Pablo Emilio Lopez Avila Mat. 1740087 Matemáticas computacionales.

Algoritmos de Ordenamiento.

Algoritmo Quicksort:

Es un algoritmo de ordenamiento que se descubrió “por casualidad” buscando algoritmos que buscaban facilitar la búsqueda de palabras en el diccionario; Actualmente en su tipo es la mejor opción en cuanto a **complejidad** ya que en el mejor de sus casos su tiempo de ejecución es $On(ln(n))$ y en el peor On^2. Sin tanta formalidad lo que hace Quicksort es tomar un número que debería ser no tan grande en la lista ni tan pequeño, el cual servirá como punto de comparación entre 2 grupos de números, a este número se le llama “Pivote”. Los números más pequeños o iguales al pivote formaran otra lista, el elemento pivote se aparatará y la tercer lista la cual contiene números mayores o iguales al pivote. Estas nuevas listas irán formando otras pequeñas listas en las cuales se tomarán nuevos pivotes hasta que se ordenen en su totalidad los elementos.

Este es un pseudocodigo para utilizar el algoritmo desde una lista a ingresar.

```
def quicksort(L, first, last):
   # definimos los índices y calculamos el pivote
   i = first
   j = last
   pivote = (L[i] + L[j]) / 2

   # iteramos hasta que i no sea menor que j
   while i < j:
      # iteramos mientras que el valor de L[i] sea menor que pivote
      while L[i] < pivote:
         # Incrementamos el índice
         i+=1
      # iteramos mientras que el valor de L[j] sea mayor que pivote
      while L[j] > pivote:
         # decrementamos el índice
         j-=1
      # si i es menor o igual que j significa que los índices se han cruzado
      if i < = j:
         # creamos una variable temporal para guardar el valor de L[j]
         x = L[j]
         # intercambiamos los valores de L[j] y L[i]
```

```
        L[j] = L[i]
        L[i] = x
        # incrementamos y decrementamos i y j respectivamente
        i+=1
        j-=1

    # si first es menor que j mantenemos la recursividad
    if first < j:
        L = quicksort(L, first, j)
    # si last es mayor que i mantenemos la recursividad
    if last > i:
        L = quicksort(L, i, last)

    # devolvemos la lista ordenada
    return L
```

Algoritmo Selection

Es un algoritmo de ordenamiento, el cual no es muy eficiente del todo, su **complejidad** varia del número de elementos a ordenar, representado como O= $\frac{n^2+n}{2}$ **,** de manera simple este algoritmo se basa en leer la lista completa y tomar el valor más mínimo y ponerlo en la primer posición intercambiando el lugar con el elemento en esa posición , seguido de volver a leer ahora toda la lista a partir del segundo elemento buscando el segundo más mínimo y ponerlo en esa posición sustituyendo al elemento de la segunda posición de esa manera hasta que se ordena completamente la lista.

```
def selectionsort(arr): #definimos nuestra función
    n=len(arr) # len(arr) es la longitud del arreglo
    for i in range(0,n-1): #se utiliza n-1 ya que no se necesita la comparación propia.
        posición=i
        valor=arr[i]
        for j in range (i,n):
            if arr[j]< valor:
                valor=arr[j]
```

```
                posición=j
    arr[posición]=arr[i]
    arr[l]=valor

return arr
```

Algoritmo Bubble

Es un algoritmo basado en un ordenamiento muy simple en el cual va revisando cada elemento del arreglo o lista, en la cual toma el primer elemento y lo va comparando de izquierda a derecha viendo si ese primer elemento es mayor a los que se va encontrando, si encuentra un elemento mayor a él mismo, deja ahí el primer término y continua la comparación con ese elemento mayor, así sigue hasta acabar con los elementos en la lista o arreglo, lo que hace necesario que se repita varias veces la misma acción. Su Complejidad va dependiendo de sus elementos representado como O= $\frac{n^2 - n}{2}$. Su nombre proviene de una representación de una burbuja de gas la cual crece conforme avanza, así es con el elemento que vamos comparando en el arreglo o lista.

```
def burbuja(A):
    for i in range (1,len(A)):
        for j in range (0,len(A)-1):
            if (A[j+1]<A[j]):
                aux=A[j]
                A[j]=A[j+1]
                A[j+1]=aux

        print(A)
```

Algoritmo Insertion

Es un algoritmo muy común debido a que se parece mucho a la manera en que todos nosotros como humanos ordenamos las cosas, toma el primer elemento el cual se compara con el segundo, se hace el cambio si es necesario, ahora que tenemos dos elementos ordenados los tomamos de referencia a ambos, el tercer elemento lo comparamos con el mayor de los elementos que teníamos para saber si es mayor o menor a este si es mayor se deja dónde está, si es menor al segundo elemento pero mayor al primero se pone entre los dos, en resumidas palabras toma a esos dos números y los compara uno por uno con el elemento que le sigue al mismo, para así ir ordenando la lista p arreglo. La complejidad de este algoritmo siempre es $O=n^2$

```
def ord_insercion(lista):
    """ Ordena una lista de elementos según el método de inserción.
        Pre: los elementos de la lista deben ser comparables.
        Post: la lista está ordenada. """

    # i va desde la primera hasta la penúltima posición de la lista
    for i in xrange(len(lista)-1):
        # Si el elemento de la posición i+1 está desordenado respecto
        # al de la posición i, reubicarlo dentro del segmento (0:i]
        if lista[i+1]< lista[i]:
            reubicar(lista, i+1)

        print "DEBUG: ", lista

def reubicar(lista, p):
    """ Reubica al elemento que está en la posición p de la lista
        dentro del segmento (0:p-1].
        Pre: p tiene que ser una posicion válida de lista. """

    # v es el valor a reubicar
    v = lista[p]

    # Recorre el segmento (0:p-1] de derecha a izquierda hasta
    # encontrar la posición j tal que lista(j-1] <= v < lista(j].
    j = p
    while j > 0 and v < lista[j-1]:
        # Desplaza los elementos hacia la derecha, dejando lugar
        # para insertar el elemento v donde corresponda.
```

```
        lista[j] = lista[j-1]
        # Se mueve un lugar a la izquierda
        j -= 1

    # Ubica el valor v en su nueva posición
    lista[j] = v
```

Proviene: http://librosweb.es/libro/algoritmos_python/capitulo_19/ordenamiento_por_insercion.html