TEC | Tecnológico de Costa Rica

Quiz 4

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computadores

Algoritmos y Estructuras de Datos 1

II Semestre 2024

Estudiantes:

Yerik Jaslin Castro - 2024231163

Andy López Mo5ra - 2024130299

Profesor:

Leonardo Araya Martinez

Asistente:
Jimena Leon Huertas

Grupo:
02

Fecha:
22 de octubre

1. Problema 1

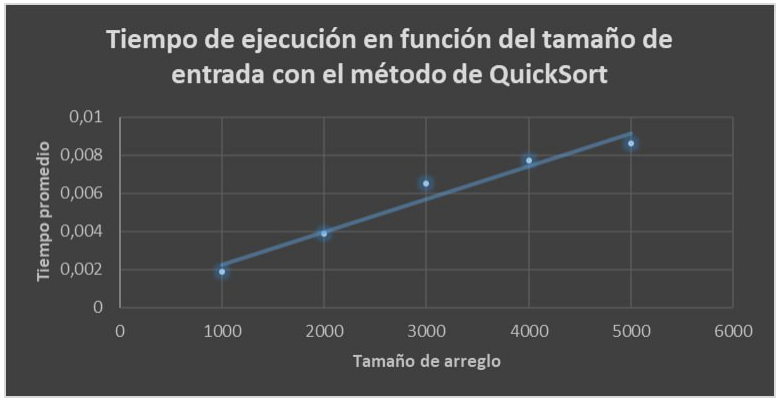
1.1. QuickSort

Codigo para ejecución de arreglos de 1000, 2000, 3000, 4000 y 5000 digitos, con sus respectivos tiempos promedios utilizando el algoritmo QuickSort:

```python
import random
import time

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    izquierda = [x for x in arr if x < pivot]
    medio = [x for x in arr if x == pivot]
    derecha = [x for x in arr if x > pivot]
    return quick_sort(izquierda) + medio + quick_sort(derecha)

def tiempo_promedio_ejecucion(array_tamaño, runs = 10):
    tiempo_total = 0
    for i in range(runs):
        arr = [random.randint(0, 10000) for i in range(array_tamaño)]
        iniciar_tiempo = time.time()
        quick_sort(arr)
        tiempo_total += time.time() - iniciar_tiempo
    return tiempo_total / runs

entradas = [1000, 2000, 3000, 4000, 5000]

tiempo_de_ejecucion = {}

for entrada in entradas:
    tiempo_de_ejecucion[entrada] = tiempo_promedio_ejecucion(entrada)

for entrada, tiempo_promedio in tiempo_de_ejecucion.items():
    print(f"Tiempo promedio para un arreglo de {entrada} elementos: {tiempo_promedio:.5f} segundos")
```

Figura 1: Codigo de QuickSort.

Grafica de tiempo de ejecución en función con el tamaño de entrada:



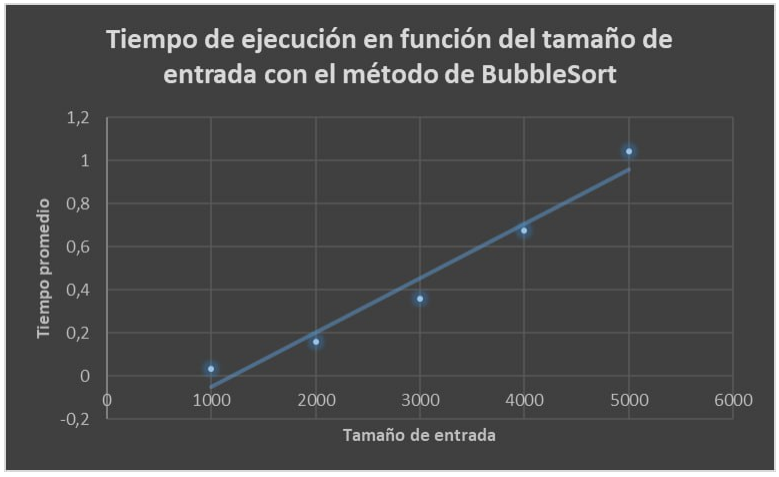
1.2. BubbleSort

Codigo para ejecución de arreglos de 1000, 2000, 3000, 4000 y 5000 digitos, con sus respectivos tiempos promedios utilizando el algoritmo BubbleSort:

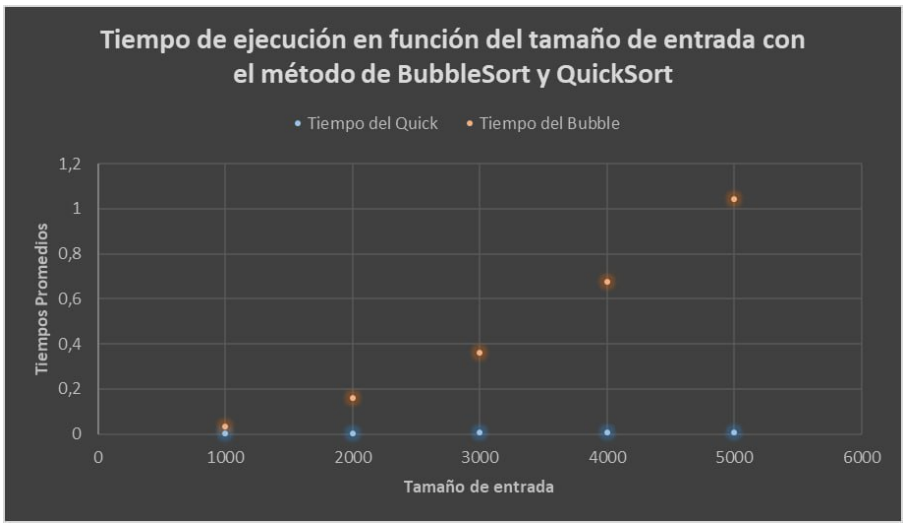
```python
import random
import time

def BubbleSortt(lista):
    a = len(lista)
    if isinstance(lista, list):
        for i in range(a):
            for j in range(0, a-i-1):
                if lista[j]>lista[j+1]:
                    lista[j], lista[j+1] = lista[j+1], lista[j]

        return lista
    return "ERROR"

def tiempo_promedio_ejecucion(array_tamaño, runs = 10):
    tiempo_total = 0
    for i in range(runs):
        arr = [random.randint(0, 10000) for i in range(array_tamaño)]
        iniciar_tiempo = time.time()
        BubbleSortt(arr)
        tiempo_total += time.time() - iniciar_tiempo
    return tiempo_total / runs

entradas = [1000, 2000, 3000, 4000, 5000]

tiempo_de_ejecucion = {}

for entrada in entradas:
    tiempo_de_ejecucion[entrada] = tiempo_promedio_ejecucion(entrada)

for entrada, tiempo_promedio in tiempo_de_ejecucion.items():
    print(f"Tiempo promedio para un arreglo de {entrada} elementos: {tiempo_promedio:.5f} segundos")
```

Figura 2: Codigo de BubbleSort.

Grafica de tiempo de ejecución en función con el tamaño de entrada:



1.3. Bubblesort vs QuickSort



1.4. Preguntas teóricas sobre los métodos

1. ¿Cuál algoritmo es más rápido y por qué?

   R/ El método de ordenamiento Bubblesort es más lento, porque este necesita recorrer la lista varias veces hasta asegurarse que los elementos en el array estén ordenados, mientras que el Quicksort divide el array de forma más estratégica y no tiene que recorrer todos los elementos.

2. ¿El tiempo de ejecución será el mismo si la implementación del algoritmo es iterativa o recursiva?

   R/ No serán los mismos, ya que la iteración es más eficiente gracias a su orden de ejecución, además la recursividad genera un stack en cada llamada por lo que lo hace muy lento en comparación a la iteración.

3. ¿Es posible que exista un algoritmo de ordenamiento que sea muy eficiente en consumo de recursos pero que a la vez sea relativamente rápido?

   R/ Depende del problema, para listas enlazadas es mejor utilizar Mergesort y es más estable que el Quicksort, el Timesort es más complicado de implementar, el Heap sort es más lento que el Quicksort. Entonces para hacerlo más general, el Quicksort es mejor en cuanto a implementación, eficiencia, consumo de recursos y rapidez.

4. Suponga que se planea ejecutar el algoritmo en un sistema computacional con extremadamente bajos recursos de memoria. ¿Cuál de los dos algoritmos de ordenamiento escogería y por qué?

R/ Escogería BubbleSort aunque sea menos práctico, ya que QuickSort al ser recursiva necesita muchas llamadas a memoria, por lo que en un sistema con bajos recursos de memoria provocaría problemas en su ejecución.

2. Aplicaciones de los algoritmos

1. ¿Cuál es la diferencia entre el algoritmo de búsqueda lineal y búsqueda por interpolación?

   R/ La búsqueda lineal busca un elemento en una lista o arreglo de forma secuencial, mientras que la búsqueda por interpolación localiza una clave en una matriz ordenada númericamente.

2. Suponga que se tiene que buscar un elemento en una lista desordenada, pero se desea optimizar el tiempo de búsqueda por sobre cualquier otra métrica ¿Cómo se podría hacer eso?

   R/ Existen diferentes maneras de buscar un elemento en una lista desordenada, una de las formas más rapidas de encontrar este número sería ordenar la lista y realizar una busqueda binaria.

3. Busque y explique alguna aplicación de la vida real donde el tiempo de búsqueda en una lista o en un arreglo sea crítico para que la aplicación se pueda dar.

   R/ Los sistemas que utilicen navegación GPS, porque dependen de búsquedas rápidas en grandes listas de datos geográficos para dar instrucciones precisas y en tiempo real. Cuando un usuario solicita una ruta o ubicación, el sistema necesita buscar rápidamente en enormes bases de datos de puntos de interés, calles, intersecciones y rutas alternativas, además estarse actualizando para estar en tiempo real y también, el sistema o aplicación debe tener buen rendimiento para los dispositivos.