**Resumen**

Una de las aplicaciones más frecuentes, en programación es la ordenación.

• Existen dos técnicas de ordenación fundamentales en gestión de datos: ordenación de listas y ordenación de archivos.

• Los datos se pueden ordenar en orden ascendente o en orden descendente.

• Cada recorrido de los datos durante el proceso de ordenación se conoce como pasada o iteración.

• Los algoritmos de ordenación básicos son:

• Selección.

• Inserción.

• Burbuja.

• Los algoritmos de ordenación más avanzados son:

• Shell.

• Mergesort.

• Quicksort.

• La eficiencia de los algoritmos de burbuja, inserción y selección es 0(n2).

• La eficiencia de los algoritmos heapsort, radixsort, mergesort y quicksort es 0(n Log n).

• La búsqueda es el proceso de encontrar la posición de un elemento destino dentro de una lista. • Existen dos métodos básicos de búsqueda en arrays: búsqueda secuencial y binaria.

• La búsqueda secuencial se utiliza normalmente cuando el array no está ordenado. Comienza en el principio del array y busca hasta que se encuentra el dato buscado y se llega al final de la lista.

• Si un array está ordenado, se puede utilizar un algoritmo más eficiente denominado búsqueda binaria.

• La eficiencia de una búsqueda secuencial es 0(n).

• La eficiencia de una búsqueda binaria es 0(log n).

**Taller**

1. **(10) Se desea eliminar todos los números duplicados de una lista**

**Por ejemplo**

**Si se le ingresan los valores [4,7,11,4,9,5,11,7,3,5]**

**Se debe cambia a [4,7,11,9,5,3]**

Implementación en Python (Punto 1/ elementos\_repetidos\_1.py)

1. **(10) Elimine los elementos duplicados de un vector ordenado.**

**¿Cuál es la eficiencia del método?**

**Compárela con la eficiencia del punto 1**

Implementación en Python (Punto2/elementos\_repetidos\_vector\_ordenado\_2.py)

Comparación de Eficiencia:

Complejidad: O(N) (Primero) --- O(N)(Segundo)

Tiempo:

0.001161216898803711 (Primero)

0.002035736246948242 (Segundo)

1. **(5) Dada la siguiente lista**

**[47,3,21,32.56,92]**

**Después de 2 “pasadas” de un algoritmo de ordenación, la lista ha quedado dispuesto así**

**[3,21,47,32,56,92]**

**¿Qué algoritmo de ordenación se esta utilizando (selección, burbuja o inserción)?**

**Justifique su respuesta**

Burbuja, pues a la hora de comparar mira si el primer elemento es mayor que el segundo y lo cambia, después este, ahora el segundo elemento, se compara con el tercero y se cambia.

1. **(10) Utilizar el algoritmo de ordenación SHELL, encuentre las pasadas e intercambios que se realizan para la ordenación de la siguiente lista**

****

Implementación en Python (Punto4/ shell\_sort\_4.py)

1. **(10) Dada una lista voto [0.......n-1], donde cada elemento de lista representa un voto en las elecciones. Suponga que cada voto se da como un número entero que representa el ID del candidato elegido. Desarrolle un algoritmo para determinar quién gana la elección. Determine la complejidad del algoritmo**

Implementación en Python (Punto5/votos\_candidatos\_5.py)

1. **(10) Se cuenta con una lista de tuplas**

**futbolistasTup = [(1, "Casillas"), (15, "Ramos"), (3, "Pique"), (5, "Puyol"), (11, "Capdevila"), (14, "Xabi Alonso"), (16, "Busquets"), (8, "Xavi Hernandez"), (18, "Pedrito"), (6, "Iniesta"), (7, "Villa")]**

**Si se aplica futbolistasTup.sort(key=lambda futbolista: futbolista[0])**

1. **Que resultado se obtiene al aplicar el método .sort**
2. **Que se esta especificando en los parámetro (key=lambda futbolista: futbolista[0])**
3. **Aplique este método a las listas de los puntos 1,3, 4. Que conclusión puede obtener**
4. **Por favor según opinión realice una tupla con los mejores inventos del 2019.**

**Donde usted califica el que mas le gusta o le parece importante. **

**Anotación la escala con la que usted cuenta es de 1 a 100 (no tiene que asignar ninguno de los extremos si no lo desea)**

Implementación en Python (Punto6/ futbolistas\_6.py)

**c)** Básicamente este método sirve como ordenador de listas, ya esta implementado para los arreglos en python y tiene su funcionamiento tanto en tuplas, listas, etc.

1. **(10) Diseñe e implemente una función para encontrar todos los valores negativos dentro de una lista dada. Tu función debería devolver una nueva lista que contiene los valores negativos.**

**¿Cuándo ocurre el peor de los casos y cuál es el tiempo de ejecución para ese caso?**

Implementación en Python (Punto7/ ValoresNegativos\_7.py)

Peor de los casos: O(n)

Tiempo de ejecución: 0.0013133279522705079

1. **(5) Dada la siguiente lista de números:**

**[21, 1, 26, 45, 29, 28, 2, 9, 16, 49, 39, 27, 43, 34, 46, 40]**

**¿Cuál sería el resultado a la lista que después de 3 llamadas recursivas de ordenamiento por mezcla?**

**¿Por qué?**

[1,2,9,21,26,28,29,45,16,27,39,49, 43, 34, 46, 40]

Pues el algoritmo alcanza a ordenar el primer subarreglo y el subsubarreglo del segundo subarreglo

1. **(5) Dado el siguiente código evalué su complejidad.**

**Determine que hace este esté algoritmo**

**Explique cada uno de sus funciones y sus clases**

****

Básicamente este algoritmo implementa un tipo de arreglo ordena el tiene funciones para agregar, eliminar, buscar

Def \_\_init\_\_ (self): Crea el set

Def \_\_len\_\_ (self): Determina el largo del set

Def \_\_contains\_\_ (self,element): determina si un elemento esta en el set

Def add(self, element): Añade un elemento al ser

Def remove(self, element): Elimina un elemento del set

Def isSubSetOf(self, setB): Determina si un set es un sublista de SetB

Def \_\_iter\_\_ (self):

Def \_findPosition(self, element): Busca el elemento (element) en la set ordenada y si esta retorna su posición

COMPLEJIDAD: O(N)

1. **(10) Construya un algoritmo para encontrar un valor específico en una matriz de valores ordenada por filas y columna.**

**El algoritmo toma como entrada una matriz de valores donde cada fila y cada columna están en orden, junto con un valor para ubicar en esa matriz. Devuelve si ese elemento existe en la matriz.**

**Por ejemplo, dado la siguiente matriz y buscar el 7, el algoritmo daría como resultado sí**

**Pero si se pide encontrar el número 0, el algoritmo daría como resultado no**

****

Implementación en Python (Punto10/ matriz\_busqueda\_10.py)

1. **(10) se tiene una lista A con 100 elementos A[ a1……a100 ]**

**B de 60 elementos B[ b1……b60 ]**

**Se desean resolver las siguientes tareas**

1. **Ordenar cada lista aplicando el método Quicksort**
2. **Crear una lista C que sea la unión de la lista A y B**
3. **Ordenar la lista C y visualizarla**

Implementación en Python

(Punto11 /union\_y\_ordenamiento\_de\_listas\_11.py)

1. **(5) Investigue en que consisten las técnicas de ordenación Radixsort y Binsort, en lo posible haga un ejemplo**

Radixsort: El radixsort se basa en la comparación por guarismos primero se comparar los guarismos que tienen menos cifras significativas y así sucesivamente hasta llegar a la última cifra significativa.

Binsort: En el binsort también conocido como ordenamiento de cubetas, primero se inicializa un arreglo vacío, después se pone los elementos correspondientes en cada cubeta, se organiza cada cubeta que no este vacía, se visita cada cubeta en orden y se ponen de nuevo en el arreglo original

Julian David Valencia Restrepo

202110151010