ANÁLISIS Y DISEÑO DE ALGORITMOS I

Práctico Nº 1. Análisis de eficiencia de algoritmos

1- De acuerdo a la definición formal de la notación Big-Oh (O) y Big-Omega (Ω), determine si las siguientes sentencias son verdaderas o falsas:

a. $99 \in O(1)$
b. $15n \in O(n)$
c. $15n \in \Omega(n)$
d. $15n \in O(n^2)$
e. $15n \in \Omega(n^2)$
f. $n \log n \in O(n)$
g. $n \log n \in O(n^2)$
h. $\sum_{i=1}^{n} i \in O(n^2)$
i. $10n^3 + 15n^4 + 100n^2 \in O(n^4)$
j. $n^3 + 15 \in O(n^2)$

2- Determine el tiempo de ejecución en notación Big-Oh de las siguientes funciones:

a.
```
void Calculo (double a, double b, double c) {
   double resultado;
   resultado = a + b + b*c + (a+b-c)/(a+b) + 4.0;
   printf ("%f\n", resultado);
}
```

b.
```
//----------- Ejercicio 2.b ---------------
#include <stdio.h>
#define m 100000000
#define m1 10000
#define b 31415821

int a;
int mult ( int p, int q){
   int p1, p0, q1, q0;
   p1 = p/m1; p0 = p%m1;
   q1 = q/m1; q0 = q%m1;
   return (((p0 * q1 + p1 * q0) % m1) * m1 + p0 * q0) % m;
}

int random (){
   a= (mult (a,b) + 1) % m;
   return a;
}

void random_1 (){
   scanf ("%d", &a);
   for (int i= 1; i <= 1000000; i++)
        printf ("%d\n ", random());
}
```

```
   void random_2 (){
      int i, N;
      scanf ("%d %d", &N , &a);
            for (i= 1; i <= N; i++)
               printf ("%d\n ", random());
   }

   int main (){
         random_1();
         random_2();
         return 0;
   }

   //----------fin ejercicio 2.b -----------------
```

c.
```
   float Suma (float arreglo[ ], int cantidad) {
      float suma= 0;
      for (int i = 0; i < cantidad; i++)
            suma += arreglo [i];
      return suma;
   }
```

d.
```
   void Transpuesta (int Matriz [][SIZE]){
      for (int i = 0; i < SIZE; i++)
            for (int j = i+1; j< SIZE; j++){
                  int aux= Matriz [i][j];
                  Matriz [i][j]= Matriz [j][i];
                  Matriz [j][i]= aux;
            }
   }
```

e.
```
   void productoMatricesCuadradas (double a[N][N], double b[N][N],
                                   double c[N][N]){
      for (int i= 0; i < N; i++)
          for (int j= 0; j < N; j++){
             c[i][j]= 0.0;
             for (int k= 0; k < N ; k++)
                 c[i][j]+= a[i][k] * b[k][j];
          }
   }
```

f.
```
   bool Buscar_Elemento (float arreglo [ ], int n, int elemento) {
      bool encontro= false;
      for (int i = n-1;   i >= 0 && !encontro ; i--)
            if (arreglo [i] == elemento)
                  encontro= true;
      return encontro;
   }
```

```
g. int Contar (int n, int m){
      int j, contadorSentencias = 0, i = 1;
      while (i <= m) {
            j = n;
            while (j != 0){
                  j = j / 2;
                  contadorSentencias ++;
            }
            i++;
      }
      return contadorSentencias;
   }

h. void par_impar (int k)  {
      int i, j, contadorSentencias = 0;
      for (i= 1; i <= k; i++) {
            for (j= 1; j <= i; j++)
                  contadorSentencias++;
            if (i % 2 == 0)
                  for (j= i; j <= k; j++)
                        contadorSentencias++;
      }
   }

i. unsigned int Fibonacci (unsigned int i) {
      int Fi_1 = 1, Fi_2 = 1, Fi= 1;
      for (int j = 2; j <= i; j++){
            Fi= Fi_1 + Fi_2;
            Fi_2 = Fi_1;
            Fi_1 = Fi;
      }
      return Fi;
   }

j. int Potencia(int base, int n) {
      int i= 1, p= 1;
      while( i <= n ){
            p = p*base;
            ++i;
      }
      return p;
   }

   void ImprimirCalculos (int k, int x){
      for (int i= 0; i <= k; i++){
               int t = Potencia(x,(2*i+1)) / (2*i+1);
               printf("%d  \n",t);
      }
   }
```

```
k. void Ordenamiento_Seleccion (double arreglo [], int n)  {
      for (int i= 0; i < n; i++)  {
            int j= i;
            for (int k= i+1; k < n; k++)
                  if (arreglo [k] < arreglo [j])
                        j= k;
            double aux = arreglo [i];
            arreglo [i] = arreglo [j];
            arreglo [j] = aux;
      }
   }

l. void CaminosMinimos (float costos[][SIZE],float A[][SIZE]) {
      for (int i = 0; i < SIZE; i++)
            for (int j = 0; j < SIZE; j++)
                  A[i] [j] = costos [i] [j];
      for (int k = 0; k < SIZE; k++)
            for (int i = 0; i < SIZE; i++)
                  for (int j = 0; j < SIZE; j++)
                        if (A[i][j] > A[i][k] + A[k][j])
                              A[i][j] = A[i][k] + A[k][j];
   }

m. int BusquedaBinariaIterativa(int Numeros[], int inicio,
                                int fin, int numeroBuscado) {
   // Números es un arreglo de enteros ordenados de menor a mayor
      while (inicio <= fin) {
            int pos = (inicio + fin) / 2;
            if (numeroBuscado < Numeros[pos])
                  fin = pos - 1;
            else
                  if (numeroBuscado > Numeros[pos])
                        inicio = pos + 1;
                  else
                        return pos;
      }
      return -1;
   }

n. void Contador (int n) {
      int x, m, cuenta;
      for (m = 2; m <= n; m++) {
            cuenta = 0;
            x = 2;
            while (x <= m) {
                  x = 2*x;
                  cuenta++;
            }
            printf ("%d", cuenta);
      }
   }
```