Tema: Algoritmos de ordenación

Nombre: Alexandra Valeria Hernández Quintero **Matricula:** 1663507

Unidad de aprendizaje: Matemáticas Computacionales **Fecha:** 1/09/2017

Resumen: Este reporte trata sobre de los 4 tipos de algoritmos de ordenación que vimos en clase: Bubble sort, insercion sort, selection sort y quick sort.

Bubble sort:

El tipo de algoritmo “burbuja” (su traducción en español), es el más ineficiente y “lento” podríamos decir, como ya vimos anteriormente en clases, en si no es que tarde más que los otros algoritmos, más bien que hace más operaciones, su complejidad es de n^2 , el peor de los casos seria que el arreglo estaría ordenado al revés, es decir que si se quiere ordenar de manera ascendente, el arreglo este ordenado de forma descendente y viceversa. En si lo que hace este algoritmo es agarrar los primeros dos números compararlos,” preguntar” cuál de los dos es el menor (o mayor), para luego moverlos o dejarlos donde estaban, según sea el caso, asi hasta terminar de comparar todos los pares de números, y eso solo es en la primera pasada, porque tiene que dar varias “pasadas”, para ordenar los números de un arreglo.

Ejemplo:

```
1  def bubbleSort(alist):
2      for passnum in range(len(alist)-1,0,-1):
3          for i in range(passnum):
4              if alist[i]>alist[i+1]:
5                  temp = alist[i]
6                  alist[i] = alist[i+1]
7                  alist[i+1] = temp
8
9  alist = [54,26,93,17,77,31,44,55,20]
10 bubbleSort(alist)
11 print(alist)
12
```

Insertion sort:

El tipo de algoritmo “inserción” (su traducción en español), también es uno de los más ineficientes, pero este a diferencia de Bubble, no empieza a comparar con el primer número (elemento del arreglo), más bien compara a partir del segundo número, entonces si quisiéramos ordenar de forma ascendente, “pregunta” si el segundo es menor que el anterior, y si es lo es, lo desplaza a donde debe de ir , después toma el tercer elemento y los compara con el primer y segundo elemento, en pocas palabras va “preguntando” si es menor (o mayor) según sea el caso, para luego ir desplazando números hasta llegar a él orden deseado. Pero mientras esta en el proceso, “elimina “ los valores comparados que no cumplían la condición(solo temporalmente), y luego vuelve a agregarlos en las posición correcta, de ahí el nombre de inserción, su complejidad al igual que Bubble es de n^2.

Ejemplo:

```
def insertionSort(alist):
   for index in range(1,len(alist)):

     currentvalue = alist[index]
     position = index

     while position>0 and alist[position-1]>currentvalue:
         alist[position]=alist[position-1]
         position = position-1

     alist[position]=currentvalue

alist = [54,26,93,17,77,31,44,55,20]
insertionSort(alist)
print(alist)

```

Selection sort:

El tipo de algoritmo “selección” (su traducción en español), este en comparación con inserción y Bubble, realiza menos operaciones, ya que selection busca el valor más grande desde el primera pasada y después de completar el pase lo pone en la posición correcta, en la segunda pasada, toma el segundo valor más grande y lo ubica en las posición deseada, de esta forma continua hasta que tiene todos los elementos del arreglo ordenados, es más sencillo de visualizarlo que un Bubble o un inserción, se puede ver que el orden de selection hace el mismo número de comparaciones que el tipo de bubble , por lo tanto su complejidad también es n^2. Pero debido a la reducción en el número de intercambio, selection podría decirse que “tarda” menos que un bubble.

Ejemplo:

```
def selectionSort(alist):
   for fillslot in range(len(alist)-1,0,-1):
       positionOfMax=0
       for location in range(1,fillslot+1):
           if alist[location]>alist[positionOfMax]:
               positionOfMax = location

       temp = alist[fillslot]
       alist[fillslot] = alist[positionOfMax]
       alist[positionOfMax] = temp

alist = [54,26,93,17,77,31,44,55,20]
selectionSort(alist)
print(alist)

```

Quick sort:

El tipo de algoritmo “clasificación rápida” (su traducción en español), es la más eficaz que vimos en clase, y no se parece a ningunos de los otros 3 algoritmos de ordenación mencionados anteriormente, este toma un número(elemento del arreglo), este es cualesquiera de los elementos(pero lo ideal sería justo el de la mitad), y ese va ser nuestro “pivote”, a partir de ese “pivote”, separa el arregló en dos partes, unas vez que ya está partido el arreglo, pone de un lado los menores(o mayores) según sea el caso y del otro lado los mayores(o menores) al pivote, de esta forma ira ordenando el arreglo hasta llegar a la forma deseada, su complejidad es de $n\log_2(n)$ la cual es menor que n^2, y es por eso que esta es mucho más eficiente que las 3 anteriores .

Ejemplo:

```
def quickSort(alist):
   quickSortHelper(alist,0,len(alist)-1)

def quickSortHelper(alist,first,last):
   if first<last:

       splitpoint = partition(alist,first,last)

       quickSortHelper(alist,first,splitpoint-1)
       quickSortHelper(alist,splitpoint+1,last)


def partition(alist,first,last):
   pivotvalue = alist[first]

   leftmark = first+1
   rightmark = last

   done = False
   while not done:

       while leftmark <= rightmark and alist[leftmark] <= pivotvalue:
           leftmark = leftmark + 1
```

```
       while alist[rightmark] >= pivotvalue and rightmark >= leftmark:
           rightmark = rightmark -1

       if rightmark < leftmark:
           done = True
       else:
           temp = alist[leftmark]
           alist[leftmark] = alist[rightmark]
           alist[rightmark] = temp

   temp = alist[first]
   alist[first] = alist[rightmark]
   alist[rightmark] = temp


   return rightmark

alist = [54,26,93,17,77,31,44,55,20]
quickSort(alist)
print(alist)
```

Conclusión:

En lo personal me parecen muy sencillos los tipos de ordenación, claro está que falta ponerlos a prueba en Python, el único que si me pareció un poco más difícil de visualizar fue el Quick sort, pero si debo aceptar que es el más eficiente y el que menos ·pasadas hace, si hiciéramos la prueba e escritorio con los otros 3 si nos tardaríamos mucho más. El Bubble sort aunque es el más ineficiente, está muy sencillo de comprender y si bastante tedioso. Insertion fue el que menos me gusto, eso de “eliminar” (temporalmente), siento que es un paso más y se puede olvidar de usar otra variable para guardarlo temporalmente. Selection es algo asi como la evolución de Bubble, lo que hace la diferencia, es que selection no hace tantas “pasadas”, para ordenar un arreglo.

Bibliografía:

Miller, Ranum, *Solución de problemas con algoritmos y estructuras de datos usando Python*