**Universidad Nacional**

**Escuela de Informática**

**Estructuras Discretas**

**Estudiante:**

Luis Yeric Barquero Mata

Daniel Fabián García Hernández

Luis Carlos Jiménez Seas

Miguel Augusto Rodríguez Rangel

**Profesor:**

Carlos Loria-Saenz

**Fecha de Entrega:**

20 de abril



**2018**

**Índice**

Introducción 3

Quicksort 4

Median of Medians 5

Especificación del Proyecto 6

Uso de Profiling 10

Análisis Teórico Empírico 11

Conclusiones 13

Bibliografía 14

Introducción

La presente investigación se refiere a los algoritmos Quicksort y Median of Medians. El Quicksort es uno de los algoritmos de ordenamiento más rápidos y conocido actualmente, en este caso se implementará para ordenar listas y pivotes aleatorios en lenguaje Python.

El algoritmo de Median of Medians es un algoritmo DyC que se encarga de separar una lista de 5 o más objetos en partes iguales para luego sacar la mediana de cada parte y unirlas para sacar la mediana definitiva de todas las medianas. Este algoritmo necesita de otro que ordene las listas llamado “sort”, pero en este caso se implementara un algoritmo sort propio llamado “sort5” para ordenar cada lista.

En este documento se mostrará las definiciones y procedimientos utilizados para implementar estos algoritmos, de igual manera se presentarán muestras y análisis de los tiempos de cada uno y todas las herramientas utilizadas para la prueba de cada algoritmo.

Quicksort

El QuickSort o método de ordenamiento rápido, por su traducción, es un método en cual ,por medio de un pivote, que en este caso es elegido al azar por la función random.randrange(), se busca el ordenamiento de una lista por medio de comparaciones en la cuales se toman 2 los elementos de la lista, empezando con los extremos y hasta que se recorran todo los elementos, estos se comparan con el pivote y se hacen intercambios de posición dependiendo si son mayores o menores que el pivote, de manera que al final del método queden los elementos menores del lado izquierdo y los mayores al lado derecho del pivote. Luego se parte en 2 listas, en una los menores, en la otra los mayores y se realiza de manera recursiva todo el proceso anterior mencionado en estas 2 listas y así sucesivamente hasta que queden 2 listas de un elemento, estas pequeñas listas se juntan y de esta manera se cumple con el ordenamiento.

Median of Medians

Este método denominado mediana de las medianas, toma una lista y la divide de manera que en cada división queden 5 elementos, las veces que la cantidad de elementos lo permita. Estas sublistas son ordenadas por separado aplicando el método sortList que implementamos para ordenar estas listas de 5 elementos. Una vez ordenada se toma el número que se ubica en la posición 2 de cada una de estas listas, el cual es la mediana y se crea otra lista con todas las medianas encontradas y de forma recursiva se hace el método sort, en caso de haber 2 medianas se debe seguir aplicando el sortList hasta que quede una sola mediana, la cual será la mediana de las medianas.

El método sort funciona de manera que con el primer for recorre todo la lista y declara 2 variables b y j que serán usadas en un while, en el cual si se cumplen las condiciones de que j sea menor o igual a 0, esto para que no ocurra un error que haga que j sea una posición inválida en el arreglo y menor que la posición j del arreglo entra a intercambiar las posiciones de j+1 por la de j y disminuye j. Por último la posición j+1 toma el valor de b para luego seguir iterando el for hasta que el arreglo quede acomodado.

Especificación del Proyecto

Para la implementación de los problemas realizamos los siguientes módulos en los cuales creamos los métodos con los que resolvimos lo solicitado.

**Class Control:** Módulo principal, se localiza el método main.

|  |  |
| --- | --- |
| **Métodos** | **Descripción** |
| Secuencia | Llamar a los métodos para continuar con el flujo del programa. |
| CrearLista | Crea una lista vacía. |
| Control\_Menu\_Quicksort | Método el cual permite manejar el menú QuickSort. |
| Control\_Menu\_Medians | Método el cual permite manejar el menú Medians. |

**Class Interfaz:** Métodos de impresión para interactuar con el usuario.

|  |  |
| --- | --- |
| **Método** | **Descripción** |
| Menú\_Principal | Bloque de impresión con los diferentes opciones para el usuario. |
| Menú\_Quick\_Sort | Bloque de impresión con los diferentes opciones para el usuario en base al QuickSort. |
| Menu\_Medians\_of\_Median | Bloque de impresión con los diferentes opciones para el usuario en base al Median of Median. |

**Class Medians:** Implementación del método Mendian of medians, con su tiempo de corrida.

|  |  |
| --- | --- |
| **Método** | **Descripción** |
| **\_\_init\_\_** | Método constructor. |
| **set A** | Permite Modificar el atributo. |
| **Median\_of\_Medians** | Implementación del método median of medians. |
| **Timing\_median\_of\_medians** | Tiempos del método median of medians con sus diferentes operaciones de forma detallada. |
| **Timing\_Sort\_5** | Da los tiempos del método median of medians con sus diferentes operaciones de forma detallada, llama los métodos necesarios para el timing con sort5. |
| **Timing\_Medians** | Medir el tiempo de medians con diferentes casos. |
| **Exportar\_Median\_of\_Medians\_Csv** | Exportar archivo csv con diferentes casos. |
| **Sort** | Método que permite ordenar una lista. |
| **Timing Sort** | Medir tiempos de sort5. |
| **export sort5 csv** | Exportar en csv los resultados del sort5. |
| **graficar Median of Medians** | Gráfica el método median of medians. |
| **graficar Sort5** | Crear gráfica para mostrar sort5. |

**Class QuickSort:** Implementacion del metodo QuickSort con su tiempo de corrida.

|  |  |
| --- | --- |
| **Método** | **Descripción** |
| **\_\_init\_\_** | Método constructor. |
| **quick\_Sort\_Random** | Definimos caso base del quicksort, se obtienen posición del pivote. |
| **ordenar** | Ordenamos menores a la izquierda y mayores a la derecha. |
| **swap** | Método para hacer intercambios. |
| **quicksort\_timing** | Método de timing para quicksort. |
| **quickSort\_time\_save** | Guarda resultados del quicksort timing. |
| **quickSort\_rand\_timing** | Timing para quickSort Random. |
| **quickSort\_rand\_save** | Guarda los resultados del timing. |
| **graficar\_quickSort** | Método para graficar el quickSort. |
| **graficar\_quickSort\_rand** | Método para graficar quickSort rand. |
| **timing\_quickSort** | Realiza el timing para diferentes casos. |
| **timing\_quickSort\_Rand** | Realiza el timing para diferentes casos. |

**Modulo sort.py :** Implementación del método sort para las medians.

|  |  |
| --- | --- |
| **Método** | **Descripción** |
| **sort5** | Ordena una lista. |

**Modulo timing\_sort.py:** Tiempo de corrida del método sort.

|  |  |
| --- | --- |
| **Método** | **Descripción** |
| **timing\_Sort** | Tiempo de corrida del método sort. |
| **sort\_tim\_save** | Guarda los datos del método anterior. |

Uso de Profiling

Un profile es una función de python que permite mostrar cada cuanto y cuantas veces se hacen llamadas de varias partes de los programas al ser ejecutadas. Para nuestros programas utilizamos cprofile, este se muestra al solicitar al programa la opción timing. Se muestran una serie de estadísticas que dividen en columnas, las cuales se dividen de la siguiente manera: **ncalls**, donde aparecen el número de llamadas que hay en un método, **tottime** donde muestra el tiempo que dura en esas llamadas, **percall** muestra el tiempo que dura con cada una de esas llamadas, **cumtime** es el tiempo acumulado que se dura en este y otros funciones que tengan que ver con este mientras que en percall muestra cada una de esas funciones de manera primita y por último **filename:lineno(function)** donde se muestra los datos respectivos de cada función.

Análisis Teórico Empírico

A continuación presentamos el análisis del O(.) del algoritmo Medianas:

1. Tamaño de datos de interés:

n= len(a)

2) Operaciones de interés T = C = 1

3) RR Peor caso Tmedians(n)

* Tmedians(n/5)+2x5; n<=5
* 2Tmedians(n/5)+2; si n>5

a = 2

b = 5

c = 2

d = 1

4) Resolver RR Tmedians(n/5)+2xn\*\*d

Tmedians(n/5)+2xn\*\*d

a < b\*\*d = Tmedians(n)~O(n)

2Tmedians(n/5)+2\*n\*\*d

a<b\*\*d = 2Tmedians(n)~O(n)

A continuación presentamos el análisis del O(.) del algoritmo Quicksort:

1. Tamaño de datos:

a = len(a)

inicio = 0

final = len(a)-1

final\_alter = random

indice= Tordenar

ordenar = relevante

2) Operaciones de interés:

* while
* swap
* if
* return
* def ordenar

3) Establecer la ecuación de tiempo:

TQuickSort(n) = Tordenar + 2TQuickSort

Tordenar (n)~ O(n)

n= final - indice2 , donde n es el número de iteraciones del while

4) Resolver RR TQuickSort(n/2) = Tordenar(n) + 2TQuickSort(n/2)

a = 2

b = 2

d = 0

TQuickSort(n) ~ O(n log n)

En la carpeta Excel\_Curvas\_Tiempos del directorio del proyecto se adjuntan las documentos donde evaluamos el método para diferentes casos.

Conclusiones

Para concluir gracias a los resultados obtenidos con los análisis anteriormente mencionados podemos deducir que estos métodos son más efectivos que otros que cumplan funciones, por ejemplo sabemos que el método Quicksort es el de mejor tiempo conocido a la actualidad, con un tiempo O(nlog(n)), además de que comparando el método sort con bubble, el cual realizamos en clase, resulta ser mejor donde se requiere ordenar una lista de 5 elementos.

Bibliografía

Wikipedia. (11 de abril de 2018). *Quicksort*. Wikipedia. Wikipedia Recuperado de <https://en.wikipedia.org/wiki/Quicksort>

Karleigh Moore y Geoff Pilling. (31 de mayo de 2016). *Median-Finding Algorithm*. Brilliant. Brilliant Recuperado de <https://brilliant.org/wiki/median-finding-algorithm/>

Docs.python.org. (2018). *27.4. The Python Profilers — Python 3.6.5 documentation*. [online] Recuperado de: <https://docs.python.org/3/library/profile.html> [Accesado 20 Apr. 2018].