|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Marco Martínez |
| *Asignatura:* | EDA I |
| *Grupo:* | 7 |
| *No de Práctica(s):* | 11 |
| *Integrante(s):* | Cuéllar Uribe Fernando |
| *No. de Equipo de cómputo empleado:* | 24 |
| *No. de Lista o Brigada:* | 12 |
| *Semestre:* | 2020-2 |
| *Fecha de entrega:* | 21/04/2020 |
| *Observaciones:* |  |
| Calificación: |  |

**Cuéllar Uribe Fernando  
Grupo 17  
Estructura de Datos y Algoritmos I**

**Guía práctica de estudio 0:**

**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.

**Conceptos a revisar en Python:**

♣ Escribir y leer en archivos.

♣ Graficar funciones usando la biblioteca Matplotlib.

♣ Generar listas de números aleatorios.

♣ Medir y graficar tiempos de ejecución.

**Conclusiones:**Las estrategias que nos enseñaron son muy útiles para la solución de problemas.

La estrategia llamada **divide y vencerás,** consiste en dividir tareas en subtareas hasta que queden lo mas sencillas posibles y se obtenga un resultado deseado juntando las subtareas y obteniendo un resultado único. (ej. Quicksort)

La estrategia **incremental** consiste en implementar y probar poco a poco constantemente creando cierto incremento. Ya que, en cada prueba o interacción, se va agregando información hasta completar la tarea. (ej. Instertion sort)

En **top-down** los cálculos, como su nombre lo dice, se hacen de arriba para abajo, en cierto problema. Por ejemplo en una serie de números como en la sucesión de Fibonacci.

**Bottom-up** como su nombre lo dice, hace cálculos de abajo hacia arriba, es decir, resolver un problema a partir de subproblemas que ya han sido resueltos. La solución final se forma a partir de la combinación de una o más soluciones que se guardan en una tabla, ésta previene que se vuelvan a calcular las soluciones.

La estrategia de **algoritmos ávidos (greedy)** se basa en decisiones en orden especifico, pero una vez que se ejecuta una decisión, esta ya no se vuelve a considerar.

Por ultimo, a la **fuerza bruta**, es decir, se hace una amplia búsqueda de todas las opciones que hay para resolver un problema, por ejemplo, para descifrar una contraseña de 3 o 4 caracteres, concede todas las combinaciones posibles y las aplica hasta obtener el resultado correcto, la desventaja de este método es el tiempo que toma.

**Bibliografía:**El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.  
  
Laboratorio de Computación Salas A y B Manual de prácticas de Estructura de Datos y Algoritmos I MADO-19\_EDAI  
Guía práctica de estudio 11:

Aplicaciones de <http://lcp02.fi-b.unam.mx/>

[1] Design and analysis of algorithms; Prabhakar Gupta y Manish Varshney; PHI Learning, 2012, segunda edición.

[2] Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein; The MIT Press; 2009, tercera edicion.

[3] Problem Solving with Algorithms and Data Structures using Python; Bradley N. Miller y David L. Ranum, Franklin, Beedle & Associates; 2011, segunda edition.

[4] https://docs.python.org/3/library/itertools.html#

[5] https://docs.python.org/3/library/itertools.html#itertools.product

[6] https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

[7] https://docs.python.org/3.5/library/pickle.html