> lista = [21, 10, 0, 11, 9, 24, 20, 14, 1]
>>> print("lista desordenada {}".format(lista))
lista desordenada [21, 10, 0, 11, 9, 24, 20, 14, 1]
>>> quicksort(lista)
Valor del pivote 21
Indice izquierdo 1
Indice derecho 8
[21, 10, 0, 11, 9, 1, 20, 14, 24]
Valor del pivote 14
Indice izquierdo 1
Indice derecho 6
[14, 10, 0, 11, 9, 1, 20, 21, 24]
Valor del pivote 1
Indice izquierdo 1
Indice derecho 4
[1, 0, 10, 11, 9, 14, 20, 21, 24]
Valor del pivote 10
Indice izquierdo 3
Indice derecho 4
[0, 1, 10, 9, 11, 14, 20, 21, 24]
>>> print("lista ordenada {}".format(lista))
lista ordenada [0, 1, 9, 10, 11, 14, 20, 21, 24]
>>>
```

### Explicación de código "Divide y Vencerás":

Comenzando por decir que este está compuesto por 3 funciones donde una de ellas se llama así mismas cuando el programa se está ejecutando, los elementos que usa son el pivote que es el primer elemento del arreglo.

Para lograr hacer el ordenamiento este ejecuta la función `quicksort(lista)` haciendo que de esta manera se le pase a arreglo en el cual va a trabajar, hablando de la la función `quicksort_aux` esta comparar que el inicio se mayor al fin con la finalidad de realizar el arreglo en el intervalo dado, siguiendo por la variable `pivote` que se iguala a la función `partición` siendo esta función la que contiene el algoritmo de `divide y vencerás`, lo último que se realiza es que la función

quicksort\_aux se llama de nuevo así misma pero ahora en intervalos de 2 grupos ya formados.

### **Conclusión**

La práctica me sirvió para familiarizarme con estos algoritmos y revisar el comportamiento que tienen y el uso que se le puede dar a cada uno dependiendo la aplicación que se necesitó.