Universidad Nacional de Ingeniería (UNI)
Escuela Profesional de
Ciencia de la Computación
Sílabo 2026-I

1. CURSO

CS212. Análisis y Diseño de Algoritmos (Obligatorio)

2. INFORMACIÓN GENERAL

2.1 Curso	:	CS212. Análisis y Diseño de Algoritmos
2.2 Semestre	:	5^{to} Semestre.
2.3 Créditos	:	4
2.4 Horas	:	2 HT; 4 HP;
2.5 Duración del periodo	:	16 semanas
2.6 Condición	:	Obligatorio
2.7 Modalidad de aprendizaje	:	Presencial
2.8 Prerrequisitos	:	• CS210. Algoritmos y Estructuras de Datos. (4^{to} Sem) • CS211. Teoría de la Computación. (4^{to} Sem)

3. PROFESORES

Atención previa coordinación con el profesor

4. INTRODUCCIÓN AL CURSO

Los algoritmos son el corazón de la ciencia de la computación. Este curso proporciona las herramientas matemáticas y analíticas necesarias para evaluar la eficiencia de los algoritmos en términos de tiempo y espacio. Los estudiantes explorarán estructuras de datos fundamentales, análisis de complejidad y varios paradigmas de diseño, como fuerza bruta, algoritmos voraces y programación dinámica. Comprender estas técnicas es esencial para construir software que no solo sea funcional, sino también escalable y eficiente en el uso de recursos.

5. OBJETIVOS

- Dominar el uso de la notación asintótica para analizar el rendimiento de los algoritmos.
- Aplicar estructuras de datos fundamentales para resolver problemas computacionales.
- Usar estrategias algorítmicas como Voraz, Divide y Vencerás y Programación Dinámica.
- Analizar la complejidad de algoritmos en el mejor, promedio y peor de los casos.
- Desarrollar la capacidad de seleccionar el algoritmo más apropiado para un problema dado.

6. RESULTADOS DEL ESTUDIANTE

AG-C08) Análisis de Problemas: Identifica, formula y analiza problemas complejos de computación. (Usage)

AG-C12) Usa enfoques sistémicos para seleccionar, desarrollar, aplica, integrar y administrar tecnologías seguras de computación para lograr los objetivos del usuario. (Usage)

7. TEMAS

Unidad 1: Estructuras de Datos Fundamentales (6 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Tipo de Dato Abstracto (ADT) y operaciones sobre un ADT: – Operaciones de diccionario (insertar, eliminar, encontrar) • Arreglos: – Numéricos vs no numéricos, cadenas de caracteres – Unidimensionales (vector) vs multidimensionales (matriz) • Registros/Estructuras/Tuplas y Objetos • Listas enlazadas (por razones históricas): – Simples vs Dobles y Lineales vs Circulares • Pilas • Colas y deques: – Cola de prioridad basada en montículo • Tablas/mapas hash: – Resolución de colisiones y complejidad (por ejemplo, sondeo, encadenamiento, rehash) • árboles: – Binarios, n-arios y árboles de búsqueda – Balanceados (por ejemplo, AVL, Rojo-Negro, Montículo) • Conjuntos	• Para cada ADT/Estructura de Datos en esta unidad: – Explicar su definición, propiedades, representación(es) y operaciones de ADT asociadas. – Explicar paso a paso cómo las operaciones de ADT asociadas con la estructura de datos la transforman. [Explicar] • Explicar cómo se maneja la evitación de colisiones y la resolución de colisiones en tablas hash [Explicar] • Explicar la propiedad de montículo y el uso de montículos como una implementación de una cola de prioridad [Explicar]
Lecturas : [Cor+22], [KT05]	

Unidad 2: Algoritmos Fundamentales (6 horas)

Resultados esperados: AG-C08,AG-C12

Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Grafos (por ejemplo, [no]dirigidos, [a]cíclicos, [no]conexos y [no]ponderados): – Representación de grafos: lista de adyacencia vs matriz • Algoritmos de búsqueda: – Complejidad $O(n)$ (por ejemplo, búsqueda lineal/secuencial en arreglo/lista) – Complejidad $O(\log_2 n)$ (por ejemplo, búsqueda binaria) – Complejidad $O(\log_b n)$ (por ejemplo, búsqueda en árbol no informada en profundidad/amplitud) • Algoritmos de ordenamiento (por ejemplo, estables, inestables): – Complejidad $O(n^2)$ (por ejemplo, inserción, selección) – Complejidad $O(n \log n)$ (por ejemplo, quicksort, merge, timesort) • Algoritmos de grafos: – Camino más corto (por ejemplo, Dijkstra, Floyd) – árbol de expansión mínima (por ejemplo, Prim, Kruskal) • Algoritmos de ordenamiento: – Complejidad $O(n \log n)$ heapsort – Pseudo $O(n)$ complejidad (por ejemplo, bucket, counting, radix) • Algoritmos de grafos: – Clausura transitiva (por ejemplo, Warshall) – Ordenamiento topológico • Emparejamiento: – Emparejamiento eficiente de cadenas (por ejemplo, Boyer-Moore, Knuth-Morris-Pratt) – Emparejamiento de subsecuencia común más larga – Emparejamiento de expresiones regulares	• Para cada algoritmo en esta unidad explicar paso a paso cómo opera el algoritmo [Explicar] • Para cada enfoque algorítmico (por ejemplo, ordenamiento) en esta unidad aplicar un ejemplo prototípico del enfoque (por ejemplo, ordenamiento por mezcla) [Aplicar] • Dados los requisitos para un problema, desarrollar múltiples soluciones usando varias estructuras de datos y algoritmos. Posteriormente, evaluar la idoneidad, fortalezas y debilidades seleccionando un enfoque que satisfaga mejor los requisitos [Crear] • Explicar factores más allá de la eficiencia computacional que influyen en la elección de algoritmos, como el tiempo de programación, la mantenibilidad y el uso de patrones específicos de la aplicación en los datos de entrada [Explicar] • Para cada uno de los algoritmos y enfoques algorítmicos en los temas del Núcleo KA: – Explicar un ejemplo prototípico del algoritmo, y – Explicar paso a paso cómo opera el algoritmo. [Explicar]

Lecturas : [Cor+22], [KT05]

Unidad 3: Algoritmos Avanzados (4 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Algoritmos de criptografía (por ejemplo, SHA-256) • Algoritmos paralelos • Algoritmos de consenso (por ejemplo, Blockchain): – Prueba de trabajo vs prueba de participación • Algoritmos de computación cuántica: – Basados en oráculo (por ejemplo, Deutsch-Jozsa, Bernstein-Vazirani, Simon) – Aceleración superpolinomial mediante QFT (por ejemplo, Shor) – Aceleración polinomial mediante amplificación de amplitud (por ejemplo, Grover) • Algoritmo de Transformada Rápida de Fourier (FFT) • Algoritmo de evolución diferencial	• Una apreciación de la computación cuántica y su aplicación a ciertos problemas [Explicar]
Lecturas : [Cor+22], [KT05]	

Unidad 4: Estrategias Algorítmicas (24 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Paradigmas: – Fuerza Bruta (por ejemplo, búsqueda lineal, ordenamiento por selección, viajante de comercio, mochila) – Disminuir y Vencer: ∗ Por una Constante (por ejemplo, ordenamiento por inserción, ordenamiento topológico) ∗ Por un Factor Constante (por ejemplo, búsqueda binaria) ∗ Por un Tamaño Variable (por ejemplo, Euclides) – Dividir y Vencer (por ejemplo, búsqueda binaria, quicksort, mergesort, Strassen) – Voraz (por ejemplo, Dijkstra, Kruskal, Mochila) – Transformar y Vencer: ∗ Simplificación de instancia (por ejemplo, encontrar duplicados mediante preordenamiento de lista) ∗ Cambio de representación (por ejemplo, heapsort) ∗ Reducción de problema (por ejemplo, mínimo común múltiplo, programación lineal) ∗ Programación dinámica (por ejemplo, Floyd, Marshall, Bellman-Ford) – Compromisos espacio vs tiempo (por ejemplo, hash) • Manejo del crecimiento exponencial (por ejemplo, heurística A*, ramificación y poda, retroceso) • Iteración vs recursión (por ejemplo, factorial, búsqueda en árbol) • Paradigmas: – Algoritmos de aproximación – Mejora iterativa (por ejemplo, Ford-Fulkerson, simplex) – Algoritmos aleatorizados/estocásticos (por ejemplo, corte máximo, bolas y cubos) • Computación cuántica	• Para cada uno de los paradigmas en esta unidad: – Explicar sus características definitorias – Explicar un ejemplo que demuestre el paradigma incluyendo cómo este ejemplo satisface las características del paradigma. [Explicar] • Para cada uno de los algoritmos en la unidad Fundamentos Algorítmicos (AL)-FoundationalDataStructuresAlgorithms, explicar el paradigma utilizado por el algoritmo y cómo ejemplifica este paradigma [Explicar] • Dado un algoritmo, explicar el paradigma utilizado por el algoritmo y cómo ejemplifica este paradigma [Explicar] • Dar un problema del mundo real, evaluar paradigmas algorítmicos apropiados y algoritmos de estos paradigmas que aborden el problema incluyendo evaluar los compromisos entre los paradigmas y algoritmos seleccionados [Evaluar] • Dar ejemplos de algoritmos iterativos y recursivos que resuelvan el mismo problema, explicar los beneficios y desventajas de cada enfoque [Explicar] • Evaluar si un enfoque voraz conduce a una solución óptima [Evaluar] • Explicar varios enfoques para abordar problemas computacionales cuyas soluciones algorítmicas son exponenciales [Explicar]
Lecturas : [Cor+22], [DPV06]	

Unidad 5: Marco de Análisis de Complejidad (6 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Marco de Análisis de Complejidad: – Rendimiento de un algoritmo en el mejor caso, caso promedio y peor caso – Mediciones empíricas y relativas (Orden de Crecimiento) – Tamaño de entrada y operaciones primitivas – Eficiencia de tiempo y espacio • Mediciones empíricas de rendimiento • Compromisos tiempo-espacio en algoritmos	• Preparar una presentación que explique a estudiantes de primer año los conceptos básicos de complejidad algorítmica incluyendo comportamiento de algoritmo en mejor caso, caso promedio y peor caso, notaciones Big-O, Omega y Theta, clases de complejidad, compromisos tiempo-espacio, medición empírica e impacto en problemas prácticos [Explicar] • Para cada algoritmo en la unidad Fundamentos Algorítmicos (AL)-FoundationalDataStructuresAlgorithms, explicar su clase de complejidad de tiempo de ejecución y por qué pertenece a esta clase [Explicar] • Evaluar informalmente la clase de complejidad fundamental de algoritmos simples [Evaluar] • Desarrollar estudios empíricos para determinar y validar hipótesis sobre la complejidad de tiempo de ejecución de varios algoritmos ejecutando algoritmos con entradas de varios tamaños y comparando el rendimiento real con el análisis teórico [Crear] • Explicar ejemplos que ilustren los compromisos tiempo-espacio de algoritmos [Explicar] • Explicar cómo el balance del árbol afecta la eficiencia de las operaciones de árbol de búsqueda binaria [Explicar]
Lecturas : [Cor+22], [Sip12]	

Unidad 6: Notación Asintótica y Clases de Complejidad (6 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Análisis de complejidad asintótica (cotas promedio y peor caso): – Notaciones formales Big-O, Big-Omega y Big-Theta – Clases de Complejidad Fundamentales y Ejemplos/Problemas Representativos: ∗ $O(1)$ Constante (por ejemplo, acceso a arreglo) ∗ $O(\log_2 n)$ Logarítmica (por ejemplo, búsqueda binaria) ∗ $O(n)$ Lineal (por ejemplo, búsqueda lineal) ∗ $O(n \log_2 n)$ Log Lineal (por ejemplo, mergesort) ∗ $O(n^2)$ Cuadrática (por ejemplo, ordenamiento por selección) ∗ $O(n^c)$ Polinomial (por ejemplo, $O(n^3)$ eliminación gaussiana) ∗ $O(2^n)$ Exponencial (por ejemplo, Mochila, Satisfactibilidad (SAT), Viajante de Comercio (TSP), todos los subconjuntos) ∗ $O(n!)$ Factorial (por ejemplo, circuito hamiltoniano, todas las permutaciones)	• Usando ejemplos, explicar cada una de las clases de complejidad fundamentales en esta unidad [Explicar] • Para cada clase de complejidad fundamental en esta unidad, explicar un algoritmo que demuestre la complejidad de tiempo de ejecución asociada [Explicar] • Explicar a una audiencia no técnica la importancia de los algoritmos tratables versus intratables usando una explicación intuitiva de la complejidad Big-O [Explicar] • Aplicar la notación Big-O para dar cotas superiores en la complejidad tiempo/espacio de algoritmos [Aplicar]
Lecturas : [Cor+22], [Sip12]	

Unidad 7: Análisis de Complejidad II: Recursión, Amortización y Cotas Ajustadas (12 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Notaciones Little-o, Little-Omega y Little Theta • Análisis recursivo formal • Análisis amortizado	• Dado un problema para programar para el cual puede haber varios enfoques algorítmicos, evaluarlos y determinar cuáles son factibles, y seleccionar uno que sea óptimo en implementación y comportamiento de tiempo de ejecución [Evaluar] • Usar relaciones de recurrencia para evaluar la complejidad de tiempo de algoritmos definidos recursivamente [Aplicar] • Aplicar relaciones de recurrencia elementales usando una forma del Teorema Maestro [Aplicar]
Lecturas : [Cor+22], [Sip12]	

Unidad 8: Teoría de la Complejidad Computacional (6 horas)	
Resultados esperados:	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Tractabilidad e intractabilidad: – Clases de Complejidad P, NP y NP-Completo – Problemas NP-Completos (por ejemplo, SAT, Mochila, TSP) – Reducciones • Modelos de complejidad basados en Máquinas de Turing: – Complejidad de tiempo: ∗ Clases P, NP, NP-C y EXP ∗ Teorema de Cook-Levin – Complejidad de espacio: ∗ NSpace y PSpace ∗ Teorema de Savitch	• Explicar la importancia de la NP-Completitud [Explicar] • Explicar cómo NP-Duro es una cota inferior y NP es una cota superior para la NP-Completitud [Explicar] • Explicar ejemplos de problemas NP-completos [Explicar] • Explicar el Teorema de Cook-Levin y la NP-Completitud de SAT [Explicar] • Explicar las clases P y NP [Explicar] • Demostrar que un problema es NP-Completo reduciendo un problema NP-C clásico conocido a él (por ejemplo, 3SAT y Clique) [Crear] • Explicar la clase P-espacio y su relación con la clase EXP [Explicar]
Lecturas : [Sip12], [Cor+22]	

8. PLAN DE TRABAJO

8.1 Metodología

Se fomenta la participación individual y en equipo para exponer sus ideas, motivándolos con puntos adicionales en las diferentes etapas de la evaluación del curso.

8.2 Sesiones Teóricas

Las sesiones de teoría se llevan a cabo en clases magistrales donde se realizarán actividades que propicien un aprendizaje activo, con dinámicas que permitan a los estudiantes interiorizar los conceptos.

8.3 Sesiones Prácticas

Las sesiones prácticas se llevan en clase donde se desarrollan una serie de ejercicios y/o conceptos prácticos mediante planteamiento de problemas, la resolución de problemas, ejercicios puntuales y/o en contextos aplicativos.

9. SISTEMA DE EVALUACIÓN

********* EVALUATION MISSING ********

10. BIBLIOGRAFÍA BÁSICA

[KT05] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Pearson, 2005.

[DPV06] Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill Education, 2006.

[Sip12] Michael Sipser. *Introduction to the Theory of Computation*. 3rd. Cengage Learning, 2012.

[Cor+22] Thomas H. Cormen et al. *Introduction to Algorithms*. 4th. MIT Press, 2022.