Programación Avanzada y Estructuras de Datos
Exam, Form: A

Name: _____
Student Number: _____
TA: _____
Date: _____

1. El operador `new` en C++:
   (a) Reserva memoria que se libera automáticamente al salir de la función en la que se invocó a `new`
   (b) Reserva memoria que debe liberarse con el operador `delete`.
   (c) Sólo puede emplearse para reservar memoria para tipos de datos simples (no clases).
2. El modificador `const` a la derecha de la declaración de un método en el fichero `.h`, como en `int categoriaMasFrecuente() const;`, quiere decir:
   (a) Que el método sólo puede ser invocado por objetos constantes.
   (b) Que el método sólo puede ser invocado por objetos no constantes.
   (c) Que el método no modifica al objeto que lo invoca.
3. En los tipos abstractos de datos, la relación entre especificación e implementación es la siguiente:
   (a) Una especificación puede tener múltiples implementaciones distintas.
   (b) Una implementación puede corresponder a múltiples especificaciones distintas.
   (c) Ninguna de las anteriores es correcta.
4. Un paso de programa es:
   (a) Un conjunto de operaciones cuyo coste temporal NO es dependiente de la talla del problema.
   (b) Un conjunto de operaciones cuyo coste temporal SÍ es dependiente de la talla del problema.
   (c) Una sola línea de código cuyo coste temporal NO es dependiente de la talla del problema.
5. Cuando un algoritmo presenta caso mejor y caso peor:
   (a) No es posible definir una función que nos indique, para cada talla, cuántos pasos de programa necesitará
   (b) El caso mejor suele corresponder a tallas de problema pequeñas
   (c) Normalmente empleamos notación $\Theta$ para especificar su complejidad temporal asintótica

6. En el TAD pila:
   (a) Sólo podemos conocer el valor del primer elemento que se insertó.
   (b) Sólo podemos conocer el valor del último elemento que se insertó.
   (c) Podemos conocer ambos.
7. La complejidad en el mejor caso del algoritmo de búsqueda binaria en un TAD vector implementado mediante un array de C++ es:
   (a) $\Omega(1)$
   (b) $\Omega(\log n)$
   (c) $\Omega(n)$
8. Imagina que empleamos una lista enlazada para implementar el TAD vector. ¿Cuál sería la complejidad en el mejor caso del algoritmo de búsqueda binaria?
   (a) $\Omega(1)$
   (b) $\Omega(\log n)$
   (c) $\Omega(n)$
9. El coste temporal de insertar un elemento en un árbol binario de búsqueda en el peor caso es:
   (a) $O(1)$
   (b) $O(\log(n))$
   (c) $O(n)$
10. Es posible reconstruir un único arbol binario si conocemos esta pareja de recorridos:
    (a) preorden y postorden
    (b) preorden y niveles
    (c) inorden y niveles
11. Es posible reconstruir un único arbol binario de búsqueda si conocemos este recorrido:
    (a) preorden
    (b) inorden
    (c) Es imposible reconstruir un árbol binario de búsqueda a partir de un único recorrido
12. El grado de un árbol es:
    (a) El número máximo de elementos que puede contenter
    (b) El número máximo de niveles que puede tener
    (c) El número máximo de hijos que tiene un nodo de ese árbol

13. En la operación de borrado en un árbol AVL:
    (a) Siempre se realiza al menos una rotación
    (b) Se realiza como máximo una rotación
    (c) Ninguna de las respuestas anteriores es correcta

14. Imagina que insertamos los mismos elementos, en el mismo orden, en una tabla hash con estrategia de redispersión "hash abierto" y en otra con estrategia de redispersión "hash cerrado con segunda función de hash". Ambas tablas tienen el mismo tamaño y están configuradas para realizar un *rehashing* (duplicar el tamaño) cuando el factor de carga supera el umbral que hace el coste promedio de búsquedas e inserciones deje de ser constante. ¿Cuál de las dos tablas realizará antes el *rehashing*?
    (a) La tabla con estrategia de redispersión "hash abierto"
    (b) La tabla con estrategia de redispersión "hash cerrado con segunda función de hash"
    (c) Ambas realizarán el *rehashing* a la vez

15. El TAD `unordered_map` en C++ está implementado como:
    (a) Un árbol AVL
    (b) Un árbol rojo-negro
    (c) Una tabla hash

16. La complejidad temporal de buscar un elemento (operación `find`) en un conjunto implementado como un vector de bits es:
    (a) $O(n)$
    (b) $O(\log(n))$
    (c) $O(1)$

17. La complejidad temporal asintótica de insertar un elemento en un heap (montículo):
    (a) Es mayor (tarda más) si se representa mediante un vector que si se implementa mediante punteros
    (b) Es menor (tarda menos) si se representa mediante un vector que si se implementa mediante punteros
    (c) Ninguna de las dos opciones anteriores es correcta

18. Al aplicar heapsort empleando un heap máximo:
    (a) El vector se ordena de menor a mayor
    (b) El vector se ordena de mayor a menor
    (c) Ninguna de las dos opciones anteriores es correcta

19. Para implementar un grafo, es preferible emplear la representación mediante lista de adyacencia frente a la representación mediante matriz cuando:

    (a) El grafo es muy disperso (tiene pocos arcos), porque se ahorra mucha memoria

    (b) El grafo es muy denso (tiene muchos arcos), porque el coste temporal de calcular la adyacencia de entrada de un vértice es menor en la lista de adyacencia

    (c) Ambas representaciones son equivalentes en cuanto a tiempos de ejecución y requisitos de memoria y se pueden usar indistintamente

20. Encontrar un arco de cruce en el árbol extendido en profundidad de un grafo dirigido nos garantiza que:

    (a) El grafo contiene un ciclo

    (b) El grafo es conexo

    (c) Ninguna de las anteriores