UNIDAD TEMÁTICA 2: DISEÑO Y ANÁLISIS DE ALGORITMOS

TRABAJO DE APLICACIÓN 1

Ejercicio #1

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
public static int enRango (int[] a, int bajo, int alto) {
      int contador = 0; O(1)
      for (int i=0; i<a.length; i++) {  O(N)
            if (a[i] >= bajo && a[i] < alto) O(1)
                  contador++; O(1)
      }
      return contador; O(1)
}
```

ORDEN TOTAL O(N)

Ejercicio #2

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
unaFunción ( N de tipo entero)
  i ← 1   O(1)
  j ← N   O(1)
  mientras i < N hacer   O(LOG N)
    j ← N – 1   O(1)
    i ← i * 2   O(1)
  fin mientras
  devolver (j)   O(1)
fin
```

ORDEN TOTAL O(LOG N)

Ejercicio #3

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
int[] cuentas = new int [100];  O(1)
for (int i = 0; i<100; i++) {  O(1)
  cuentas[i] = enRango (notas, i, i+1);  O(N)
}
```

ORDEN TOTAL O(N)

Ejercicio #4

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
unValor (A, N de tipos enteros)
i ← 0                                                    O(1)
Si N < 3 entonces                                        O(1)
devolver (A)                                             O(1)
fin si
mientras i < 3 hacer                                     O(1)
      si arreglo[i] = A entonces                         O(1)
            devolver ((arreglo[0] + arreglo[N-1]) div 2) O(1)
      fin si
      i ← i + 1                                          O(1)
fin mientras
devolver (A div N)                                       O(1)
Fin                                                      ORDEN TOTAL O(1)
```

Ejercicio #5

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
otraFunción (claveAbuscar)
  inicio ← 0 O(1)
  fin ← N-1  O(1)
  mientras inicio ≤ fin hacer  O(LOG N)
    medio ← (inicio + fin) div 2  O(1)
    si (arreglo[medio] < claveAbuscar) entonces  O(1)
      inicio ← medio + 1  O(1)
    sino
      si (arreglo[medio] > claveAbuscar) entonces  O(1)
        fin ← medio – 1   O(1)
      sino
        devolver medio O(1)
      fin si
    fin si
  fin mientras                    ORDEN TOTAL O(LOG N)
  devolver -1  O(1)
fin
```

Ejercicio #6

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
function particion( i, j: integer; pivote: TipoClave): integer;
{divide V[i], .., V[j] para que las claves menores que pivote estén a la izquierda y las mayores o iguales a la derecha. Devuelve el lugar donde se inicia el grupo de la derecha.}
COMIENZO
  L ← i;  O(1)
  R ← j;  O(1)
  Repetir
    intercambia(V[L],V[R]);
    mientras V[L].clave < pivote hacer   L := L + 1; fin mientras  O(N)
    mientras V[R].clave >= pivote hacer R := R - 1; fin mientras  O(N)
  Hasta que L > R   O(1)
  Devolver L;  O(1)
FIN; {particion}
```

ORDEN TOTAL O(N)

Ejercicio #7

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
miFunción
  Desde i = 1 hasta N-1 hacer O(N)
    Desde j = N hasta i+1 hacer   O(N)
      Si arreglo[j].clave < arreglo[j-1].clave entonces   O(1)
        Intercambia(arreglo[j], arreglo[j-1])O(1)
      Fin si
    Fin desde
  Fin desde                                  ORDEN TOTAL O(N**2)
Fin
```