

**Facultad de Ciencias Físico Matemáticas**

**FCFM**

***Reporte***

***Algoritmos de Ordenamiento***

**Materia:** Metodología Científica  
**Maestro:** José Hernández

**Alumna:** Brenda Espinosa Coronado

**Matricula:** 1729180

**Carrera:** Lic. Matemáticas

**Grupo:** 01 **Aula**: 120

9 de Abril del 2018

**Algoritmos de ordenamiento**

Los algoritmos de ordenamiento nos permiten ordenar un grupo o colección de valores que poseen una relación de orden. El ordenar un grupo de datos significa mover los datos o sus referencias para que queden en una secuencia tal que represente un orden, el cual puede ser numérico, alfabético o incluso alfanumérico, ascendente o descendente.

**Tipos de algoritmos de ordenamiento**

Existen distintos métodos de ordenamiento con distintas características y complejidad. Para saber que algoritmo se debe usar en cada situación, los factores importantes son el tiempo de ejecución de un algoritmo y la memoria que este usa. Los algoritmos de los cuales hablaremos son:

* Bubble Sort
* Insertion Sort
* Selection Sort
* Quick Sort

***Bubble Sort***

El bubble sort, también conocido como ordenamiento burbuja, toma dos elementos de la izquierda y los compara tomando el elemento mayor y este se va comparando con los siguientes hasta que llega a su lugar. Esto se repite cuantas veces sea necesario para terminar de ordenar todo nuestro conjunto. El tiempo de ejecuciones de n^2

Código:

def Bubble(V):  
 ord = True  
 it = 0  
 while ord == True:  
 ord = False  
 it = it + 1  
 for actual in range(0, len(V) - it):  
 if V[actual] > V[actual + 1]:  
 ord = True  
 # Intercambiamos los dos elementos  
 V[actual], V[actual + 1] = \  
 V[actual + 1],V[actual]  
 return V

***Selection Sort***

El método de ordenamiento por selección consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenar todo el arreglo. Su tiempo de ejecución es en promedio n^2

Código:

def Selection(V):  
 nb = len(V)  
 for actual in range(0,nb):  
 menor = actual  
 for j in range(actual+1,nb) :  
 if V[j] < V[menor] :  
 menor = j  
 if min is not actual :  
 temp = V[actual]  
 V[actual] = V[menor]  
 V[menor] = temp  
 return V

***Insertion Sort***

El insertion sort, también conocido como ordenamiento de inserción, consiste en insertar cada elemento (partiendo por el segundo) en la posición correcta, desplazando cada elemento que sea mayor a su izquierda, hasta encontrar un elemento que sea menor, insertando el elemento a la derecha de este. El tiempo de ejecución (al igual que el ordenamiento burbuja) es de n^2

Código:

def Insertion(V):  
 for i in range(1,len(V)):  
 actual = V[i]  
 j = i  
 while j>0 and V[j-1]>actual:  
 V[j]=V[j-1]  
 j = j-1  
 V[j]=actual  
 return V

***Quick Sort***

Este método está basado en la teoría de divide y vencerás. Lo que hace este algoritmo es dividir recursivamente el vector en partes iguales, indicando un pivote que nos permite dividir nuestra lista. Una vez dividida, lo que hace, es dejar todos los mayores que el pivote a su derecha y todos los menores a su izquierda, ya que tenemos unas sublistas, a estas les aplicamos el mismo procedimiento hasta que estén acomodados y así poder formar nuestro nuevo arreglo ya ordenado. Su tiempo de ejecución es de n (log n), pero todo depende del pivote escogido

Código:

def QuickSort(V):  
 if not V:  
 return []  
 else:  
 P = V[-1]  
 menor = [x for x in V if x < P]  
 mayor = [x for x in V[:-1] if x >= P]  
 return QuickSort(menor) + [P] + QuickSort(mayor)

***Realizamos una comparación entre nuestros 4 métodos de ordenamiento ya mencionados y obtenemos lo siguiente:***

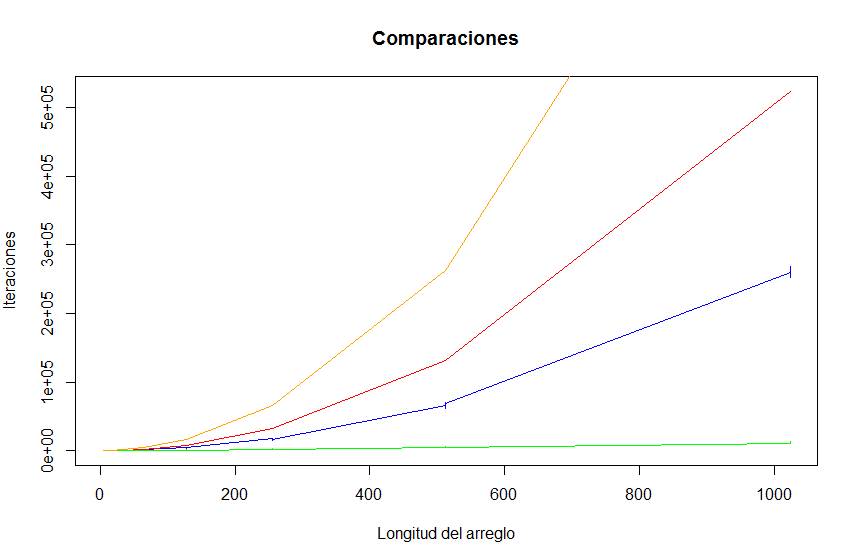
***Individualmente***

|  |  |
| --- | --- |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

De lado izquierdo, podemos observar que la gráfica nos muestra la tendencia del número de iteraciones con respecto a la longitud del arreglo. Por otra parte, si observamos la gráfica de nuestro lado derecho, esta nos muestra la misma tendencia solo que ella nos da debajo la longitud del arreglo con la que se trabajó, también nos muestra unas pequeñas cajas y de ellas podemos observar los máximos y mínimos de iteraciones en esa longitud y aparte de esto nos muestra unos pequeños círculos que nos genera un valor que se sale de nuestro máximo o mínimo el cual es único y por eso no se marca como un límite.

***Comparando***

n^2 operaciones

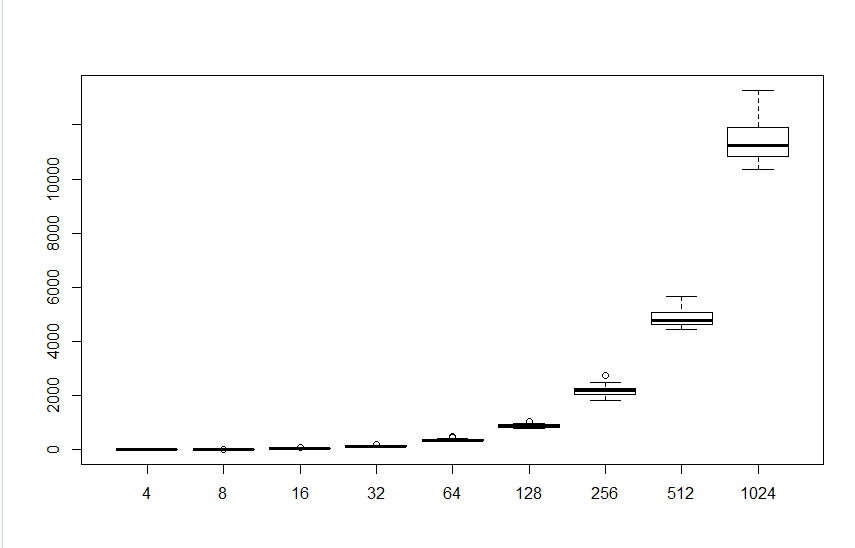
Selection

Bubble

Intersertion

Quick Sort

Aquí realizamos 270 comparaciones con 9 tamaños de arreglos diferentes y por cada uno realizamos 30 veces el experimento y esto fue lo que resulto



Grafica de cajas

Analizando nuestras graficas de comparaciones, podemos observar que tanto el método de selección como el de burbuja tardan exactamente lo mismo y son los que hacen el mayor número de iteraciones seguidas del método de inserción mientras que el método de Quick Sort hace mucho menos iteraciones con respecto a mis otros tres métodos. De aquí podemos concluir que el método de Quick Sort es el más eficiente ya que hace el menor número de iteraciones