**TECNICATURA UNIVERSITARIA UTN 2025**

**EN PROGRAMACIÓN**

**A DISTANCIA**

**ALGORITMOS DE BUSQUEDA Y ORDENAMIENTO**

**Alumnos:**

Santiago Pace – pacesantiago@gmail.com

Nicolas Viruel – nicolasviruel@gmail.com

**Materia:** Programación I

**Profesor:** Ing. Laura Fernández

**Fecha de entrega:** 9 de mayo de 2025

**Índice:**

1. Introducción

2. Marco Teórico

3. Caso Práctico

4. Metodología Utilizada

5. Resultados Obtenidos

6. Conclusiones

7. Bibliografía 8. Anexos

**Introducción**

En este trabajo integrador abordamos el tema de **algoritmos de búsqueda**, fundamentales en la programación y ciencias de la computación. Los algoritmos permiten encontrar información dentro de una estructura de datos, como listas o arreglos. Elegimos enfocarnos en dos tipos: la **búsqueda lineal**, simple y directa, y la **búsqueda binaria**, más eficiente pero que requiere una lista ordenada. Además de implementarlos en Python, analizamos sus diferencias en cuanto a rendimiento y aplicación práctica.

**Marco Teórico**

Un **algoritmo de búsqueda** tiene como objetivo localizar un elemento específico dentro de una estructura de datos.

* **Búsqueda Lineal**: recorre la lista elemento por elemento hasta encontrar el objetivo o llegar al final. Su complejidad es **O(n)**, donde n es el tamaño de la lista.
* **Búsqueda Binaria**: requiere que la lista esté ordenada. Divide el arreglo por la mitad y decide en qué sublista continuar la búsqueda. Su complejidad es **O(log n)**, mucho más eficiente para listas grandes.

Ambos algoritmos son fundamentales para entender cómo se accede a los datos, y su elección depende del tamaño y características de la lista.

**Caso Practico**

Para poner en práctica ambos algoritmos, realizamos un programa en Python donde el usuario puede ingresar un número a buscar en una lista ordenada y elegir entre búsqueda lineal o binaria. El programa devuelve la posición del número (si lo encuentra) y mide el tiempo de ejecución para comparar el rendimiento.

import time

# --------------------------

# Algoritmo de búsqueda lineal

# --------------------------

def busqueda\_lineal(lista, objetivo):

for i, elemento in enumerate(lista):

if elemento == objetivo:

return i

return -1

# --------------------------

# Algoritmo de búsqueda binaria

# --------------------------

def busqueda\_binaria(lista, objetivo):

inicio = 0

fin = len(lista) - 1

while inicio <= fin:

medio = (inicio + fin) // 2

if lista[medio] == objetivo:

return medio

elif lista[medio] < objetivo:

inicio = medio + 1

else:

fin = medio - 1

return -1

# --------------------------

# Menú para que el usuario elija

# --------------------------

def menu():

lista = sorted([3, 7, 10, 15, 20, 25, 30, 35, 40]) # lista ordenada

objetivo = int(input("¿Qué número querés buscar?: "))

print("\nElegí el tipo de búsqueda:")

print("1 - Búsqueda Lineal")

print("2 - Búsqueda Binaria")

opcion = input("Opción: ")

if opcion == '1':

inicio = time.time()

resultado = busqueda\_lineal(lista, objetivo)

fin = time.time()

print("Resultado:", resultado)

print("Tiempo de ejecución (lineal):", fin - inicio, "segundos")

elif opcion == '2':

inicio = time.time()

resultado = busqueda\_binaria(lista, objetivo)

fin = time.time()

print("Resultado:", resultado)

print("Tiempo de ejecución (binaria):", fin - inicio, "segundos")

else:

print("Opción inválida.")

menu()

**Metodología Utilizada**

La metodología consistió en:

* Implementar ambos algoritmos en Python.
* Utilizar la función time() del módulo time para medir el tiempo de ejecución.
* Probar la búsqueda con distintos valores y observar los tiempos.
* Comparar los resultados obtenidos al aplicar los algoritmos sobre la misma lista ordenada.

**Resultados obtenidos**

Al ejecutar el programa con diferentes números, observamos que:

* La **búsqueda binaria** fue más rápida que la lineal en todos los casos, especialmente cuando el número estaba al final o no estaba en la lista.
* La **búsqueda lineal** es más simple pero menos eficiente, ya que debe recorrer todos los elementos en el peor de los casos.
* En listas cortas, la diferencia de tiempo es mínima, pero en listas grandes la diferencia se vuelve significativa.

**Conclusiones**

A través de este trabajo pudimos entender cómo funcionan y se aplican distintos algoritmos de búsqueda. Aprendimos que:

* La elección del algoritmo depende del tamaño de los datos y si están ordenados.
* La búsqueda binaria es más eficiente, pero necesita una lista ordenada.
* La búsqueda lineal, aunque más lenta, puede usarse en listas desordenadas sin necesidad de preprocesamiento.

Este análisis nos permitió aplicar conceptos clave como eficiencia algorítmica y estructuras de datos.

**Bibliografía**

* Documentación oficial de Python: <https://docs.python.org/3/>
* Enlaces de Guía: https://4geeks.com/es/lesson/algoritmos-de-ordenamiento-y-busqueda-en-python

**Anexos**

Repositorio de Github: <https://github.com/NicolasViruel/Integrador-Programacion-Busqueda-y-Ordenamiento>

Link de video Youtube: