AEDD - Guía Práctica 17: Complejidad

Ejercicios propuestos:

1. Obtener la función O de la función BuscarEnPares:

```
//Busca el numeroBuscado en las posiciones pares
//del arreglo ordenado pasado como parámetro.
//Si lo encuentra devuelve la posición; si no lo encuentra devuelve -1
```

Variante 1:

```
#define N 10
int BuscarEnPares(int arreglo[], int numeroBuscado) {
	bool encontrado = false;
	int i = 0;
	while (!encontrado && i < N) {
		if (arreglo[i] == numeroBuscado)
			encontrado = true;
		else
			i+= 2;
	}
	if (encontrado)	return i;
	else		return -1;
}
```

Variante 2:

```
int BuscarEnPares(int arreglo[], int numeroBuscado) {
	int izquierda = 0, derecha = N - 1;
	while (izquierda <= derecha)	{
		int m = izquierda + (derecha-izquierda )/2;
		if (arreglo[m] == numeroBuscado && m % 2 == 0)
			return m;
		if (arreglo[m] < numeroBuscado)
			izquierda = m + 1;
		else
			derecha = m - 1;
	}
	return -1;
}
```

2. Escribir el algoritmo e indicar el orden de complejidad (O) de las siguientes operaciones:

   - **Buscar el máximo elemento en un arreglo desordenado:**
     - Aplicando la definición: "K es el mayor de un arreglo A si (existe un i entre 0 y n-1 tal que K=A[i] y además K>=A[i] para todo i entre 0 y n-1)".
     - Tomando el primer elemento como pivote.
   - **Buscar el máximo elemento en un arreglo ordenado:**
     - descendentemente.
     - ascendemente.
   - **Adivinar un número entre 1 y 1000**
     - sin pistas
     - con pistas
   - **Determinar si un número N es primo.**
   - **Determinar si N números leídos, siendo N<= 10.000 son primos**
   - **Encontrar el menor elemento de la diagonal principal de una matriz cuyos elementos se encuentran ordenados ascendentemente.**
   - **Encontrar el mayor valor de la fila X en una matriz.**
   - **Determinar si un arreglo de caracteres es un palíndromo.**
   - **Para el cálculo de 2n, escribir un algoritmo:**
     - De orden O(n)
     - De orden O(log n)
   - **Unir dos arreglos ordenados pero eliminando los repetidos.**

3. Para el siguiente algoritmo:

```
int arr[N]; //Arreglo de enteros aleatorios ordenados
int l = 0, r = N;
while (l <= r) {
     int m = l + (r - l) / 2;
     if (arr[m] == x)
         return m;

      if (arr[m] < x)
          l = m + 1;
      else
          r = m - 1;
 }
```

   - Obtener la función T(N). Calcular T(N) para N: 4,100,10000, 1000000.
   - Obtener la función O.

4. Para las siguientes implementaciones de Fibonacci, calcular la función T(N) y estimar el valor de la función O:

Variante 1:

```
unsigned long long int variante1 (int n){
	int i = 0;
	int j = 1;
	for (int k = 1; k < n; k++){
		int t;
		t = i + j;
		i = j;
		j = t;
	}
	return j;
}
```

Variante 2:

```
unsigned long long int variante2(int n){
	if (n < 2)
		return n;
	else
		return variante2(n-2) + variante2(n-1);
}
```

Variante 3:

```
unsigned long long int variante3(int n){
	unsigned long long numerador, denominador;

	numerador = pow(1 + sqrt(5.0), n) - pow(1 - sqrt(5.0), n);
	denominador = pow(2,n) * sqrt(5.0);

	return numerador / denominador;
}
```