Departamento de Computación y Sistemas Inteligentes
Computación y Estructuras Discretas II
Tarea Integradora 1 2025-2

Objetivos Terminales
RA1. Aplicar el paradigma funcional en el análisis, diseño, evaluación, selección e implementación de algoritmos para dar solución a problemas cuya estructura es naturalmente autocontenida.
RA3. Aplicar conceptos fundamentales de la teoría de números en el análisis de problemas computacionales que lo requieran, y analizar relaciones binarias, en particular, aquellas que forman una relación de equivalencia o relación de orden y aplicar este concepto a la solución de problemas.
RA4. Expresar o comunicar con el vocabulario y lenguaje adecuado/especializado las ideas principales sobre estructuras discretas o la programación funcional.

Objetivo

El objetivo de este proyecto es **diseñar e implementar un Árbol B inmutable** (B-tree) en Scala aplicando los principios de la **programación funcional**.
Los estudiantes deberán:

1. Implementar la estructura de datos **Árbol B** desde cero.
2. Crear **dos instancias** del árbol a partir de un mismo conjunto de datos (dataset):
    - Una instancia usando una **clave numérica** del dataset.
    - Otra instancia usando una **columna tipo texto** del dataset, para la cual deberán **generar una clave numérica** aplicando un **algoritmo de generación de claves** basado en teoría de números o criptografía ligera.
3. Evaluar experimentalmente los tiempos de ejecución de las operaciones de **inserción** y **búsqueda**, y compararlos con la **complejidad teórica**.

B-Tree

Un B-tree es una estructura de datos que almacena información **ordenada** y **balanceada**, que permite realizar operaciones de búsqueda, inserción y eliminación en tiempo logarítmico $T(n) = O(log\ n)$.
A diferencia de un árbol binario, un árbol B puede contener varios valores (claves) y varios subárboles. Esto último, junto con el balanceo, causa que la altura del árbol crezca lentamente.

El **grado mínimo** o cantidad de subárboles se limita por un parámetro (que aplica para la raíz y todos los subárboles) denotado por la letra t, $t \geq 2$. Existen diferentes definiciones de un B-Tree, en esta tarea debe considerarse la definición de B-Tree del texto de Cormen - Sección 18.1, que ha sido adaptada para el paradigma funcional de este curso a continuación.

Un B-Tree es un árbol que tiene n **claves y** $n + 1$ **subárboles**. Además, un B-Tree puede ser de uno de los siguientes subtipos: raíz, interior, hoja

- **Las claves:** las claves se guardan en una estructura ordenada de tamaño n, las claves están ordenadas de forma ascendente. Las claves se denotan por key_i , $i \leq n$ y siempre se cumple que en la estructura $key_1 \leq key_2 \leq key_3 \ldots \leq key_n$.
- **Subárboles:** los subárboles también se guardan en una estructura ordenada, cada subárbol se identifica por c_j, $j \leq n + 1$.
- **Orden.** Las claves de cada nodo separan los rangos de valores de sus subárboles. Esto significa que si hay n claves así: $[key_1, key_2, key_3 \ldots, key_n]$ hay $n + 1$ subárboles $[c_1, c_2, c_3, \ldots, c_n, c_{n+1}]$ y se cumple que si quiero insertar una nueva $newKey$ debería considerar el siguiente orden:
  - Si $newKey < key_1$, $newKey$ deberá insertarse en el subárbol c_1
  - Si $key_1 < newKey < key_2$, $newKey$ deberá insertarse en el subárbol c_2
  - Si $key_2 < newKey < key_3$, $newKey$ deberá insertarse en el subárbol c_3
  - Si $key_{n-1} < newKey < key_n$, $newKey$ deberá insertarse en el subárbol c_n
  - Si $key_1 < newKey$, $newKey$ deberá insertarse en el subárbol c_{n+1}
- **Profundidad.** Todas las hojas del árbol están al mismo nivel, es decir, todas tienen la misma profundidad o altura h. Esto garantiza que el árbol esté siempre balanceado.
- **Grado mínimo.** La cantidad de claves, y en consecuencia de hijos, de cada árbol y subárbol está limitada por un parámetro llamado **grado mínimo** y denotado por t, $t \geq 2$. Este grado mínimo es el mismo en todo el árbol.
  - Cada nodo (excepto la raíz) tiene **al menos `t - 1` claves**, por tanto, tiene **al menos `t` hijos** si es un nodo interno.
  - Cada nodo puede tener **como máximo `2t - 1` claves**, y en consecuencia **hasta `2t` hijos**.
  - La **raíz** tiene al menos **una clave** (a menos que el árbol esté vacío).

  La siguiente tabla resume estas definiciones:

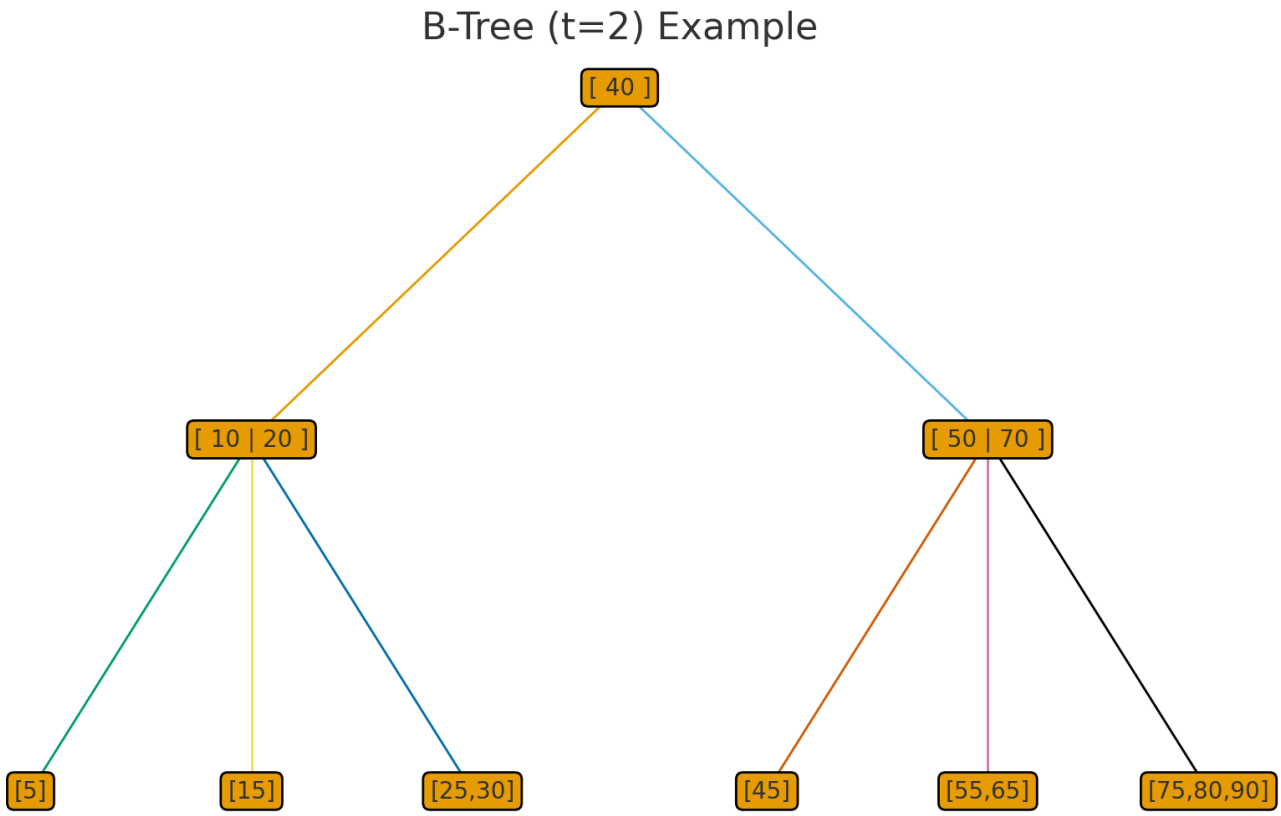
Subtipo	Mínimo de claves	Máximo de claves	Mínimo de hijos	Máximo de hijos
Hoja (no raíz)	t − 1	2t − 1	—	—
Interno (no raíz)	t − 1	2t − 1	t	2t
Raíz	1 *(si el árbol no está vacío)*	2t − 1	0 si es hoja / 2 si es interno	2t

Ejemplo de un B-tree con t =2

A continuación se muestra un B-tree con t= 2, los siguientes son sus parámetros:

- Máximo de claves por nodo: 2t−1=3
- Mínimo de claves (no raíz): t−1=1

- Máximo de hijos por nodo interno: 2t=4
- Mínimo de hijos por nodo interno (no raíz): t=2
- Raíz: al menos 1 clave si el árbol no está vacío; si es interna debe tener ≥2 hijos.



De acuerdo al ejemplo:

- La **raíz** tiene 1 clave → permitido, porque la raíz puede tener de **1 a 3 claves**
- Los **nodos internos** tienen **2 claves** (dentro del rango mínimo 1 — máximo 3)
- Cada nodo interno tiene **3 hijos** (entre el mínimo 2 y máximo 4)
- Las **hojas** están todas al mismo nivel
- Las claves dentro de cada nodo están **ordenadas** de menor a mayor
- Las claves **separan los rangos** correspondientes a sus hijos

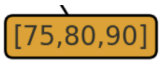
Búsqueda en un B-tree

La operación de búsqueda en un B-Tree sigue un proceso muy similar al de un árbol binario de búsqueda, pero en lugar de decidir entre dos caminos (izquierda o derecha), se toman decisiones entre múltiples subárboles en función del número de claves que tenga cada nodo.

La búsqueda de una clave $keyB$ es **recursiva**, y según la estructura del árbol se distinguen dos casos:

- En un árbol hoja, se busca linealmente dentro de las claves del nodo,

a. Si se encuentra la clave, se retorna el valor encontrado
b. Si no se encuentra la clave, se retorna un valor para indicar que la clave no está en el árbol(por ejemplo -1)



Ejemplo. Si buscamos la clave 65 en la hoja de la imagen, se recorre la lista de claves y se retorna -1. ¿Qué pasaría si se busca la clave 80?

- En un árbol que no es hoja se busca considerando la propiedad de orden entre las claves y los subárboles. Recuerde que el árbol tiene:

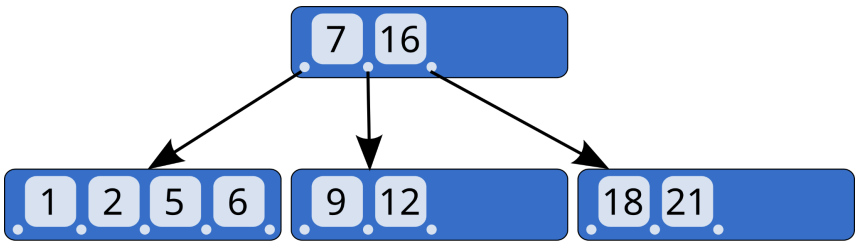
→ Una lista ordenada de claves así: $[key_1, key_2, key_3 \dots, key_n]$

→ $n + 1$ subárboles $[c_1, c_2, c_3 \dots, c_n, c_{n+1}]$

→ y que $keyB$ es la clave que se está buscando.

Pasos:

1. Se encuentra la primera clave en la que se cumple que $keyB \leq key_i$ y se identifica la posición i de key_i en la lista de claves
2. Si existe esta primera pareja $keyB \leq key_i$
   a. En el caso en que $keyB = key_i$ → entonces se encontró la clave buscada
   b. De lo contrario, se continúa buscando por medio de un llamado recursivo con el subárbol c_i
3. Si no hay una clave en la que se cumpla la condición $keyB \leq key_i$, se concluye que $keyB$ es mayor que todas las claves del árbol, en ese caso se continúa buscando por medio de un llamado recursivo con el subárbol c_{n+1}

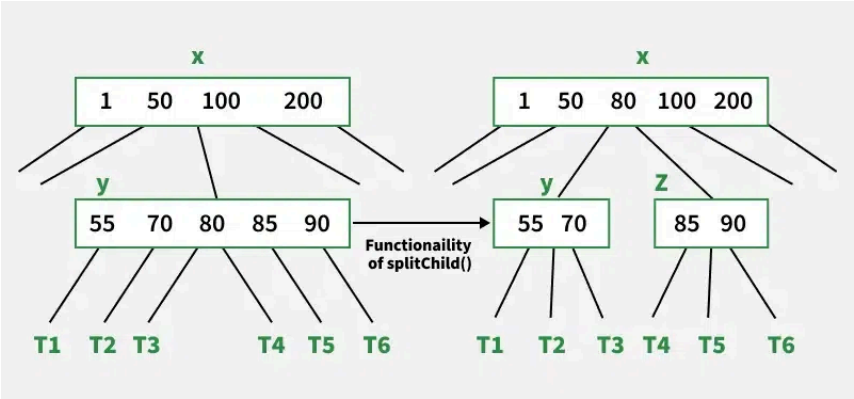


Ejemplo.
Si buscamos la clave key=2, en el árbol de la imagen, nos damos cuenta de que key < 7 y por tanto hay que buscar en el primer subárbol.
Si buscamos la clave key= 10, en este caso se busca en el segundo subárbol. Finalmente, si buscamos la clave 25, se busca en el tercer subárbol.

Insertando en un B-tree

Insertar una clave en un B-tree es considerablemente más complejo que hacerlo en un árbol binario de búsqueda. Al igual que en los árboles binarios, primero buscamos la posición de la hoja donde correspondería insertar la nueva clave. Sin embargo, en un B-tree no podemos simplemente crear una nueva hoja y colocar en ella la clave, porque el árbol resultante podría violar las propiedades del B-tree. En su lugar, insertamos la nueva clave en una hoja existente.

Como no es posible insertar en una hoja que está llena, introducimos la operación de **dividir (split)** un nodo lleno. Sea y un nodo que contiene 2t−1 claves; dividimos y alrededor de su clave mediana en dos nodos que contienen t−1 claves cada uno. La clave mediana se mueve hacia arriba y queda en el padre de y, donde actúa como separador entre los dos nuevos hijos, como se muestra en la siguiente imagen:



Tenemos inicialmente un árbol **x**, que contiene un subárbol **y**, que está lleno y debe dividirse.

Entonces se toma la clave de la mitad (80) y se parte el árbol en dos: (1) el subárbol con las claves menores que 80, que en la imagen se llama y (2) el subárbol con las claves mayores que 80 que en la imagen se llama z.

Luego se insertan 80 en las claves y z en los hijos del árbol **x.**

Si el padre de y también estuviera lleno, habría que dividirlo antes de poder promover la mediana; por tanto, la división puede propagarse hacia arriba y, en el peor caso, llegar hasta la raíz.

Para evitar tener que retroceder y dividir nodos después de haber encontrado la hoja de inserción, el algoritmo realiza una única pasada desde la raíz hacia la hoja: **mientras descendemos, dividimos preventivamente cualquier nodo lleno que encontremos (incluida la hoja si está llena)**. De este modo, cuando lleguemos a dividir un nodo y, su padre ya no estará lleno y la promoción de la clave mediana podrá realizarse sin problemas.

Tenga en cuenta que:

- Antes de descender a un hijo, se debe comprobar si el hijo está lleno.
- Si lo está, aplicar split sobre ese hijo, De ese modo, se garantiza que cuando se necesite insertar en un subárbol, este no esté lleno.
- Si la raíz está llena al inicio, crear una nueva raíz vacía y dividir la antigua raíz para aumentar la altura del árbol.

Actividades de la TI

1. TAD B-tree

Diseñe e implemente el TAD B-tree, siguiendo lo aprendido en la clase (especificación, diseño Orientado a Objetos, estructura inmutable). Además, su estructura debe cumplir con las propiedades explicadas en este enunciado, que están basadas en la definición de un B-tree de acuerdo al texto de Cormen [1]. **Recuerde:** existen otras definiciones de esta estructura, pero usted deberá implementar la que está explicada en este enunciado.

2. Operaciones del B-tree

2.1. Búsqueda recursiva

La operación de búsqueda (search) deberá implementarse de forma recursiva y puramente funcional, siguiendo la estructura del árbol. Deben contemplarse los siguientes casos:

- Árbol vacío.
- Árbol con un único nodo (raíz-hoja).
- Árbol con varios niveles (nodos internos y hojas).

2.2. Inserción funcional e inmutable

La operación de inserción (insert) debe implementarse siguiendo la estrategia **split-on-the-way-down**, que consiste en dividir los nodos llenos antes de descender hacia ellos.
Las condiciones generales son las siguientes:

- Si la raíz está llena al inicio de la inserción, crear una nueva raíz vacía y dividir la antigua raíz, incrementando la altura del árbol.
- Antes de descender a un hijo, verificar si el nodo hijo está lleno; si lo está, dividirlo antes de continuar la búsqueda.
- Insertar la nueva clave en la hoja correspondiente una vez garantizado que no se encuentra llena.

Cada operación deberá mantener la **inmutabilidad** de la estructura, es decir, crear una nueva versión del árbol sin modificar las instancias previas.

3. Llenado del Árbol y Generación de Claves

Los **B-trees** se utilizan como estructuras de **índices en bases de datos** para permitir búsquedas, inserciones y eliminaciones eficientes en disco. Al mantener los datos **ordenados y balanceados**, se reduce el número de accesos a disco necesarios para localizar un registro. Vamos a utilizar la estructura implementada para indexar información, para ello, utilizaremos un dataset para crear dos instancias del árbol (dos índices).
Para llenar los árboles, vamos a utilizar el conjunto de datos **Netflix Movies and TV Shows** disponible en Kaggle:
https://www.kaggle.com/datasets/shivamb/netflix-shows

A partir de este dataset deberán crearse **dos instancias del B-tree**, diferenciadas por el tipo de clave utilizada: una clave numérica y una clave de cadena. Implementar unos algoritmos para crear las claves, esto se debe resolver aplicando la teoría de números como se explica a continuación:

3.1 Claves numéricas

Para la indexación usando una clave numérica, se debe usar un campo numérico del dataset (a elección del grupo) y deberá implementarse una función de hashing que garantice unicidad (también a elección del grupo y aplicando teoría de números).
Esta función de hashing deberá devolver un número entero que servirá como clave primaria para insertar en el árbol cada registro del dataset.

3.1 Claves textuales

Para la indexación usando una clave textual, se utilizará una columna de tipo cadena (por ejemplo, title) y se transformará cada cadena en una clave numérica mediante un algoritmo de cifrado, o de checksum usando aritmética modular.

Ambas estrategias de generación de claves deberán ser documentadas y comparadas en términos de su distribución y comportamiento durante las operaciones de inserción y búsqueda.

4.1 Análisis experimental

Para cada una de las dos instancias del B-tree (numérica y textual), deberán registrarse los tiempos promedio de:

- Inserción de todos los datos del dataset.
- Búsqueda de un conjunto representativo de claves (por ejemplo, el 50% del total).
- Los resultados deberán representarse gráficamente, mostrando la relación entre el tamaño del árbol, el número de operaciones y el tiempo de ejecución.
- Finalmente, se deberá realizar un análisis comparativo entre los resultados empíricos y la complejidad teórica del B-tree, discutiendo las posibles diferencias y su relación con:
    - El grado mínimo t elegido.
    - La naturaleza de las claves (numéricas o textuales).
    - El impacto de la inmutabilidad en el desempeño.

Documente los resultados en un informe o presentación que muestre el análisis de los datos. Este informe debe hacer parte de la entrega final, utilice las herramientas de los cursos de matemáticas aplicadas para incluir las gráficas en los documentos del repositorio.

Actividades de Análisis, Diseño e Implementación

El resultado de la etapa de análisis y diseño deberá incluirse en una carpeta llamada **`doc/`**, que debe contener:

- El **formato de diseño de pruebas** (en Markdown).
- El **diagrama de clases UML** (archivo con extensión `.jpeg`).

Usando una herramienta que permita crear diagramas UML, elabore el **diagrama de clases del proyecto**, incluyendo las clases principales y sus relaciones.

Las pruebas del proyecto deben realizarse en **dos etapas**:

1. **Pruebas unitarias**, que validan los métodos y funciones auxiliares durante la implementación.
2. **Pruebas de ejecución**, usando el **dataset de Netflix** como fuente de datos para poblar y verificar el funcionamiento del árbol.
3. Cada grupo deberá:

- Diseñar e implementar los **casos de prueba necesarios** para cada método y/o función auxiliar.
- Incluir en el diseño las clases que permiten **cargar, generar y manipular los datos** provenientes del dataset.

Una vez completada esta parte, cree en su repositorio el commit:
“Resultados de Análisis y Diseño”
y continúe con la implementación de los métodos. Pueden realizarse commits previos.

El proyecto **DEBE** desarrollarse siguiendo la metodología **TDD (Test-Driven Development)** y gestionarse mediante un **repositorio desde el día 1**.

Durante el desarrollo deben realizarse **al menos 10 commits distribuidos equitativamente en el tiempo**.

En cada commit se debe reportar en el archivo `README` un **indicador de avance**, calculado como:

$$\text{Indicador de avance} = \frac{\text{número de funcionalidades realizadas}}{\text{total de funcionalidades planificadas}}$$

Este valor debe reflejar el progreso del desarrollo del proyecto a lo largo del tiempo.

Instrucciones para la entrega

1. Los entregables de la tarea deben enviarse por medio de GitHub classroom, para ello, los estudiantes deben ingresar a GitHub classroom usando la invitación y asignando un identificador de su equipo. Los equipos pueden mezclar estudiantes del grupo de la mañana y de la tarde, y deben estar registrados en una hoja de equipos y en GitHub classroom. Los equipos serán de tres o cuatro integrantes, los casos particulares deberán discutirse y aprobarse por los profesores. Para registrar su equipo usted deberá:
    a. Llenar la siguiente hoja de cálculo con los nombres y los usuarios de gitHub de cada integrante y el identificador del equipo (Ex, en donde x es un consecutivo). La hoja de registro es la siguiente [link a la hoja de registro].
    b. Ingresar a GitHub classroom usando la invitación y el identificador de su equipo. Tenga en cuenta que el **primer integrante en registrarse** debe crear el equipo usando el identificador de la hoja de registro [Invitación].

2. La tarea deberá desarrollarse usando la configuración de proyecto (no worksheets) y combinando la Programación orientada a objetos con la programación funcional.
3. Para llevar a cabo la etapa de experimentación deberá implementar las clases que generen, lean y carguen datos.

Presenting Your Work

Each team must prepare a **10-minute presentation (in English)** summarizing their project. The presentation should describe the **problem**, the **methodology and functional implementation**, the **results obtained using the Netflix dataset**, and the **main insights** derived from the experiments.

The presentation should include:

- A clear explanation of how the **B-tree structure** was modeled and implemented.
- Demonstrations or examples showing **key operations** such as insertion and search.
- Visual support through **diagrams, code fragments, and results**.
- Reflection on **the advantages and challenges** of implementing a functional, immutable B-tree.

Recommendations for the Presentation

To ensure a clear and professional presentation:

1. Structure and timing

- Allocate time evenly: 2 min for context and problem definition, 4 min for methodology and implementation, 2 min for results, and 2 min for conclusions.
- Rehearse the presentation to make sure it fits the 10-minute limit.

2. Visual design

- Use diagrams, figures, or flowcharts to represent the structure and operations of your B-tree.
- Keep slides concise: no more than 6–8 lines of text per slide.
- Highlight code snippets selectively — show only the parts that support your explanation.
- Include key mathematical notation (e.g., t, $2t-1$) where relevant to emphasize B-tree properties.

3. Communication

- Speak clearly and confidently; avoid reading directly from slides.
- Make eye contact with the audience and introduce yourself at the beginning.Use technical vocabulary accurately and in English (e.g., “immutable structure,” “functional insertion,” “search complexity”).
- Conclude by summarizing the main insights and potential extensions of your implementation.

4. Evidence of collaboration

- Each team member should briefly explain their specific contribution to the project.
- Coordinate transitions between speakers smoothly to maintain the flow.

Criterion	Level 0 (0 points)	Level 1 (5 points)	Level 2 (10 points)	Level 3 (15 points)
Implementation Presentation	Does not show their part of the implementation	Shows implementation with unclear or incomplete explanation	Explains their implementation clearly but without concrete examples	Explains their implementation in detail using: • Practical examples • Illustrative diagrams • References to the code
Oral Expression	Does not perform the presentation	Reads directly from slides, does not mention their name, and/or shows limited mastery	Clear presentation with partial mastery; mentions their name but lacks fluency	Professional presentation including: • Verbal fluency • Demonstrated mastery • Proper self-introduction

				• Consistent eye contact
Contribution to the Project	No evidence of participation	Describes generic contributions without concrete link to the code	Explains participation in 1–2 components with partial examples	Demonstrates contribution through: • Multiple pieces of evidence in code • Process explanation • Measurable impact on the project
Presentation Time Management	Does not attend the presentation	Exceeds or falls short of the time limit by more than 3 minutes	Stays within time but with uneven section balance	Ideal time management with: • Appropriate duration • Balanced section distribution • Consistent pacing
Slide Quality	Does not use visual aids	Slides contain long texts, lack mathematical notation, and show poor organization	Mostly visual content but with 1–2 issues in notation or structure	Professional slides with: • Correct mathematical notation • Key points listed • Relevant diagrams • Concise text on slides

Referencias

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Third Edition. The MIT Press, 3rd edition, 2009.