Diego Velásquez Varela – 202010004101

Taller Ordenación – ST0245-008

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]**

R= Solución en VsCode

**2. (10) Elimine los elementos duplicados de un vector ordenado. ¿Cuál es la eficiencia del método? Compárela con la del punto 1**

R= La eficiencia de ambos códigos es la misma, puesto que en mi caso, la implementación es la misma para ambos, además la complejidad es de O(n) por lo que su eficiencia depende de la cantidad de datos que contenga la lista.

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 está utilizando (selección, burbuja o inserción)?**

*Justifique su respuesta*

R = El algoritmo que se está realizando es el de selección, vemos que primero parte de buscar el elemento menor entre la primera y la última posición, en este caso el 3 lo cambia con el 47 (En la primera pasada queda [3,47,21,32,56,92]) posteriormente, repite el procedimiento intercambiando las posiciones del numero 21 (que es el segundo número menor) con el 47. Resultando [3,21,47,32,56,92]

***Esta metodología es la correspondiente al selección sort***

El algoritmo también podría estar utilizando el método BubbleSort, en este caso primero partiríamos de tomar el elemento [0] de la lista y compararlo con el elemento [1], como es mayor los intercambia, y posteriormente compara el elemento [0] con el elemento [2] y como de nuevo, es mayor, los intercambia y obtenemos [3,21,47,32,56,92]

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



R= Solución en VSCode. Podemos ver que se dan 3 pasadas y 12 Intercambios.

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

R= La complejidad del algoritmo es de O(n). Por la complejidad del for y la serie de instrucciones con O(1) (El código está en el repositorio)

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])

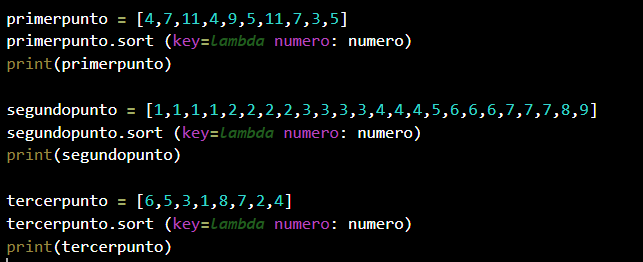
1. ¿Qué resultado se obtiene al aplicar el método.sort?

La lista de tuplas se organiza de mayor a menor según el numero que le corresponde a cada uno de los jugadores



1. Que se está especificando en los parámetro (key=lambda futbolista: futbolista[0])

Aquí estamos especificando que ordenaremos la tupla. Pero con respecto al elemento 0 de la lista, que en este caso es (1, ‘Casillas’). En el caso de poner [1] se organizan según el nombre



El método.sort nos funciona para ordenar tanto listas comunes como tuplas, sin embargo la parte la vital del algoritmo es especificar la expresión que nos indica la forma de ordenarlas.

1. VSCode Inventos tomados de ***#***[***https://time.com/collection/best-inventions-2019/***](https://time.com/collection/best-inventions-2019/)

MejoresInventos **=** [(50,'LightPhone2'),(21,'BoseFrames'), (90,'Wave'), (100,'ECOncrete'),(85,'RoliLumi'),(47,'RoybiRobot'), (100,'LightSail2'),(94,'AeroFarms')]

1. MejoresInventos.sort(key**=*lambda*** Inventos: Inventos)
2. print(MejoresInventos)

**7) (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?**

El algoritmo tiene una complejidad de O(n) (lineal) el peor de los casos es en el que todos los elementos de la lista tienen un orden negativo, puesto que se debe ejecutar un paso extra (Haciendo uso de la función Append). Al importar la librería Time para encontrar los tiempos de ejecución todos resultan en 0.0seg (esto puede ser debido al rendimiento del computador, u otros factores que trabajamos en el reto 1)

8) (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?



Después de 3 llamadas recursivas de ordenamiento obtenemos [21,1] Sabemos que el Ordenamiento por mezclas está basado en la técnica divide y vencerás (Con complejidad O (n log n)

9) (5) Dado el siguiente código evalué su complejidad. Determine que hace este esté algoritmo explique cada uno de sus fucines y sus clases.



* Init crea una lista vacía
* Len devuelve cantidad de elementos de la lista
* Contains verifica si un elemento pertenece a una lista utilizando la función findposition.
* Add Agrega un nuevo elemento a la lista, asegurándose de que no se encuentra previamente y luego llamando a la función findposition para conocer su ubicación.
* Remove elimina elementos de la lista. Se asegura primero de que el elemento está en la lista. Y de ser el caso, con FindPosition encuentra la posición y luego usa pop para eliminar el elemento.
* IsSubSetOf compara si dos listas son iguales
* Iter Recorre la lista
* FindPosition Encuentra la posición de un elemento de la lista

La complejidad la podemos obtener notando que tiene funciones de orden (O(1)) otras de orden (O(n)) como el for y una más de orden (O(Log n)).

A esos 3 ordenes les hallamos el producto y encontramos que la complejidad algorítmica es de O(nLogn)

10) Solucionado en VsCode

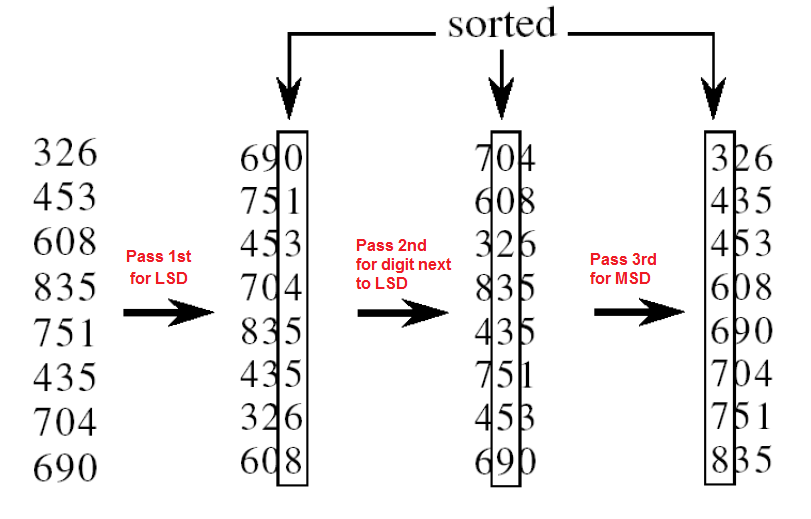
11) Solucionado en VSCode

12) RadixSort Y BinSort (Logré realizar un ejemplo de RadixSort, pero no de BinSort ☹)

RadixSort

El ordenamiento de raíz es un algoritmo de clasificación, que es eficiente si ya conocemos el rango de valores objetivo. La complejidad del algoritmo es de O(nk) dónde n es el tamaño del input y k la longitud del digito del número. Por ejemplo, la longitud del 517 es 3.

El comportamiento es similar al siguiente



Sea el siguiente ejemplo

Lista = [10,21,17,34,44,11,654,123]

Según el algoritmo, ordenaremos la lista según el dígito de uno ( dígito menos significativo).

0: 10

1: 21 11

2:

3: 123

4: 34 44 654

5:

6:

7: 17

8:

9:

Entonces, la matriz se convierte en 10,21,11,123,24,44,654,17

Ahora, ordenaremos según el dígito de la decena :

0:

1: 10 11 17

2: 21 123

3: 34

4: 44

5: 654

6:

7:

8:

9:

Ahora, la matriz se convierte en: 10,11,17,21,123,34,44,654

Finalmente, ordenamos según el dígito de las centenas ( dígito más significativo):

0: 010 011 017 021 034 044

1: 123

2:

3:

4:

5:

6: 654

7:

8:

9:

La matriz se convierte en: 10,11,17,21,34, 44,123,654 que está ordenado. Así es como funciona nuestro algoritmo.

Apoyado en: <https://www.hackerearth.com/practice/algorithms/sorting/radix-sort/tutorial/>

<https://www.lewuathe.com/radix-sort-in-python.html>

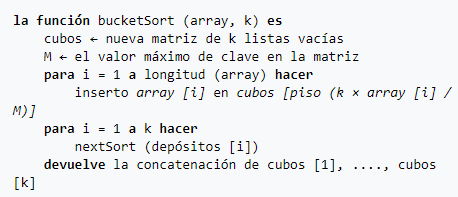
<https://es.qwe.wiki/wiki/Radix_sort>

<https://www.youtube.com/watch?v=Uey0-GOMtT8>

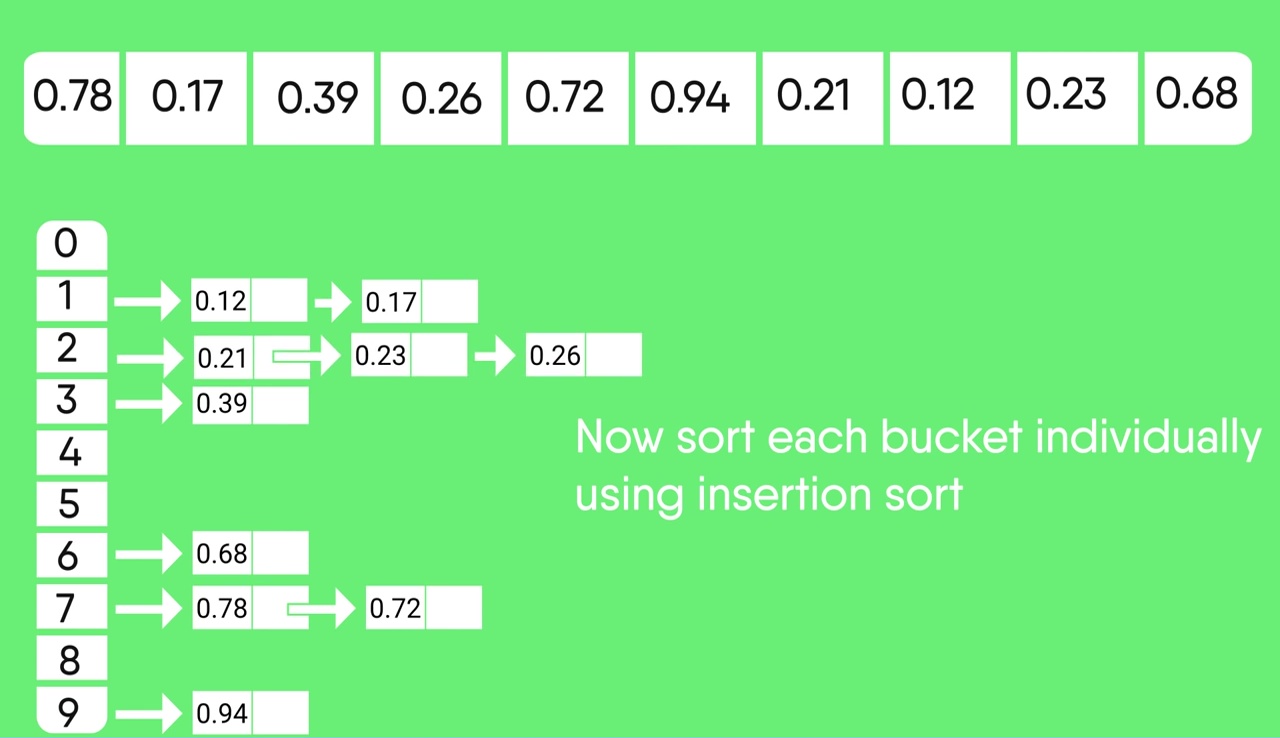
Ejemplo en Visual Studio 😊

BinSort

El ordenamiento por cubos, o por contenedores, funciona distribuyendo los elementos de la lista en varios depósitos, y posteriormente cada deposito se clasifica individualmente utilizando algún algoritmo de ordenamiento.



1. Configuramos la lista de depósitos que inicialmente están vacíos
2. Realizamos pasadas en la lista original y ubicamos cada objeto en su cubo.
3. Clasificamos cada valde que no esté vacío.
4. Recopilamos visitando los depósitos en orden, y volvemos a colocar todos los elementos en la matriz original.



Apoyado en: <https://www.cdn.geeksforgeeks.org/wp-content/uploads/scene01801.jpg>

<https://www.bing.com/search?q=BucketSort&cvid=311d16c8f07443279dbecef2b12a763d&pglt=547&FORM=ANNTA1&PC=ACTS#>

<https://www.programiz.com/dsa/bucket-sort>