UNIDAD TEMÁTICA 3
Práctico domiciliario 1

Ejercicio #1

La posible respuesta es **a o d** según mis interpretaciones**.** El nodo queda insertado antes de nodo1. Primero se crea un nuevo nodo, luego ese nodo apunta al primer nodo de la lista. Luego, por algúna razón, establece al nodo3 como siguiente del nodo2, a pesar de que ya era así. En este punto puedo pensar que se ignora el hecho de que hay una referencia al primer nodo y que este debe cambiar al nuevo nodo para que tenga efecto en la lista.

Ejercicio #2

La respuesta es **c,** elimina nodo2 de la lista. Se crea un nuevo nodo *otroNodo* y posteriormente lo iguala a *nodo1.siguiente*, que es *nodo2.* Luego *nodo1.siguiente*, que apuntaba a *nodo2*, ahora apunta al *nodo3*, es decir que salta el *nodo2*, este sigue apuntando a *nodo3* pero no es accesible. *OtroNodo*, al estar igualado a nodo2, apunta también a *nodo3* pero es inaccesible.

Ejercicio #3

La respuesta es **b,** inserta *otroNodo* entre *nodo1* y *nodo2*. Crea un nodo, hace que referencie al *nodo2*, luego hace que el *nodo1*, en vez de referenciar al *nodo2*, referencie a *otroNodo.*

Ejercicio #4

El algoritmo está mal y siempre dará error en tiempo de ejecución. El problema que tiene está en el bucle, *nodoActual* se actualiza al nodo siguiente correctamente, pero en la última iteración *nodoActual* se iguala a null. Esto genera un error porque posteriormente se intenta acceder al siguiente de un nodo nulo. Tampoco se maneja el caso borde donde la lista está vacía.

Ejercicio #5

El algoritmo está mal porque no maneja el caso en done la lista está vacía.

Primero se crea un nuevo nodo *otroNodo,* luego se crea *nodoActual*, que va a servir como nodo temporal para recorrer la lista. Se iguala al primer nodo en la lista, que si está vacía es **null**. Posteriormente entra en un bucle que se ejecuta mientras *nodoActual* no sea nulo, dentro de este bucle *nodoActual* se va actualizando al siguiente de manera sucesiva hasta que encuentra el nodo final (un nodo es el último en la lista cuando apunta a null). Finalmente hace que *nodoActual (*igualado al nodo último*)* apunte al nuevo nodo.

En el caso de una lista vacía, no entra al bucle e intenta acceder al próximo de un nodo nulo e igualarlo a *otroNodo.* Esto genera un error en tiempo de ejecución.

Ejercicio #6

Dada la información proporcionada, se puede considerar:

- La cantidad de alumnos es indeterminada:
  - **Con un array:** Un Array es un arreglo de celdas que se almacenan en memoria de manera contigua, cuando se declara un Array, se debe especificar la extensión de lo que se reservará en memoria. Cuando no se sabe previamente qué tamaño tendrá, se corre el riesgo de desperdiciar espacio si se terminan anotando pocos alumnos o de tener que redimensionar el array si hay más estudiantes de lo anticipado o si se elimina un estudiante y se deben rellenar los espacios vacíos.
  - **Con una lista enlazada:** Una lista enlazada consiste en uno o más nodos enlazados consecutivamente de manera que cada uno apunta al siguiente. Cada nodo almacena los datos que se precisan y una referencia al siguiente nodo. Esta referencia representa un costo de memoria adicional que en el Array no existe porque los elementos están almacenados de manera contigua y no se referencian entre sí. El costo de memoria de una referencia es de 4 bytes, que se agregarían como sobrecarga a cada nodo.

Finalizado el divague correspondiente, se puede resumir que, en cuanto a costo de memoria se puede decir que:

1. Una lista implementada con un array puede malgastar espacio porque siempre está usando la cantidad máxima del espacio reservado sin importar cuantos elementos hayan realmente. En este sentido, la realización con una lista enlazada garantiza un uso más eficiente de la memoria, porque utiliza sólo el espacio ocupado por los elementos que actualmente tiene.
2. Probablemente la diferencia más notoria está en el hecho de que, en una lista enlazada, cada nodo almacena, además de los datos correspondientes, una referencia al nodo siguiente, que puede llegar a ser una sobrecarga importante en cuanto a costo de memoria.

En resumen, cualquiera de las dos aproximaciones podría usar más espacio que la otra dependiendo de las circunstancias.

En cuanto a lo referente a la cantidad de alumnos que soporta cada tipo de estructura:

- **Con array:** Si se puede predecir aproximadamente la cantidad de alumnos que se van a inscribir, (puede ser en caso de haber un cupo fijo) entonces se minimiza el desperdicio de memoria de un array. Si un estudiante detesta el curso de algoritmos y decide darse de baja, queda un hueco en el array que resultaría en rellenar el espacio moviendo los elementos de lugar, una operación nefasta.
- **Con lista enlazada:** Si la cantidad de alumnos es indefinida y la inscripción es dinámica, una lista enlazada puede ser mejor opción. No hay desperdicio de memoria y si un alumno detesta el curso de sistemas operativos, y decide darse de baja, no hay necesidad de redimensionamiento ni de rellenar espacios vacíos, simplemente una operación constante de cambio de dirección de referencias.