ANÁLISIS Y DISEÑO DE ALGORITMOS I
Práctico Nº 2. Análisis de eficiencia de algoritmos recursivos

Expresar en notación Big-Oh la complejidad temporal de las siguientes funciones:

1.
```
unsigned int Factorial (unsigned int i){
    if (i < = 1)
          return 1;
    else
          return i * Factorial (i -1);
  }
```

2.
```
unsigned int Funcion1 (unsigned int i){
    if (i <= 1)
          return 1;
    else {
          unsigned int aux = Funcion1 (i-1);
          return 2 * aux;
          }
  }
```

3.
```
unsigned int Funcion2 (unsigned int i){
  if (i <= 1)
          return 1;
  else
          return Funcion2 (i-1)+ Funcion2 (i-1);
  }
```

Compare la complejidad de Funcion2 con Funcion1 del inciso 2

4.
```
double Potencia (double base, int exp) {
  if (exp == 0)
          return 1.0;
  if (exp%2 == 1)
          return base * Potencia (base, exp-1);
  else  {
          double  aux = Potencia (base, exp/2);
          return aux * aux;}
          }
```

Compare la complejidad de esta función con la función Potencia iterativa implementada en el práctico 1.

5.
```
int funcion (unsigned int x) {
    if (x = = 0)
          return(1);
    else
          return  F (3* funcion (x-1) ); //  F ∈ O(k)
}
```

6.
```
bool busqueda (int A[],int izq, int der, int dato)   {
// A es un arreglo de números ordenados de menor a mayor

     if (der == izq)
          if (A [der] == dato)
               return true;
          else
               return false;
     else {
          int medio = (der + izq ) / 2;
          if (A [medio] == dato)
               return true;
          else
               if (dato < A [medio])
                    return busqueda (A, izq, medio -1, dato);
               else
                    return busqueda (A, medio + 1, der, dato);
          }
   }
```

7. Dada la siguiente estructura resuelva:

```
     struct nodo{
               int dato;
               struct nodo *izq;
               struct nodo *der;
               };
```

a.
```
bool Pertenece (struct nodo* arbol, int elemento) {
   if ( arbol != NULL)
          if (arbol -> dato == elemento)
               return true;
          else
               if (arbol-> dato < elemento)
                    return Pertenece (arbol ->izq, elemento);
               else
                    return Pertenece (arbol ->der, elemento);
   else
          return false;
   }
```

b.
```
bool Buscar (struct nodo* arbol, int elemento) {
   if ( arbol != NULL)
          if (arbol -> dato == elemento)
               return true;
          else
               return Buscar (arbol ->izq, elemento) ||
                      Buscar (arbol ->der, elemento)  ;
   else
          return false;
   }
```

c.
```
void preorden (struct nodo* arbol) {
        if (arbol) {
              procesar (arbol -> dato); // procesar ∈ O(k)
              preorden (arbol -> izq);
              preorden (arbol -> der);

        }
   }
```

8.
```
void dibujarLineas( int i, int j, unsigned int altura){
   if (altura != 0) {
        int m= ( i + j ) / 2 ;
        trazar (m, altura);  // trazar ∈ O(altura)
        dibujarLineas (i, m, altura - 1);
        dibujarLineas (m, j, altura - 1);
        }
 }
```

9.
```
int A [MAX], aux [MAX];

void intercalar (int izq, int der, int m) {
   for (int i = m+1; i > izq; i--)
        aux[i-1] = A[i-1];
   for (int j = m; j < der; j++)
        aux[der+m-j] = A[j+1];
   for (int z = izq; z <= der; z++)
        A[z] = (aux[i] < aux[j])? aux[i++] : aux[j--];
}

void mergesort (int izq, int der) {
   if  ( der > izq ) {
        int medio = (izq + der) / 2;
        mergesort ( izq, medio);
        mergesort ( medio + 1, der );
        intercalar ( izq, der, medio);
        }
   }
```