Algoritmo MSD radix sort

Ibeth Dariana Ortega Rojas – 191988

Gerardo Alejandro Duarte Rodríguez – 192055

Facultad de Ingeniería, Universidad Francisco de Paula Santander Ocaña

193502: Análisis de Algoritmos

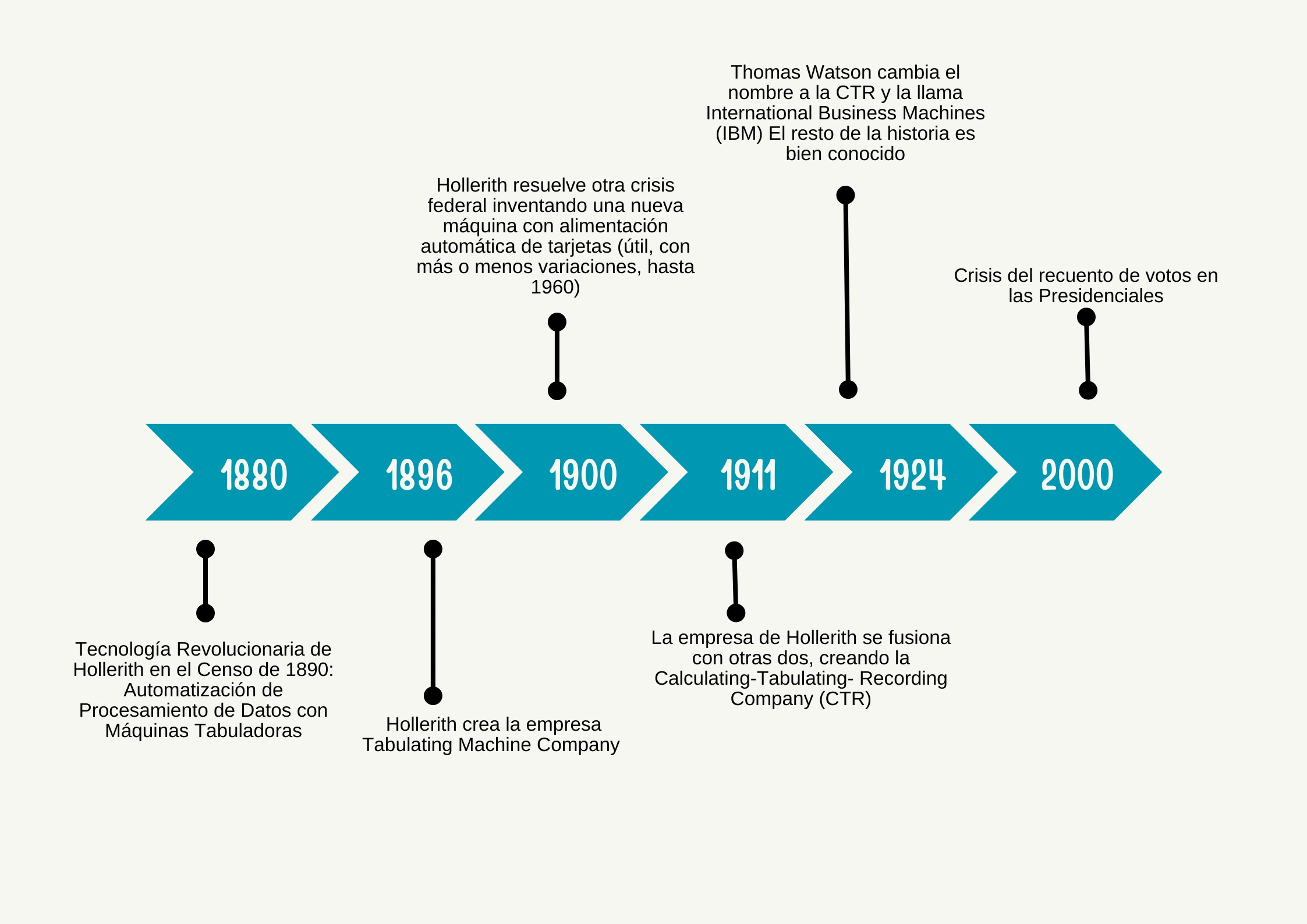
Mgrt. Alexander Guerrero Avendaño

1. de septiembre de 2023

Algoritmo MSD radix sort

1. Redactar una investigación y explicación del algoritmo asignado.
   1. Origen:

El algoritmo Radix Sort de Herman Hollerith surgió en la década de 1880, el cual desarrollo una máquina de tarjetas perforadas para procesar datos censales; el problema surgió cuando en Estados Unidos ( 1880 ) no se podía terminar el censo de la década anterior (en concreto, no se llegaba a contar el número de habitantes solteros) . Herman Hollerith (empleado de la oficina del censo, de 20 años de edad) inventa una máquina tabuladora eléctrica para resolver el problema, en esencia es una implementación física del radix Sort 1890, se usan unas 100 máquinas de Hollerith para tabular las listas del censo de la década (un operador experto procesaba tarjetas en una jornada laboral de 6,5 horas, unas 49 tarjetas por minuto). Sin embargo, el concepto de ordenación por radix (base) se ha ido perfeccionando cada vez más en la historia de la informática. LSD Radix Sort y MSD Radix Sort son las secciones más populares de este algoritmo.



Fuente: Elaboración Propia

***1.2 Definición:***

El algoritmo MSD Radix Sort (Dígito más significativo) es un algoritmo que se utiliza principalmente para ordenar datos alfabéticos, como cadenas de texto o datos numéricos ​​en notación de base fija (por ejemplo, números en base 10). Su método se basa en clasificar elementos dividiéndolos en grupos en función de los dígitos más significativos y luego cambiando el orden de cada uno de los grupos hasta tener la lista de números ordenada.

***1.3 Ventajas y desventajas:***

**Ventajas**:

* Tercer algoritmo más rápido.
* El ordenamiento es razonablemente eficiente si el número de dígitos en las llaves no es demasiado grande.
* Si las máquinas tienen la ventaja de ordenar los dígitos (sobre todo si  están en binario) lo ejecutaron con mucho mayor rapidez de lo que ejecutan una comparación de dos llaves completas.

**Desventajas**:

* Si los números son de diferente longitud son más difíciles de ordenar.
* Se requiere conocer la cantidad de dígitos del valor máximo (para saber cuándo el método ya ordeno todos los elementos).
* Se requiere de espacio para almacenar los punteros del frente y de la parte posterior de la cola, además de un campo adicional en cada registro que se utiliza como puntero a la lista encadenada.
* Requiere mas memoria que el LSD debido a la implementación recursiva.

***1.4 Clasificación por base:***

La clasificación de base MSD inicia desde el dígito más significativo. Asigna dígitos del mismo valor posicional a diferentes cubetas y luego ordena recursivamente cada depósito hasta que a cada cubeta le queda uno o ningún elemento.

“Es posible que no sea necesario examinar cada dígito de cada número para obtener un resultado ordenado” — Claire Lee, por lo que es considerablemente más rápido que el tipo de base LSD . Sin embargo, requiere más espacio que el tipo de base LSD debido a la implementación recursiva .

***1.5 Mejor, promedio y Peor Caso:***

|  |  |  |
| --- | --- | --- |
| Mejor Caso | Caso Promedio | Peor Caso |
| El mejor caso ocurre cuando todas las cadenas que se van a ordenar tienen la misma longitud y están distribuidas uniformemente entre todos los dígitos significativos. En este caso ideal, el algoritmo puede realizar una clasificación eficiente en un tiempo lineal O(n), donde "n" es el número de elementos que se van a clasificar. | El rendimiento promedio del algoritmo de clasificación de base MSD suele estar en el rango de O(n \* k) a O(n \* k^2), donde "n" es el número de elementos a ordenar y "k" es la longitud máxima. En la mayoría de los casos prácticos, "k" suele ser constante o tener un valor finito, lo que significa que el rendimiento en términos de tiempo medio de ejecución es cercano a O(n). | El peor caso ocurre cuando las cadenas tienen diferentes longitudes y/o están distribuidas uniformemente entre los dígitos más significativos. En este caso, el algoritmo puede requerir mucho tiempo y espacio, ya que genera múltiples capas de iteraciones. En el peor de los casos, la complejidad del algoritmo puede ser O(n\*k), donde "n" es el número de elementos a ordenar y "k" es la longitud máxima de las claves. |

***1.6 Complejidad de Tiempo:***

|  |  |
| --- | --- |
| Mejor Caso | Peor Caso |
| La complejidad del tiempo en el mejor de los casos es O(N). | La complejidad del tiempo en el peor de los casos es O(N\*M), donde M = la longitud promedio de las cadenas. |

***1.7 Limitaciones:***

* Requiere memoria adicional:

El algoritmo de clasificación MSD Radix a menudo requiere una memoria adicional significativa para almacenar grupos temporales de elementos durante el proceso de clasificación. Esto puede ser un problema si está ordenando grandes conjuntos de datos o si la memoria es limitada.

* No es eficiente para representaciones binarias:

El algoritmo de clasificación MSD Radix está diseñado principalmente para ordenar datos alfabéticos o numéricos en caracteres decimales o de base fija.

* Dependiente del sistema de numeración:

El algoritmo está especialmente diseñado para 10 números base y caracteres alfabéticos. Para otros sistemas de numeración, como el binario, el octal o el hexadecimal, se requiere personalización y puede que no sea tan eficiente.

En el caso de números del 1 al 100, cada número tiene hasta tres dígitos (por ejemplo, 1 tiene un dígito, 45 tiene dos dígitos, y 100 tiene tres dígitos). El algoritmo de ordenación por MSD pasa por cada dígito de los números en orden, desde el dígito más significativo hasta el menos significativo, y realiza un conteo de apariciones de los dígitos en ese lugar (0-9 en base decimal). Luego, reorganiza los números en función de estos conteos. Debido a que los números en este rango tienen un máximo de tres dígitos, el algoritmo realizará como máximo tres pasadas (una para cada dígito). Sin embargo, cuando se trata de números más grandes, como números de 1 al 1000 o números aún más grandes, el algoritmo tendrá que realizar más pasadas debido a la cantidad de dígitos en esos números. Por ejemplo, para números de 1 al 1000, el algoritmo podría requerir hasta cuatro pasadas (una para cada dígito). En general, cuanto más grande sea el número y cuantos más dígitos tenga, más pasadas requerirá el algoritmo, lo que aumenta la complejidad y el número de pasos necesarios para ordenar la lista de números.

***1.6 Aplicaciones más comunes donde se puede usara este algoritmo de ordenamiento:***

Ordenación decimal de números enteros o números de coma flotante. Si necesita ordenar números con decimales, el algoritmo de clasificación de base MSD es una opción adecuada. Esto es útil en aplicaciones financieras, contabilidad, análisis de datos y cualquier situación en la que los números deban organizarse por su valor numérico.

***1.7 Procedimiento:***

1. Define la longitud máxima de una secuencia numérica o de texto, esto es necesario para determinar cuántos pasos se necesitan en el proceso de clasificación.

2. Inicialice una matriz de contadores o segmentos, donde cada elemento representa un posible dígito o carácter en la secuencia, inicializado a cero.

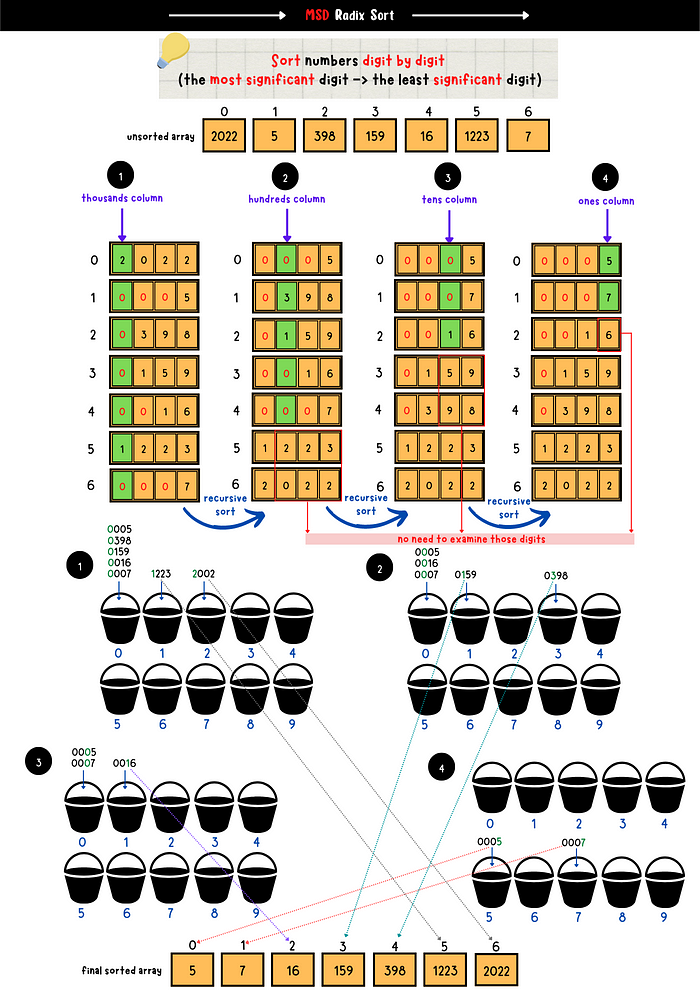
3. Divida los elementos en cubetas , cada elemento en la cubeta correspondiente según su dígito más significativo.

4. Ordene cada segmento aplicando el algoritmo MSD Radix Sort a cada cubeta. Esto implica repetir los pasos 1 a 3 para cada cubeta por separado.

5. Repita el procedimiento para el siguiente dígito más significativo. Si quedan dígitos por procesar (por ejemplo, si está ordenando números en notación decimal y aún quedan dígitos menos significativos por considerar), repita los pasos del 1 al 5 para el siguiente dígito más significativo.

6. Al final del proceso, recibirá una lista de elementos, ordenados numéricamente en orden de menor a mayor.

***Imagen de referencia:***



Nota: adapto de Lee, C. (2002). MSD radix sortf. Medium. https://yuminlee2.medium.com/radix-sort-algorithm-764a260f2a04

***1.8 Análisis de Complejidad :***

La complejidad temporal del algoritmo MSD Radix es O(d \*( n+b)), donde:

* d: el número de dígitos de los elementos
* n el número d elementos

La complejidad espacial del algoritmo MSD Radix es O(n + b), donde:

* n es el número de elementos
* b es el número de buckets

***Imagen de referencia:***

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 1 | 6 | 7 |

|  |  |  |
| --- | --- | --- |
| 1 | 6 | 7 |
| 1 | 2 | 3 |

|  |  |  |
| --- | --- | --- |
| 4 | 5 | 6 |
| 1 | 6 | 7 |
| 2 | 7 | 6 |
| 4 | 6 | 3 |
| 2 | 7 | 6 |
| 1 | 2 | 3 |

|  |  |  |
| --- | --- | --- |
| 4 | 5 | 6 |
| 4 | 6 | 7 |

|  |  |  |
| --- | --- | --- |
| 2 | 7 | 6 |

|  |  |  |
| --- | --- | --- |
| 2 | 7 | 3 |

|  |  |  |
| --- | --- | --- |
| 4 | 5 | 6 |
| 4 | 6 | 7 |

|  |  |  |
| --- | --- | --- |
| 2 | 7 | 6 |
| 2 | 7 | 3 |

Fuente: Elaboración Propia

1. **Elabore un pseudocódigo del funcionamiento del algoritmo asignado**

***2.1 Pseudocodigo del algoritmo MSD radix sort***

array 🡨 array original que es dado

max\_length 🡨 cantidad de dígitos del mayor numero del array

auxiliar 🡨 array auxiliar para trabajar los números como sublistas

buckets 🡨 array temporal de números ordenados con counting sort

auxiliar\_buckets 🡨 array que copia a buckets

otro\_mas 🡨 array acumulador de buckets y auxiliar\_buckets

lista\_resultante 🡨 array resultante

max\_num = EncontrarMaximo(array)

max\_length = LongitudDeCadena(max\_num)

auxiliar = InicializarArreglo(max\_length)

Para cada elemento en array:

elemento = ConvertirACadenaYAgregarCerosALaIzquierda(elemento, max\_length)

auxiliar[i] = elemento

auxiliar = ConvertirAArregloDeArreglos(auxiliar)

otro\_mas = [ ]

Para cada posición de dígito desde la Izquierda hasta la derecha:

buckets = InicializarArregloDeArreglos(10)

buckets\_buckets= InicializarArregloDeArreglos(10)

Para cada subarreglo en auxiliar:

Si la posición del digito es diferente de cero:

buckets = InicializarArregloDeArreglos(10)

Para cada elemento en subarray:

Si subarreglo no está vacío:

dígito = ObtenerDígitoEnPosición(subarreglo, posición)

Agregar subarreglo a buckets[dígito]

Si es el último elemento en el subarreglo:

Si posición no es cero:

otro\_mas = CombinarArreglos(auxiliar\_buckets, buckets)

auxiliar\_buckets = otro\_mas

Sino:

CopiarValoresDeArreglo( de: buckets, a: auxiliar\_buckets)

Reiniciar auxiliar

CopiarValoresDeArreglo(de: auxiliar\_buckets, a: auxiliar)

lista\_resultante = [ ]

Para cada subarreglo en auxiliar:

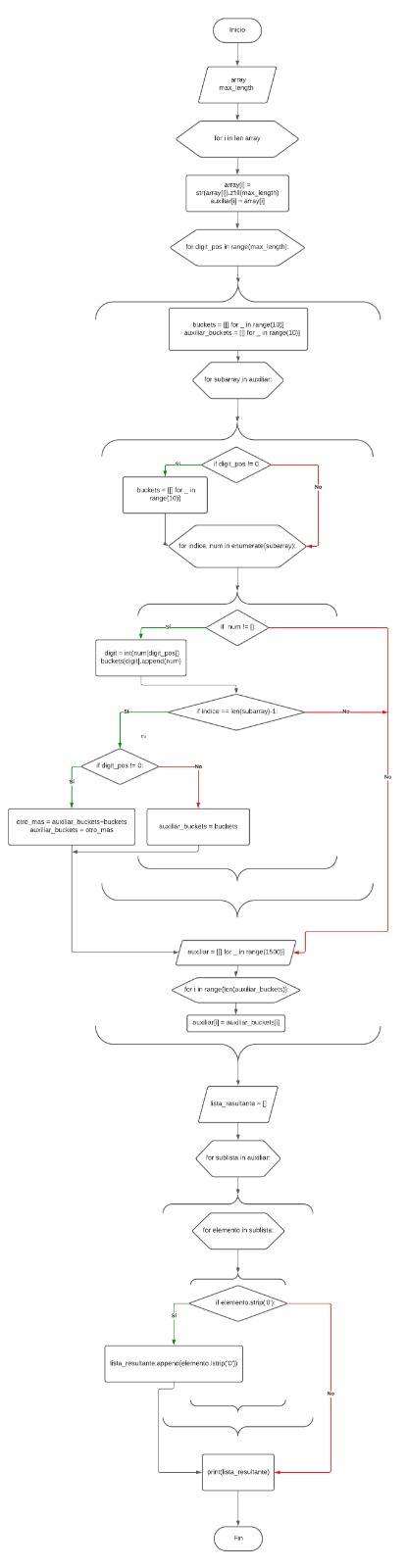
Para cada elemento en subarreglo:

Si elemento no contiene solo ceros a la izquierda:

Agregar elemento a lista\_resultante después de eliminar ceros a la izquierda

Imprimir(lista\_resultante)

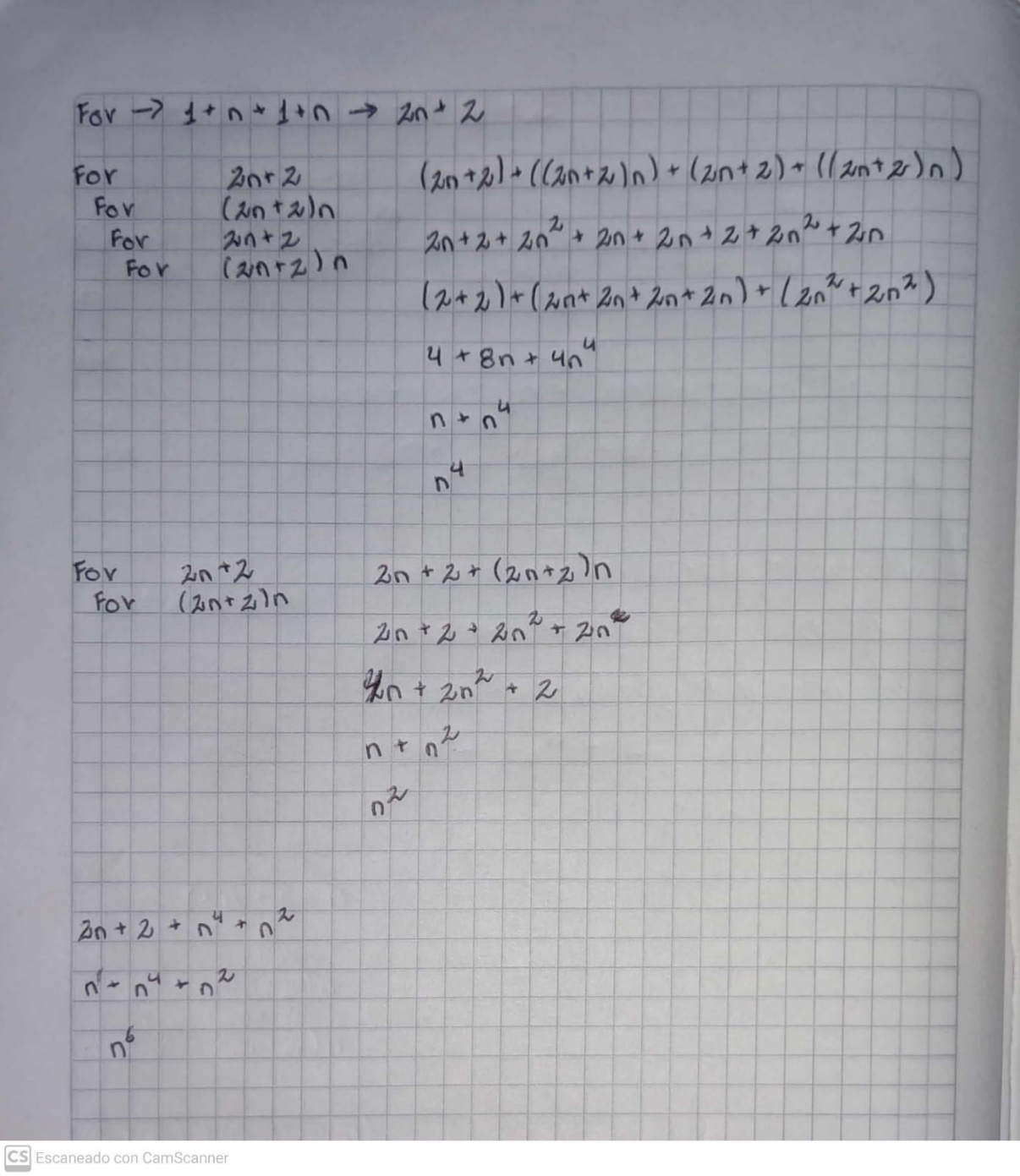
***2.2 Pseudocódigo el diagrama de flujo***



***3.* Codifique en lenguaje de programación PHP y Python el algoritmo asignado sin usar funciones ni objetos.**

La solución se adjunta en un archivo python.

***4.* Calcule el orden de complejidad del funcionamiento del algoritmo asignado y analice la eficiencia para ordenar los ocho (8) vectores de prueba y al realizar la búsqueda del primer elemento en los vectores de prueba.**



Referencias

Follow, S. (2021, mayo 18). MSD( most significant digit ) Radix sort. GeeksforGeeks. https://www.geeksforgeeks.org/msd-most-significant-digit-radix-sort/

Método Radix. (s/f). Prezi.com. Recuperado el 26 de septiembre de 2022, de <https://prezi.com/-ukvwkc3of9n/metodo-radix/>

Lee, C. (2022, septiembre 25). Radix sort algorithm - Claire lee. Medium. https://yuminlee2.medium.com/radix-sort-algorithm-764a260f2a04

Jindal, H. (2021, febrero 25). Ordenamiento Radix. Delft Stack. <https://www.delftstack.com/es/tutorial/algorithm/radix-sort/>

Wikipedia contributors. (2023, agosto 24). Radix sort. Wikipedia, The Free Encyclopedia. <https://en.wikipedia.org/w/index.php>

CS102: Radix sort. (s/f). Saylor Academy. Recuperado el 26 de septiembre de 2023, de https://learn.saylor.org/mod/page/view.php?id=32985

Function template integer\_sort - 1.62.0. (s/f). Boost.org. Recuperado el 26 de septiembre de 2023, de <https://www.boost.org/doc/libs/1_62_0/libs/sort/doc/html/boost/sort/spreadsort/integer__idm45516074556032.html>

Werrebrouck, G. (s/f). msdradixsort. <https://github.com/GillisWerrebrouck/Algorithms/blob/master/sort/msdradixsort.h>

15 algoritmos de ordenación de enteros, cada uno a su ritmo. (2015, diciembre 30). baobab soluciones. https://baobabsoluciones.es/blog/2015/12/30/15-algoritmos-de-ordenacion-de-enteros-cada-uno-a-su-ritmo/