

Prueba 1

1S - 2015

NOMBRE:

NRO.MATRICULA :

☐ Estructura de Datos ☐ Complejidad Computacional

Programación Orientada a Objetos

Teórico

1. En programación orientada a objetos, un objeto [1 pto]
 - a. puede contener clases.
 - b. es una clase.
 - c. es un programa.
 - d. puede contener datos y métodos.
2. Cuando un objeto quiere hacer algo, este usa un _____. [1 pto]
3. Nombre una ventaja de la herencia en programación orientada a objetos. [1 pto]
 - a. _____
4. ¿Qué valor de z imprime este código? [3 ptos]

```
int x=5;
int z=0;
for (int y=4; y>0; y--) {
    z += x++ - y;
}
System.out.println("z="+z);
```

z = _____

Arreglos

Teórico

5. Insertar un elemento en un arreglo no ordenado [1 pto]
 - a. Requiere múltiples comparaciones
 - b. Toma un tiempo proporcional a tamaño del arreglo
 - c. Toma el mismo tiempo independiente del número de elementos actuales.
 - d. Requiere mover elementos para hacer espacio al nuevo elemento.

6. En un arreglo desordenado, es generalmente más lento encontrar un elemento que no está en el arreglo, a encontrar un elemento que se encuentra en él. JUSTIFIQUE. [3 ptos]

a. verdadero

b. falso

Porque, _____

7. Los arreglos ordenados, comparados con los arreglos desordenados [1 pto]

- a. Son más rápidos de crear.
- b. Son más lentos en la eliminación.
- c. Son más rápidos en la búsqueda.
- d. Son más rápidos en la inserción.

8. Ordene los siguientes tiempos de ejecución del **menos** eficiente al **más** eficiente y dé un ejemplo de algún método que se ejecute en ese tiempo: $O(N^2)$, $O(1)$, $O(\log N)$, $O(N)$. [4 ptos]

a. $O(\quad)$ – ejemplo: _____

b. $O(\quad)$ – ejemplo: _____

c. $O(\quad)$ – ejemplo: _____

d. $O(\quad)$ – ejemplo: _____

9. El número máximo de elementos que deben ser examinados para completar una búsqueda binaria en un arreglo de 300 elementos es [1 pto]

- a. 100.
- b. 50.
- c. 12.
- d. 9.

10. La notación **O** indica [1 pto]

- a. El tiempo de ejecución de un algoritmo para el tamaño de una estructura de datos determinada.
- b. El tiempo en segundos que tarda un algoritmo en procesar un número determinado de ítems.
- c. Como el tamaño de la estructura de datos se relaciona con el número de ítems.
- d. Cómo se relaciona la velocidad de un algoritmo al número de ítems.

Práctico

11. Dada la siguiente clase. Implemente el método insertar. No olvide incluir los comentarios respectivos al código según corresponda.
[6 ptos]

```
class arregloOrdenado {  
    private long[] a;  
    private int nElems;  
  
    public arregloOrdenado (int max) {  
        a = new long[max];  
        nElems = 0;  
    }  
}
```

```
public void insertar (long valor)  
{
```

```
}
```

Ordenamiento Simple

Teórico

12. Los algoritmos de ordenamiento de computadores son más limitados que el ordenamiento hecho por humanos porque: [1 pto]
- Los computadores sólo pueden manejar una cantidad limitada de datos.
 - Los humanos son mejores inventando nuevos algoritmos.
 - Los computadores sólo pueden comparar dos cosas a la vez.
 - Los humanos saben lo que hay que ordenar, mientras que las computadoras necesitan una especificación de lo que hay que ordenar.

13. ¿Por que el ordenamiento por Inserción es mejor que el ordenamiento por Selección? [2 pto]

14. ¿Por que el ordenamiento por Selección es mejor que el ordenamiento de la Burbuja? [2 pto]

Práctico

15. Describa los pasos necesarios para ordenar el arreglo A = [15,8,4] usando el Algoritmo de Inserción. Explique en cada paso cual fue la operación realizada (Ejemplo: copiar, mover, etc.) [6 pts]

15	8	4
----	---	---

--	--	--

--

--	--	--

--

--	--	--

--

--	--	--

--

--	--	--

--

--	--	--

--

--	--	--

--

16. Especifique como queda el arreglo después de insertar el elemento con el valor "101" [2 pts]

0	599	←	12	
1	786		13	
2	492		14	
3	192		15	
4	348		16	
5	465		17	
6	166		18	
7	985		19	
8	944			
9	659			
10				
11				

0		←	12	
1			13	
2			14	
3			15	
4			16	
5			17	
6			18	
7			19	
8				
9				
10				
11				

17. Identifique el nombre del algoritmo [4 ptos]

```
public boolean A(long value) {
    int j;
    for(j=0; j<nElems; j++)
        if( value == a[j] ) break;
    if(j==nElems) return false;
    else {
        for(int k=j; k<nElems; k++) a[k] = a[k+1];
        nElems--;
        return true;
    }
}
```

```
public void B() {
    int out, in;
    for(out=nElems-1; out>1; out--)
        for(in=0; in<out; in++)
            if( a[in] > a[in+1] )
                swap(in, in+1);
}
```

```
public void C() {
    int in, out;
    for(out=1; out<nElems; out++) {
        long temp = a[out];
        in = out;
        while(in>0 && a[in-1] >= temp) {
            a[in] = a[in-1];
            --in;
        }
        a[in] = temp;
    }
}
```

```
public int D(int left, int right, long pivot) {
    int leftPtr = left - 1;
    int rightPtr = right + 1;
    while(true) {
        while(leftPtr < right && theArray[++leftPtr] <
            pivot);

        while(rightPtr > left && theArray[--rightPtr] >
            pivot);

        if(leftPtr >= rightPtr)
            break;
        else
            swap(leftPtr, rightPtr);
    }
    return leftPtr;
}
```