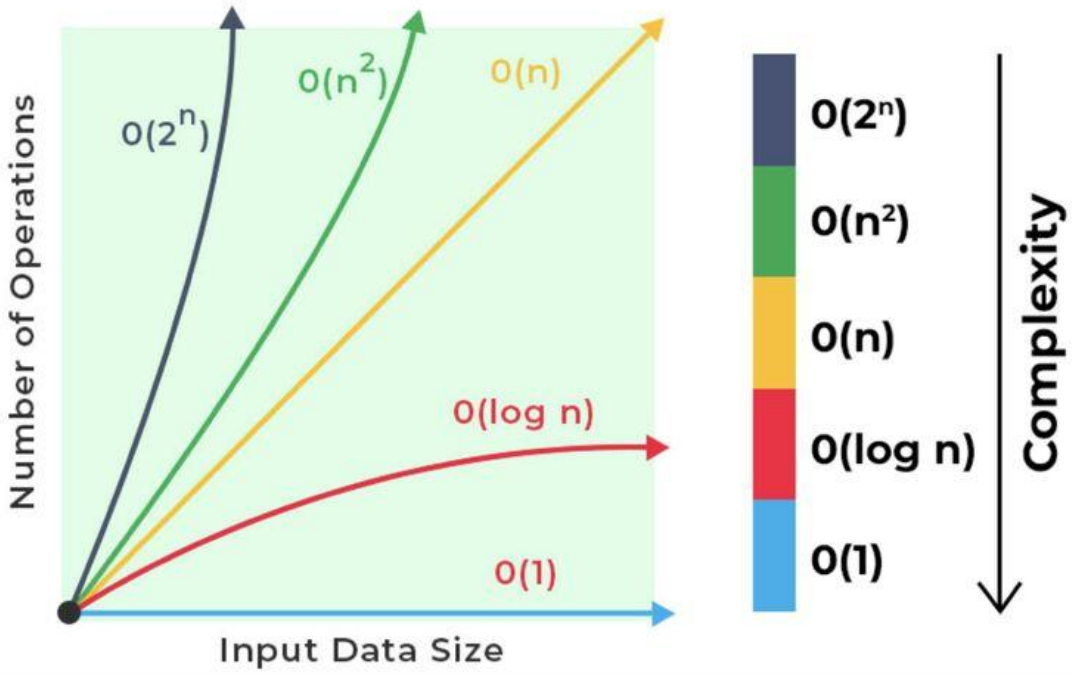
INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo
(ESCOM)

Practica 2 – Complejidad algorítmica



Análisis y diseño de Algoritmos

3CV5

Prof. Andrés García Floriano

Alumno: Solis Bustos Daniel

Introducción

El análisis de algoritmos es una parte fundamental en la ingeniería en sistemas computacionales, ya que permite evaluar el desempeño de los programas en función del **tiempo de ejecución** y el **consumo de memoria**. Estos dos factores son clave para determinar la eficiencia de una solución, especialmente cuando se trabaja con grandes volúmenes de datos.

En esta práctica se implementaron y analizaron distintos algoritmos clásicos: **búsqueda lineal**, **búsqueda binaria**, **ordenamiento por burbuja**, **Merge Sort** y el **algoritmo recursivo de Fibonacci**, utilizando los lenguajes de programación **C y Python**. Para cada algoritmo se midió el tiempo de ejecución y se estimó el consumo de memoria, mostrando los resultados mediante tablas comparativas que permiten analizar su comportamiento.

El objetivo principal es **comparar el desempeño de cada algoritmo**, identificar diferencias entre ellos y observar cómo influyen la complejidad algorítmica y el lenguaje de programación en los resultados obtenidos.

Objetivo.

- Medir el **tiempo de ejecución** de cada algoritmo.
- Analizar el **consumo de memoria** utilizado.
- Mostrar los resultados mediante **tablas comparativas**.
- Comparar el comportamiento del tiempo y la memoria.
- Identificar diferencias entre algoritmos eficientes e ineficientes.

Desarrollo.

❖ Búsqueda Lineal

La búsqueda lineal consiste en recorrer el arreglo elemento por elemento hasta encontrar el valor deseado o llegar al final de la lista. No requiere que los datos estén ordenados.

Características:

- Complejidad temporal: **O(n)**
- Complejidad espacial: **O(1)**

Análisis:

- El tiempo de ejecución aumenta de manera proporcional al tamaño del arreglo.
- El consumo de memoria se mantiene constante.
- En listas grandes resulta poco eficiente en comparación con otros métodos de búsqueda.

❖ Búsqueda Binaria

La búsqueda binaria divide repetidamente el arreglo en dos mitades, descartando la parte donde no se encuentra el elemento. Requiere que el arreglo esté ordenado.

Características:

- Complejidad temporal: **O(log n)**
- Complejidad espacial: **O(1)**

Análisis:

- El tiempo de ejecución crece muy lentamente conforme aumenta el tamaño del arreglo.
- Es considerablemente más rápida que la búsqueda lineal.
- El consumo de memoria es bajo y constante.

Ordenamiento por Burbuja

El algoritmo de burbuja compara elementos adyacentes e intercambia sus posiciones si están en desorden. Este proceso se repite hasta que el arreglo queda ordenado.

Características:

- Complejidad temporal: **$O(n^2)$**
- Complejidad espacial: **O(1)**

Análisis:

- El tiempo de ejecución aumenta de forma cuadrática.
- Para arreglos grandes, el tiempo es muy elevado.
- El consumo de memoria es bajo, pero el costo en tiempo lo hace ineficiente.

Merge Sort

Merge Sort es un algoritmo de ordenamiento basado en la técnica de **divide y vencerás**, donde el arreglo se divide en subarreglos más pequeños que luego se combinan de forma ordenada.

Características:

- Complejidad temporal: **O(n log n)**
- Complejidad espacial: **O(n)**

Análisis:

- El tiempo de ejecución es mucho menor que el de burbuja.
- El consumo de memoria es mayor debido a los arreglos auxiliares.
- Presenta un comportamiento estable incluso con grandes volúmenes de datos.

Algoritmo Recursivo de Fibonacci

Este algoritmo calcula el valor n de la sucesión de Fibonacci mediante llamadas recursivas basadas en su definición matemática.

Características:

- Complejidad temporal: **$O(2^n)$**
- Complejidad espacial: **O(n)**

Análisis:

- El tiempo de ejecución crece exponencialmente.
- Se repiten muchos cálculos innecesarios.
- El consumo de memoria aumenta debido a la profundidad de la recursión.
- Es uno de los algoritmos más ineficientes analizados.

Comparación de Resultados

Tiempo de ejecución

- **Más rápido:** Búsqueda binaria y Merge Sort.
- **Intermedio:** Búsqueda lineal.
- **Más lento:** Ordenamiento por burbuja y Fibonacci recursivo.

El comportamiento del tiempo depende directamente de la **complejidad algorítmica**, observándose crecimiento lineal, logarítmico, cuadrático y exponencial según el caso.

Consumo de memoria

- Algoritmos como búsqueda lineal, binaria y burbuja usan poca memoria.
- Merge Sort consume más memoria debido a estructuras auxiliares.
- Fibonacci recursivo incrementa el uso de memoria conforme crece la profundidad de la recursión.

Comparación tiempo vs memoria

Sí existe un **comportamiento diferente** entre ambas gráficas:

- Algunos algoritmos son rápidos pero usan más memoria (Merge Sort).
- Otros usan poca memoria pero son muy lentos (Burbuja).
- Fibonacci recursivo es ineficiente en ambos aspectos.

Conclusión.

Los resultados obtenidos demuestran que la eficiencia de un algoritmo depende tanto de su complejidad como del problema que resuelve. Algoritmos simples como la búsqueda lineal o el ordenamiento por burbuja son fáciles de implementar, pero presentan un desempeño pobre cuando el tamaño de los datos es grande.

Por otro lado, algoritmos más avanzados como la búsqueda binaria y Merge Sort ofrecen un mejor equilibrio entre tiempo y memoria, siendo más adecuados para aplicaciones reales. El algoritmo recursivo de Fibonacci, aunque útil para fines didácticos, resulta altamente ineficiente y no es recomendable para uso práctico.

Finalmente, el análisis comparativo permite comprender que no existe un algoritmo universalmente mejor, sino que la elección adecuada depende del contexto, el tamaño de los datos y los recursos disponibles.

Resultados.

1. Búsqueda lineal

Entrada:

```
BUSQUEDA LINEAL DEL 1 AL 1,000,000
1. Busqueda en lista ordenada
2. Busqueda en lista desordenada
3. Salir
Opcion:
```

Salida:

```
LISTA DESORDENADA
Elemento 1 encontrado en la posicion 487571 de la lista
Tiempo de ejecucion: 0.074544 segundos
Memoria aproximada usada: 64000000 bytes

Tabla comparativa:
Tamanio Tiempo(s)       Memoria(bytes)
1000000 0.074544        64000000

Presiona Enter para continuar...
```

2. Búsqueda binaria

Entrada:

```
BUSQUEDA BINARIA DEL 1 AL 1,000,000
1. Busqueda binaria (lista ordenada)
2. Salir
Opcion:
```

Salida:

```
LISTA ORDENADA
Elemento 1 encontrado en la posicion 0 de la lista
Tiempo de ejecucion: 0.000032 segundos
Memoria aproximada usada: 64000000 bytes

Tabla comparativa:
Tamanio Tiempo(s)       Memoria(bytes)
1000000 0.000032        64000000

Presiona Enter para continuar...
```

3. Ordenamiento por burbuja

Entrada: Proporciona un arreglo de 10 mil elementos como demostrativo.

```
ORDENAMIENTO POR BURBUJA
1. Proporcionar arreglo para ordenar
2. Usar arreglo definido (10,000 elementos)
3. Salir
Opcion:
```

Salida.

```
RESULTADOS:
Tamanio del arreglo: 10000
Tiempo de ejecucion: 7.856041 segundos
Memoria aproximada usada: 640000 bytes

Tabla comparativa:
n       Tiempo(s)       Memoria(bytes)
10000   7.856041        640000

Presiona Enter para continuar...
```

4. Merge Sort

Entrada:

```
ORDENAMIENTO POR MERGE SORT
1. Proporcionar arreglo para ordenar
2. Usar arreglo definido (10,000 elementos)
3. Salir
Opcion:
```

Salida:

```
96351 96358 96369 96390 96396 96402 96406 96416 96437 96466 96467 96473 96474 96482 96487 96490 96534 96555 96579 96590 96595 96600 9
6600 96603 96609 96616 96619 96653 96657 96659 96667 96676 96697 96698 96703 96724 96733 96735 96744 96749 96750 96759 96768 96788 96
792 96854 96855 96875 96893 96902 96903 96908 96908 96913 96926 96929 96942 96955 96957 96963 96980 96989 96999 97007 97017 97018 970
18 97025 97036 97037 97043 97073 97092 97096 97100 97107 97113 97115 97119 97121 97124 97125 97129 97146 97149 97150 97168 97170 9718
1 97182 97194 97201 97208 97210 97224 97231 97239 97245 97253 97270 97300 97305 97310 97319 97323 97327 97344 97366 97373 97373 97403
 97406 97406 97411 97417 97430 97472 97475 97493 97523 97535 97541 97549 97551 97574 97579 97582 97590 97598 97601 97619 97622 97655
97666 97677 97677 97681 97695 97714 97728 97738 97744 97747 97760 97780 97786 97794 97798 97798 97800 97812 97823 97828 97849 97849 9
7863 97867 97873 97888 97890 97891 97928 97955 97981 97981 97988 97997 97998 98001 98005 98011 98034 98037 98047 98057 98060 98069 98
077 98088 98112 98144 98149 98152 98156 98206 98224 98247 98257 98269 98275 98277 98290 98295 98312 98314 98315 98316 98327 98337 983
42 98347 98355 98362 98363 98370 98374 98391 98440 98459 98462 98468 98472 98493 98497 98511 98515 98521 98523 98531 98532 98543 9854
4 98549 98552 98560 98570 98597 98603 98606 98626 98648 98652 98657 98667 98670 98679 98680 98688 98695 98704 98717 98728 98732 98741
 98744 98795 98798 98803 98812 98822 98835 98836 98842 98848 98857 98882 98882 98891 98895 98933 98940 98953 98970 98975 98976 98991
98994 99002 99011 99053 99058 99066 99066 99072 99077 99080 99086 99116 99117 99139 99143 99167 99185 99191 99200 99214 99222 99225 9
9232 99250 99254 99266 99284 99315 99315 99347 99351 99354 99378 99413 99445 99455 99460 99461 99464 99470 99471 99479 99484 99492 99
501 99537 99540 99551 99560 99562 99569 99578 99580 99603 99603 99604 99610 99627 99663 99671 99682 99684 99687 99705 99714 99729 997
31 99746 99750 99772 99803 99832 99841 99848 99855 99859 99874 99887 99913 99918 99919 99922 99930 99942 99951 99970

RESULTADOS:
Tamanio del arreglo: 10000
Tiempo de ejecucion: 0.031696 segundos
Memoria aproximada usada: 640000 bytes

Tabla comparativa:
n       Tiempo(s)       Memoria(bytes)
10000   0.031696        640000

Presiona Enter para continuar...
```

5. Algoritmo recursivo de Fibonacci

```
C:\Users\solis\OneDrive\Escritorio\ESCOM\3er Semestre\ADA\P2_Complejidad>py fibonacci.py
========================================
 ALGORITMO RECURSIVO DE FIBONACCI
========================================
Este algoritmo calcula el n-esimo numero
de la sucesion de Fibonacci usando
recursividad.

Regla de Fibonacci:
F(0) = 0
F(1) = 1
F(n) = F(n-1) + F(n-2)

Nota: Este metodo es ineficiente para
valores grandes de n debido a la
repeticion de calculos.
========================================

Ingrese el valor de n: 6

RESULTADO:
Fibonacci(6) = 8

ESTADISTICAS:
Tiempo de ejecucion: 0.000012 segundos
Memoria aproximada usada: 168 bytes

Tabla comparativa:
n       Tiempo(s)       Memoria(bytes)
6       0.000012        168
```