Díaz Hernández Marcos Bryan
Tarea N°1
Grupo: 09

Tarea 1

- Que significa que un algoritmo sea estable.

Basandome en la definicion de inestabilidad menciona los errores de redondeo los cuales se basan en entre la diferencia de un resultado bueno, es la diferencia que existe entre la representacion real de un numero y la aproximacion que hace la computadora. esto es mas notorio en los divisiones. por la limitante de los numeros representados. Ahora la estabilidad es lo contrario.

La estabilidad no tendrá errores de redondeo por la representación y algunos ejemplos son:

$\alpha = 3.1415169265$ $\alpha = 0.3141593 \times 10^{1}$ y $\beta = 0.3141592 \times 10^{1}$
en este caso el error es por un decimal igual a: tal que $\alpha^* = \tilde{\pi}$

$$\frac{|\alpha - \alpha^*|}{\alpha^*} = \frac{0.0000003\ldots}{\tilde{\pi}} < \frac{4}{3}10^{-7} : \text{Poco error}$$

Tal que el error es poco pero si se tiene $y^* = x^* - p^*$
y $p^* = \tilde{\pi} - 6 \cdot 10^{-7}$

$$\frac{|y - y^*|}{y^*} = \frac{10^{-6} - 6 \cdot 10^{-7}}{6 \cdot 10^{-7}} = 2/3 : \text{Mucho error}$$

Significa que el error incrementa en base a como se use el primero y este incrementa el error en los siguientes problemas y operaciones.

- Algoritmos estos estables:

Los estables son los que no realizan operaciones por lo mismo, operaciones como divisiones ya que generan errores.

Insertion: es estable	Selection: No estable
He Merge: es estable	Heapsort: No estable
Bobble: es estable	Quicksort: No estable
Canting: es estable	Radix: no estable

Los estables solo realizan comparaciones, intercambios, intercalaciones pero no modifican los valores y los no estables si hacen operaciones con los valores. Ya que determina la variacion de los valores en los algoritmos.

Díaz Hernández Marcos Bryan
Tarea N°1
Grupo: 09

- Investigar 4 algoritmos de ordenamientos distintos.

1) Concom Sorting

El algoritmo en cada iteracion coloca el valor en su respectiva posicion por medio de la comparacion del primer elemento con los demás y si en la comparacion el valor el es menor que el primero se tiene que incrementar la variable counto que se inicializo en 0, tal que antes del intercombio el valor encuentre su posicion final.

- Pseudocodigo

```
Concom_sort(A, N)
 initialize k, count, i, m to 0
  while k<N
  For j=k+1 to N
  IF( A[k] > A[j])
     increment count
  end for
  if  count =0
      x=0
  IF (A[n] ==A[k+ count])
  swap(A[k+count+i], A[k+count])
    i++
  else
     i to 0  i=0
  if( count != 0)
     swap(A[n], A[k+count])
  if ( count == 0)
    k++
  count ==0
   end while
```

- Complejidad: $O(n^2)$ Ya que sus operaciones dependen de la entrada
- Estabilidad: si ya que no realiza operaciones sobre los valores

- Prueba de escritorio: I: {67, 34, 23} => {23, 34, 67}

1) k=0, count=0, i=0, m=0

2) while (0<2)
   For (j=1 to N)
   if(67 > 34)
      count=1
   if(j=2 to 2)
   if (67 > 23)
      count=2
      i=0

3) if(count != 0)
   swap(67, 23)
   count=0

El arreglo queda listo.

Díaz Hernández Marcos Bryan
Tarea N°1
Grupo: 09

2) Couple Sort.

Crea parejas de números, desde el primer valor hasta el segundo, el tercero y así hasta terminar el tamaño de la lista, Las parejas son comparadas y se intercambian si son sece necesarios.

Y la segunda operación comienza con el segundo elemento y se repite el proceso de comparación e intercambiar si es necesario.

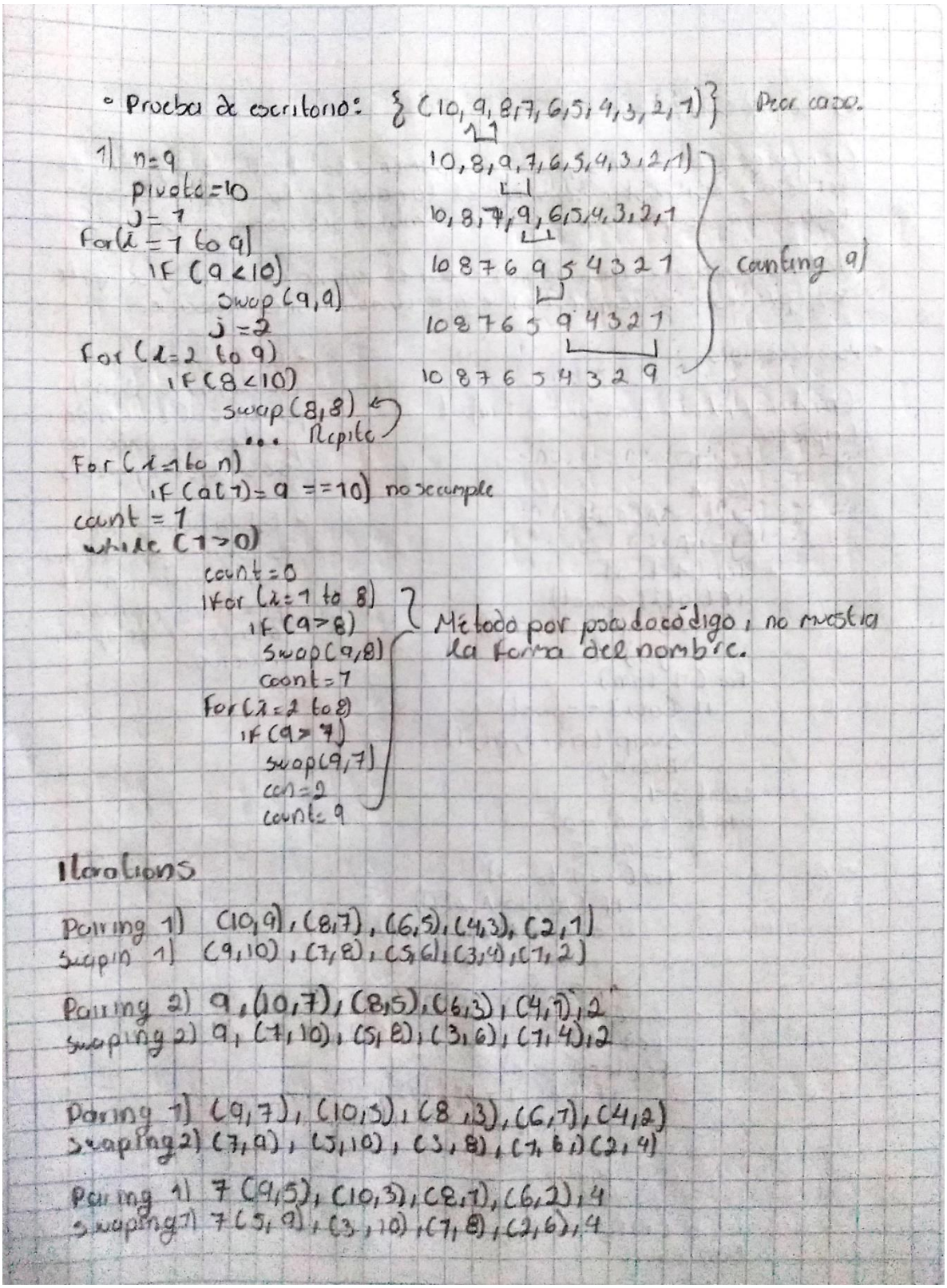
- Pseudócodigo.

```
(n, Array, A, i, count, pivot) : input
Empieza
        Get n
        Get A
        pivote = A[n/2]
        j = 1
        for(1 to n)
        IF(a[i] <pivot)
              Swap(a[i], a[j])
              j = j+1
        For(1 to n)
           IF (a[i] == pivot)
              swap (a[i], a[j])
              Break;
        count = 1
        while (count > 0)
              count = 0
              For(i = 1 to n-1)
                  IF (A[i] > A[i+1])
                    swap (A[i], A[i+1])
                    count = cont + 1
                    end if
              i = i + 2
              END FOR
              For (i = 2 to n)
                  IF (A[i] > A[i+1])
                  swap (A[i], A[i+1])
                  count = count + 1
                  end if
              i = i + 2
              EndFor
        END while
```

- Complejidad: O(n log n) o O(n²) por la utilización de pivotes o cuando no se utilizan pivotes

- Estabilidad: No es estable por la división de obtención del pivote ya que eso crea error de aproximación.

Díaz Hernández Marcos Bryan
Tarea N°1
Grupo: 09



Díaz Hernández Marcos Bryan
Tarea N°1
Grupo: 09

Paring (7,5), (9,3), (10,1), (8,2), (6,4)
swap (5,7), (3,9), (1,10), (2,8), (4,6)

Paring (5), (7,3), (9,1), (10,2), (8,4), 6
swaping (5), (3,7), (1,9), (2,10), (4,8), 6

Paring (5,3), (7,1), (9,2)(10,4), (8,6)
swaping (3,5), (1,7), (2,9), (4,10), (6,8)

paring 3(5,7), (7,2), (9,4), (10,6), 8
swaping (3), (1,5), (2,7), (4,9), (6,10), 8

paring (3,7), (5,2)(7,4), (9,6), (10,8)
swaping (1,3) (2,5) (4,7), (6,9), (8,10)

paring (1, (3,2), (5,4), (7,6) (9,8), 10)
swaping (1,2,3,4,5,6,7,8,9,10)

3) Quicksh sort

El algoritmo usa dos arrays adicionales, uno para la diferencia y otro para cuando se repiten los valores, todos inicializados con 0 (-1), se selecciona el mayor valor y con ese se hace la diferencia, en este arreglo de la diferencia es donde se hace el ordenamiento en base a esta misma diferencia con el ejemplo sera mas claro.

- Pseudocódigo

1) el maximo y minimo valor son determinados en el primer for

2) se inicializan el arreglo de diferencia y repeticion con -1

3) obtener la diferencia y colocar en el arreglo de diferencia y coloca en el arreglo de acuerdo a la posición.

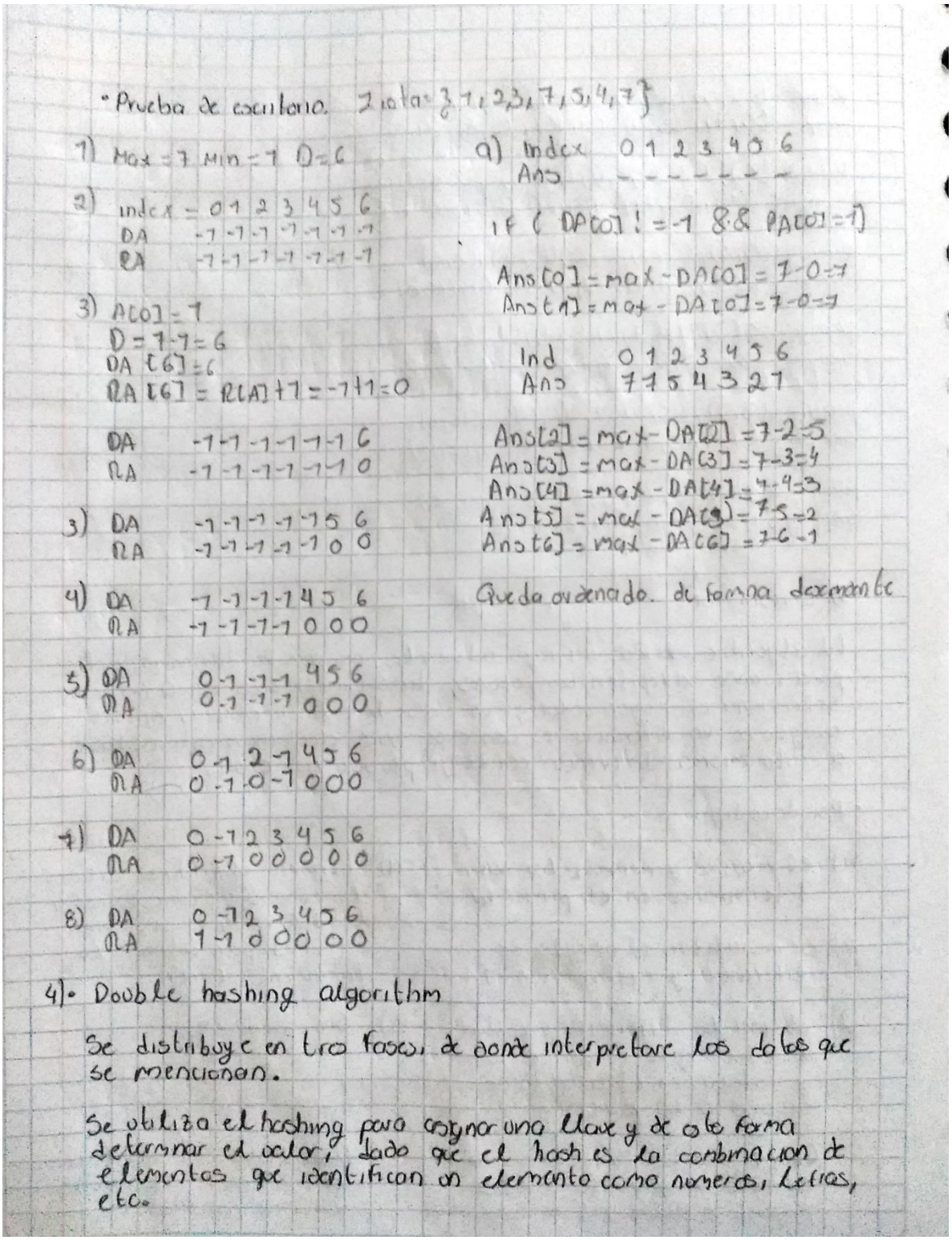
4) Utilizando el (if), se extraen los valores que corresponden a la diferencia y se utiliza un (for) para obtener la repeticion (si tiene)

5) Son ordenados.

- Complejidad: $O(n^2)$ debido a la cantidad de operaciones e incluso mayores a bubble sort en comparativa

- Estabilidad: Es estable pero no es eficiente incluso menos eficiente que bubble.

Díaz Hernández Marcos Bryan
Tarea N°1
Grupo: 09



• Prueba de escritorio. Lista = {7, 2, 3, 7, 5, 4, 7}

1) Max = 7 Min = 7 D = 6

2) index = 0 1 2 3 4 5 6
DA = -1 -1 -1 -1 -1 -1 -1
RA = -1 -1 -1 -1 -1 -1 -1

3) A[0] = 7
D = 7 - 7 = 6
DA[6] = 6
RA[6] = R[A] + 7 = -1 + 1 = 0

DA = -1 -1 -1 -1 -1 -1 6
RA = -1 -1 -1 -1 -1 -1 0

3) DA = -1 -1 -1 -1 -1 5 6
RA = -1 -1 -1 -1 -1 0 0

4) DA = -1 -1 -1 -1 4 5 6
RA = -1 -1 -1 -1 0 0 0

5) DA = 0 -1 -1 -1 4 5 6
RA = 0 -1 -1 -1 0 0 0

6) DA = 0 -1 2 -1 4 5 6
RA = 0 -1 0 -1 0 0 0

7) DA = 0 -1 2 3 4 5 6
RA = 0 -1 0 0 0 0 0

8) DA = 0 -1 2 3 4 5 6
RA = 1 -1 0 0 0 0 0

a) Index 0 1 2 3 4 5 6
Ans = - - - - - - -

if (DA[0] != -1 && RA[0] = 1)

Ans[0] = max - DA[0] = 7 - 0 = 7
Ans[1] = max - DA[0] = 7 - 0 = 7

Ind 0 1 2 3 4 5 6
Ans 7 7 5 4 3 2 7

Ans[2] = max - DA[2] = 7 - 2 = 5
Ans[3] = max - DA[3] = 7 - 3 = 4
Ans[4] = max - DA[4] = 7 - 4 = 3
Ans[5] = max - DA[5] = 7 - 5 = 2
Ans[6] = max - DA[6] = 7 - 6 = 1

Queda ordenado. de forma descendente

4) • Double hashing algorithm

Se distribuye en tres fases, de donde interpretare los datos que se mencionan.

Se utiliza el hashing para asignar una llave y de esta forma determinar el valor, dado que el hash es la combinacion de elementos que identifican un elemento como numeros, letras, etc.

Díaz Hernández Marcos Bryan
Tarea N°1
Grupo: 09

Fase 1) Realizar el hash, donde el algoritmo recolecta información del orden de los valores, para dividirlo en bloques y después calcular los elementos en cada bloque.

Ejemplo. TheTenth = $\frac{(Max - Min + 1)}{10}$ Crea el Counter_Array

El algoritmo realiza la ecuación para conocer los elementos de cada bloque e incrementado

Specified_Block = (Item) / TheTenth
Counter_array = [Specified_Block] ++

Fase 2) Usa los elementos / información del primer hash, y obtiene el bloque de cada elemento para calcular el máximo y mínimo

Max = (Specified_block + 1) * TheTenth
Min = Specified_Block * TheTenth

Después localiza los elementos por índices virtuales en su respectivo bloque con las sig. ecuaciones.

Float index = (elemen - Min) (Max - Min)
Index = indx * (Counter_Array)[Specified_Block])

Si el elemto es E_1 el bloque es B, y el conte_Array [B] = 5, lo localiza con cinco elementos, al final obtiene un arreglo de índices de tipos varios (enteros o flotantes)

Fase 3) Porle los bloques y los índices puedan contener uno o dos índices, cada compartimiento o espacio que contiene índices se divide en dos arreglos: Equal_arrays: enteros y los valores pedidos
Create-arrays: Float índices

- Complejidad: O(n)
  Dado que los valores de la entrada se comienza a corresponder con las operaciones que se realizan.
- Estabilidad: bueno se menciona que es estable por la aparición de los valores repetidos los envía a las mismas localidades y los coloca en el orden en que aparecieron.

Díaz Hernández Marcos Bryan
Tarea N°1
Grupo: 09

Ejemplo:

1) A = {5, 7, 17, 8, 9, 17, 4, 20, 19, 7, 2, 12, 15, 4, 8}

- Se obtiene el hash TheInt = $\frac{(20-1)+1}{10} = 2$

Posicion(5) = 5/2 = 3 Bloque
Se crea el conte. Array

CA = { 2 2 1 3 1 1 0 1 2 2 }

[1,3] [3,5] [5,7] [7,9] [9,11] [11,13] [13,15] [15,17] [17,19] [19,20] Rango (Bloque)

2) Localiza en array para ordenamiento por Quicksort.

Aaux = { 1 - - 4.4 - - - - - 8.8 - - - - - - - 11.17 20 }
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Bucket = { - 2 - - 5 - 7 - - - - 9 12 - - 15 - - - 19 - }

Se envían los valores al Quicksort para ordenar.

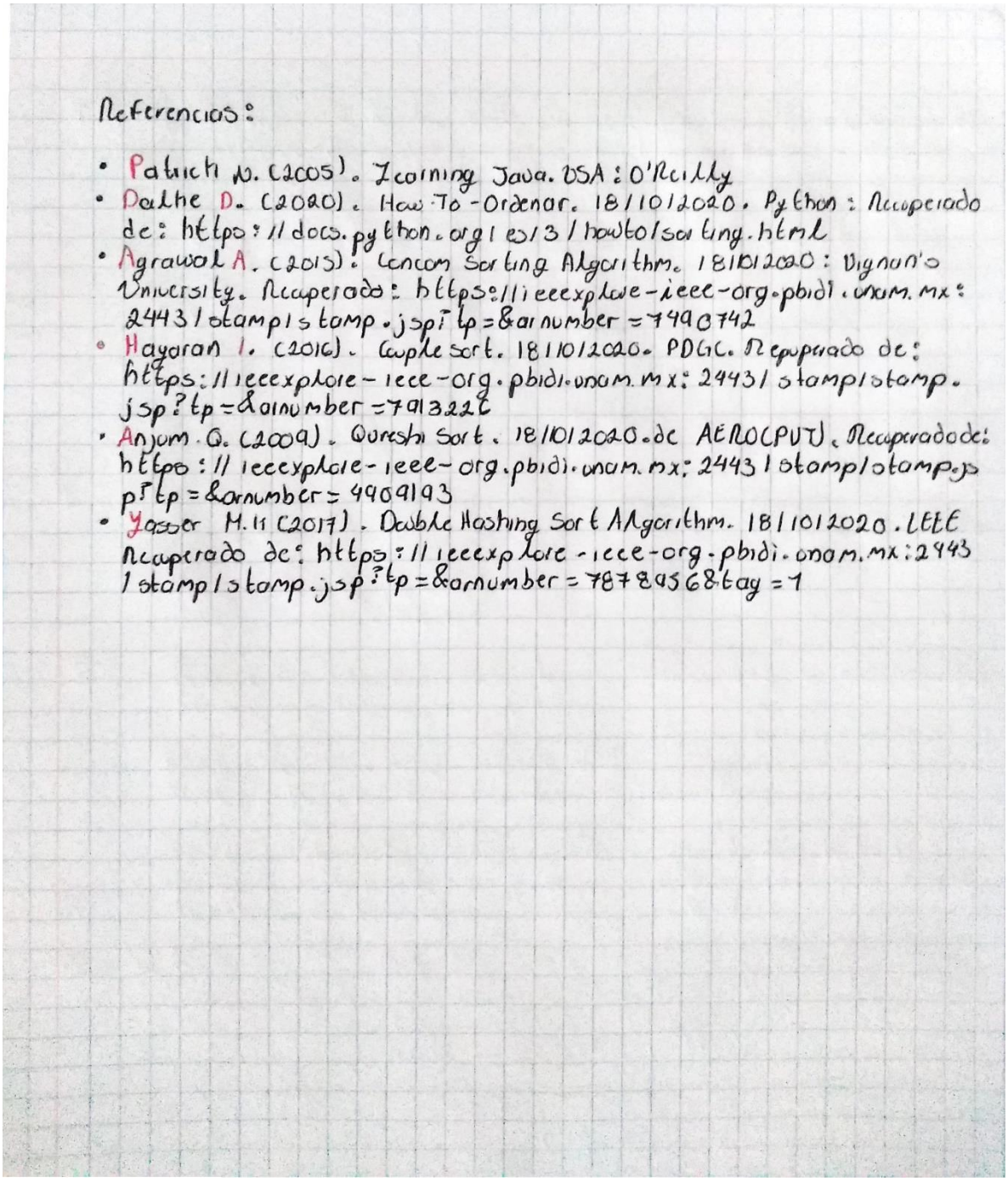
- Algoritmo de ordenamiento de los lenguajes:

- Java: en la colección de utilidades se encuentra los operaciones de ordenamiento, que son: - sort (list)
 - sor (list, c)
 El método que es utilizado es el Timsort que es una combinación de Insertion y Mergesort donde se aplica primero el insertion y después el Merge con una complejidad O(N log(N)).

- Python: en ese se localiza en el metodo list.sort() o sorted() que acepta todo tipo de ordenable, al igual que Java utiliza el TimSort que es una combinación de inserbien y Mergesort.

- c++: la función integrada std::sort() de la libreria estandar utiliza introSort que es una combinacion del Quicksort, Heapsort e insertion con una complejidad: O(n log(N)).

Díaz Hernández Marcos Bryan
Tarea N°1
Grupo: 09



Referencias:

- Patricia N. (2005). Learning Java. USA: O'Reilly
- Dalthe D. (2020). How-To-Ordenar. 18/10/2020. Python: Recuperado de: https://docs.python.org/es/3/howto/sorting.html
- Agrawal A. (2015). Concom Sorting Algorithm. 18/10/2020: Vignan's University. Recuperado: https://ieeexplore-ieee-org.pbidi.unam.mx:2443/stamp/stamp.jsp?tp=&arnumber=7490742
- Hayaran I. (2016). Cuple sort. 18/10/2020. PDGC. Recuperado de: https://ieeexplore-ieee-org.pbidi.unam.mx:2443/stamp/stamp.jsp?tp=&arnumber=7913221
- Anjum Q. (2009). Qureshi Sort. 18/10/2020. de AERO(PUT). Recuperado de: https://ieeexplore-ieee-org.pbidi.unam.mx:2443/stamp/stamp.jsp?tp=&arnumber=4909193
- Yasser M. (2017). Double Hashing Sort Algorithm. 18/10/2020. IEEE Recuperado de: https://ieeexplore-ieee-org.pbidi.unam.mx:2443/stamp/stamp.jsp?tp=&arnumber=7878956&tag=1