**Universidad Nacional**

**Proyecto programado (Partition)**

**Asignatura:**

Estructuras lógicas de programación.

EIF-293-1-2018.

**Tema:**

Particiones.

**Profesor:** Carlos Loría Sáenz.

**Grupo 4:**

Kevin Artavia Arguedas ID:402390969 correo: kevinartavia29@gmail.com HORARIO: 1:00 pm.

Jose Gerardo Matarrita Hernandez ID:402380565 correo: josemh207@gmail.com HORARIO: 1:00 pm.

Oscar Felipe Rodríguez Núñez ID:117110958 correo: oscfel98@gmail.com HORARIO: 1:00 pm.

Esteban Alonso Salas Chaves ID:207750936 correo: stebanortizserrano@gmail.com HORARIO: 10:00 am.

# Índice

Contenido

[Índice 2](#_Toc511934410)

[Introducción. 3](#_Toc511934411)

[Marco de estudio. 4](#_Toc511934412)

[Desarrollo de soluciones. 5](#_Toc511934413)

[QuickSort 5](#_Toc511934414)

[RandomQuickSort 6](#_Toc511934415)

[Sort5 7](#_Toc511934416)

[Medians 8](#_Toc511934417)

[Análisis teórico y empírico. 9](#_Toc511934418)

[QuickSort General 9](#_Toc511934419)

[RandomQuickSort 10](#_Toc511934420)

[Sort5 11](#_Toc511934421)

[Medians 12](#_Toc511934422)

[Conclusión 13](#_Toc511934423)

[Bibliografía/Referencias 14](#_Toc511934424)

# Introducción.

Con el paso del tiempo la informática ha avanzado rápidamente y de una manera evolutiva.  Jessica Rivero Espinosa dijo

“La computadora fue inventada para facilitar el trabajo intelectual. Si el hombre tiene algún problema, el diseñador define el algoritmo que resuelve el problema, el programador lo codifica en un lenguaje de programación, el cual la computadora es capaz de "entender", luego la computadora ejecuta el algoritmo expresado como programa en el lenguaje de programación en cuestión, y entrega al hombre la respuesta.”

En este reporte se analizarán dos métodos (“Quicksort” y “Medians”). Se van a analizar a fondo con el fin de entender su función, su uso y su tiempo de corrida.

# Marco de estudio.

Dado que el tema principal de este reporte es el análisis de algoritmos, es necesario demostrar conceptos básicos para apoyar la literatura interpretativa. Para empezar, entenderemos el concepto de informática. Según la revista ARQHYS(2011):

**“**La informática, es la ciencia aplicada que abarca el estudio y aplicación del tratamiento automático de la informática, utilizando sistemas computacionales, generalmente implementados como dispositivos electrónicos. También está definida como el procesamiento automático de la información. El Diccionario de la real Academia Española, define la informática como el conjunto de conocimientos científico y técnico, que hacen posible el tratamiento automático de la información por medio de ordenadores.”

Con esto queda claro que en el uso de dispositivos electrónicos es algo esencial en la informática, pero aquí es donde surge una duda ¿Cómo entiende un dispositivo electrónico lo que un humano le quiere decir? Esto sucede gracias a la creación de lenguajes de computación. Según Jorge A. Saavedra (2007):

“Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos, respectivamente.”

Teniendo en claro el concepto de “Informática” y de “Lenguaje de programación” podemos analizar dos algoritmos esenciales como lo son el “QuickSort” y el “Medianas” en los cuales se analizará su tiempo de corrida, su función y un análisis de las estrategias utilizadas en ellos. Según Andrés Capouya (2009):

**“**Un Algoritmo, se puede definir como una secuencia de instrucciones que representan un modelo de solución para determinado tipo de problemas. O bien como un conjunto de instrucciones que realizadas en orden conducen a obtener la solución de un problema. Por lo tanto, podemos decir que es un conjunto ordenado y finito de pasos que nos permite solucionar un problema. Los algoritmos son independientes de los lenguajes de programación. En cada problema el algoritmo puede escribirse y luego ejecutarse en un lenguaje de diferente programación.”

# Desarrollo de soluciones.

## QuickSort

El código mostrado abajo, es el diseño general de QuickSort. Dicho código toma un punto de partición (pivote), que en este caso es el primer elemento de la lista y después utiliza dos índices (‘j’ e ‘i’) para comparar los números y en caso de que se cumplan las condiciones ya establecidas se van a cambiar de posiciones entre el índice ‘i’ y el índice ‘j’ mediante una variable temporal (temp), llevando a cabo el ordenamiento del vector.

def quickSort(a):  
   quickSorting(a,0,len(a)-1)  
def quickSorting(a,inicio,final):  
   if inicio<final:  
       pp = particion(a,inicio,final)  
       quickSorting (a, inicio,pp-1)  
       quickSorting (a, pp+1, final)  
def particion(a , inicio, final):  
   pivote = a[inicio]  
   j = inicio+1  
   k = final  
   done = False  
   while not done:  
       while j <= k and a[j] <= pivote:  
           j = j + 1  
       while a[k] >= pivote and k >= j:  
           k = k -1  
       if k < j:  
           done = True  
       else:  
           temp = a[j]  
           a[j] = a[k]  
           a[k] = temp  
   temp = a[inicio]  
   a[inicio] = a[k]  
   a[k] = temp  
   return k

## RandomQuickSort

Este es el código de QuickSort utilizando un pivote aleatorio. Dicho código toma un elemento aleatorio que está entre ‘inicio’ y ‘fin’ (mediante el uso de la librería random, la cual se incluye con el prefijo ran), hace un ordenamiento donde coloca el elemento (pivote) al final, luego llama al método ‘particionRandom’ para conseguir un elemento nuevo. Este nuevo elemento lo toma como punto de partición, luego llama a ‘RandomQuicksort’ desde ‘inicio’ hasta la partición -1 (‘pp-1’) y desde ‘inicio’ hasta la partición +1(‘pp+1’) y ordena las particiones por separado de manera recursiva.

import random as ran  
  
def randomQuickSort(a,inicio,fin):  
    if inicio<fin:  
        randomN=ran.randint(inicio,fin)  
        temp=a[fin]  
        a[fin]=a[randomN]  
        a[randomN]=temp  
        pp=particionRandom(a,inicio,fin)  
        randomQuickSort(a,inicio,pp-1)  
        randomQuickSort(a,pp+1,fin)  
def particionRandom(a, inicio, fin):  
    randomN=ran.randint(inicio,fin)  
    temp=a[fin]  
    a[fin]=a[randomN]  
    a[randomN]=temp  
    i=inicio-1  
    for j in range(inicio,fin):  
        if a[j]<a[fin]:  
            i=i+1  
            temp=a[i]  
            a[i]=a[j]  
            a[j]=temp  
    temp=a[i+1]  
    a[i+1]=a[fin]  
    a[fin]=temp  
    return i+1

## Sort5

En este código se hizo uso del método de burbuja para el ordenamiento de los elementos. Esto es posible mediante una variable temporal que permite un intercambio de posiciones en caso de que se cumpla que el lugar j es mayor que el lugar j+1

def sort5(a):  
    for i in range(len(a)-1,0,-1):  
        for j in range(i):  
            if a[j]>a[j+1]:  
                temp = a[j]  
                a[j] = a[j+1]  
                a[j+1] = temp

## Medians

Este código utiliza el método DyC. Divide el vector en vectores más pequeños de 5 elementos. Realiza el ordenamiento de esos 5 elementos y saca la mediana. Esas medianas las guarda en un nuevo vector de medianas y saca la mediana de esa mediana. Gracias a este método se puede pedir el n-ésimo valor más pequeño de un arreglo desordenado, con la única condición de que sus valores no sean repetidos.

def medianOfMedians(a, n):

listasDivididas = [a[j:j+5] for j in range(0, len(a), 5)]

listaMedianas = [sorted(listaDividida)[len(listaDividida)//2] for listaDividida in listasDivididas]

if len(listaMedianas) <= 5:

pp = sorted(listaMedianas)[len(listaMedianas)//2]

else:

pp = medianOfMedians(listaMedianas, len(listaMedianas)//2)

menoresQuePP = [j for j in a if j < pp]

mayoresQuePP = [j for j in a if j > pp]

k = len(menoresQuePP)

if n < k:

return medianOfMedians(menoresQuePP,n)

elif n > k:

return medianOfMedians(mayoresQuePP,n-k-1)

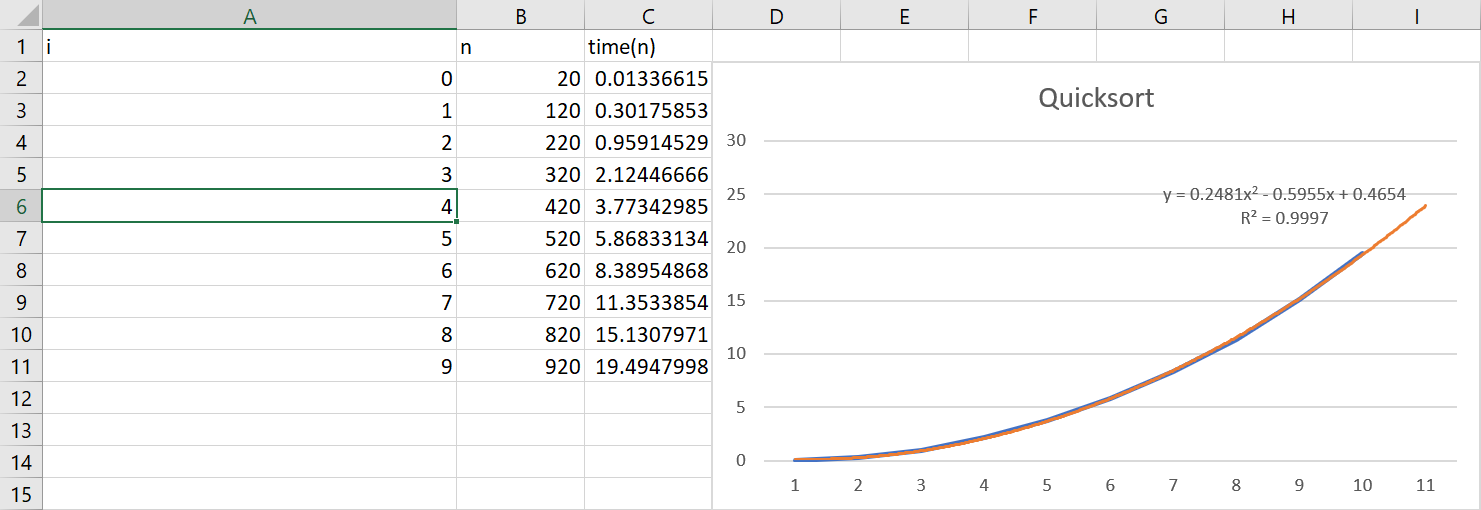
else:

return pp

# Análisis teórico y empírico.

## QuickSort General

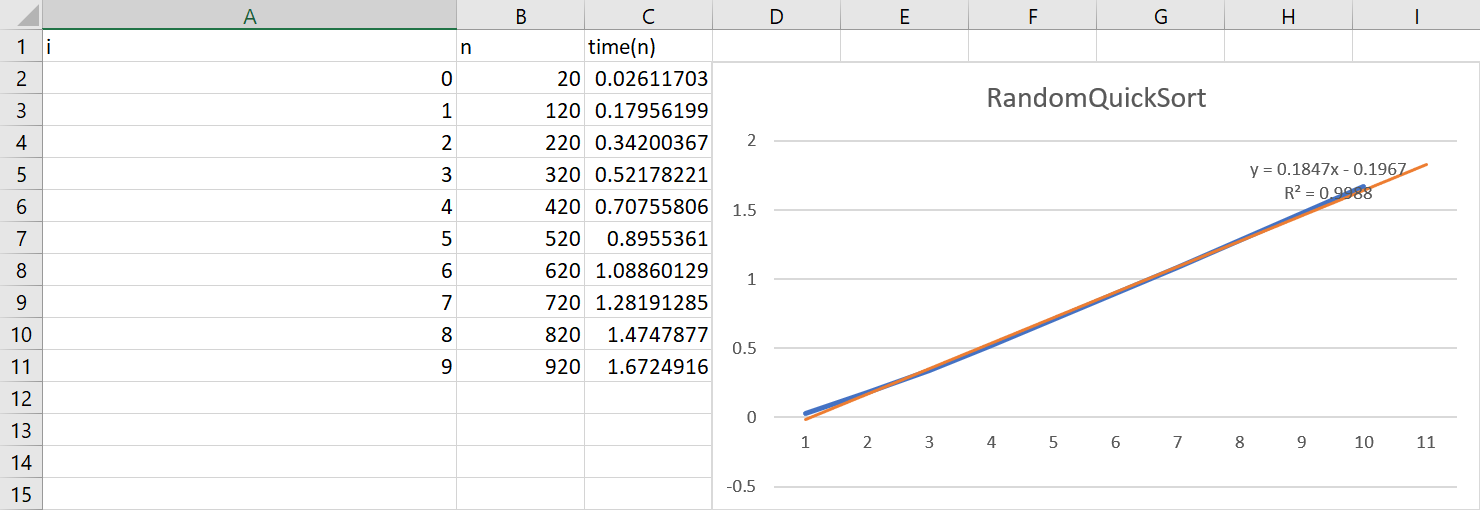




O(n2)

## RandomQuickSort

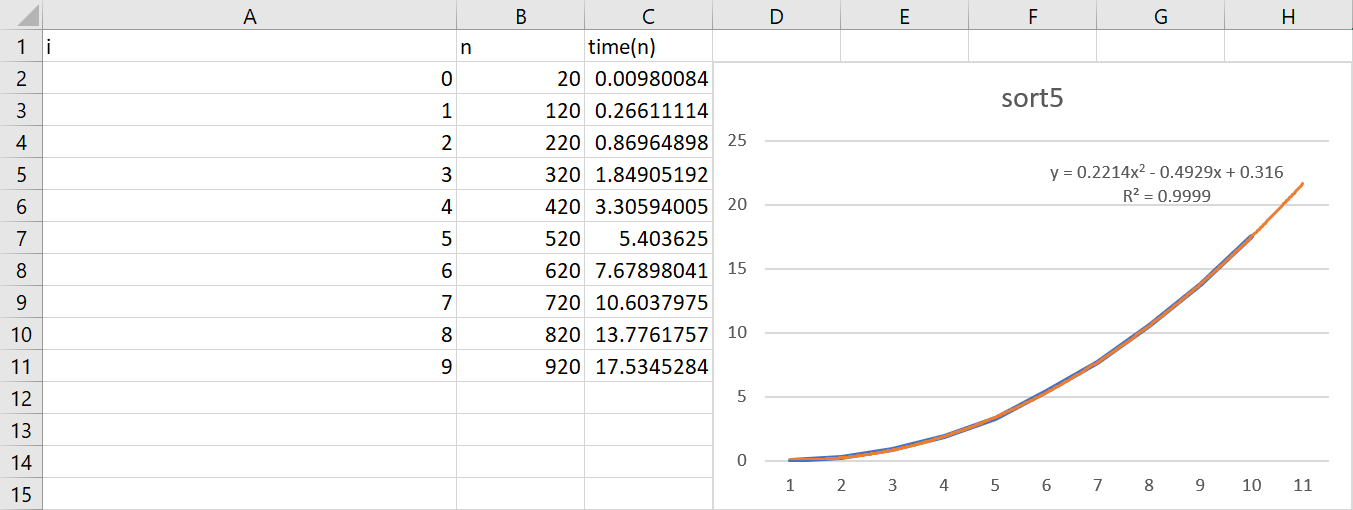
https://lh5.googleusercontent.com/4PNc5gYmE4GUrblE9l_UJ_g71gVOdB7nilF5XS90m2ARNIMfpybA-Oci4QhIQENlhRcqeXv5L-kVTxlYPBTaGCEy7ulZymA4yIP-Ja0GIRgpdpzGvz3AqVSlHUUB1zIvqiRh6MTe



O(nlog(n))

## Sort5





* Tamaño de los datos:

n=len(a)

* Operaciones de interés

=, >, for

Todas con un tiempo constante C

* RR

C\*[(n-1) +(n-2) +…+(n-k)]

C\*

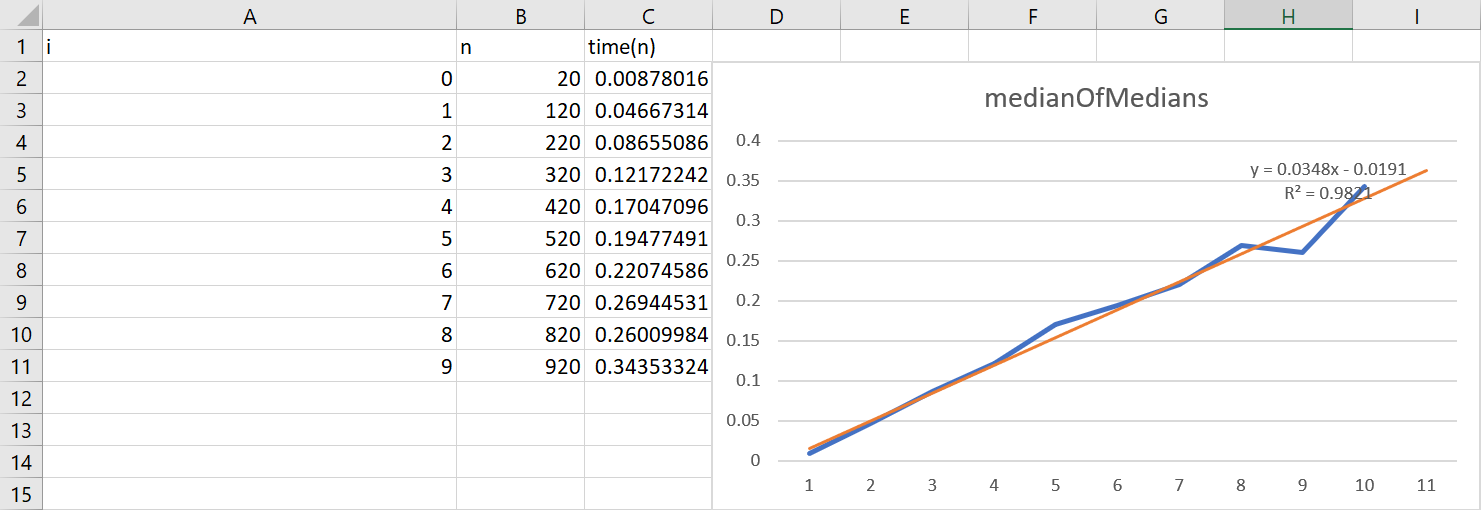


* O(.)

O(n2)

## Medians





# Conclusión

En este proyecto programado, hemos podido estudiar con éxito el algoritmo QuickSort y el Medians. Gracias a Python, hemos sido capaces de evaluar los algoritmos para así poder obtener el tiempo de corrida con diferentes muestras; esto con el fin de crear un CSV que nos permitiera mostrar un gráfico con el tiempo de corrida en cada caso probado.

Dichos análisis nos permiten concluir que existen diferentes algoritmos para realizar una tarea (por supuesto, unos más eficientes que otros); lo que demuestra que la programación es un área muy flexible, ya que permite resolver un problema de muchas maneras diferentes. Y gracias a optimizar dichas soluciones, se puede obtener un resultado favorable en relación al tiempo que requiere su ejecución.

En fin, el estudio de algoritmos nos enseña que no necesariamente el código más corto es el más rápido/optimizado y además entendimos que los algoritmos para ordenamiento de vectores utilizan al método de burbuja de alguna u otra manera.

# Bibliografía/Referencias

Conceptos:

<http://www.arqhys.com/general/concepto-de-informatica.html>

<https://jorgesaavedra.wordpress.com/2007/05/05/lenguajes-de-programacion/>

<http://informaticafrida.blogspot.com/2009/03/algoritmo.html>

Quicksort:

<https://alg12.wikischolars.columbia.edu/file/view/QUICKSORT.pdf/297575786/QUICKSORT.pdf>

<http://interactivepython.org/runestone/static/pythonds/SortSearch/TheQuickSort.html>

Medians:

<https://brilliant.org/wiki/median-finding-algorithm>

<http://austinrochford.com/posts/2013-10-28-median-of-medians.html>