Algorítmica

Grado en Ingeniería Informática

Prácticas: Sesión 2
Propiedades de órdenes de eficiencia y resolución de ecuaciones recursivas

Propiedades de los órdenes de eficiencia

1. Demostrar lo siguiente:

   a) $f(n) \in \theta(g(n)) \Leftrightarrow g(n) \in \theta(f(n))$

   b) $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$

   c) $\lim_{n \to \infty} \frac{f(n)}{g(n)} \in \mathbb{R}^{+} \to f(n) \in \theta(g(n))$

   d) $\lim_{n \to \infty} \frac{f(n)}{g(n)} \to 0 \Rightarrow f(n) \in O(g(n)) \wedge f(n) \notin \theta(g(n))$

   e) $\theta(f^2(n)) \equiv \theta(f(n))^2$

   f) $\sum_{i=1}^{n} i^k \in \theta(n^{k+1}) \forall k \in \mathbb{N}$

   g) $\log_a n \in \theta(\log_b n) \forall a, b > 1$

   h) $\sum_{i=1}^{n} i^{-1} \in \theta(\log_2 n)$

2. Encontrar el menor entero *k* tal que $f(n) \in O(n^k)$ para:

   a) $f(n) = 13n^2 + 4n - 73$

   b) $f(n) = \frac{1}{n+1}$

   c) $f(n) = (n-1)^3$

d) $f(n)=\frac{n^3+2n-1}{n+1}$

e) $f(n)=\sqrt{n^2-1}$

3. Indicar la veracidad o falsedad (incluyendo prueba de demostración) de:

a) $2^{n+1}\in O(2^n)$ b) $O((n+1)!)\equiv O(n!)$ c) $\forall f:\mathbb{N}\rightarrow\mathbb{R}^+, f(n)\in O(n)\Rightarrow 2^{f(n)}\in O(2^n)$

Cálculo de eficiencia

1. Calcule la eficiencia de los siguientes algoritmos:

a)

```
void DesglosaParesImpares(const int original[], int nOriginal,
                          int pares[], int & nPares,
                          int impares[], int & nImpares) {

  int i;

  // Inicialmente, pares e impares no tienen componentes utiles
  nPares= 0;
  nImpares= 0;

  for (i= 0; i<nOriginal; i++) {

        if (original[i]%2 == 0) {
              pares[nPares]= original[i];
              nPares++;
        } else {
              impares[nImpares]= original[i];
              nImpares++;
        }
  }
}
```

b)

```
void Multiplica(double **A, double **B, int n, int k, int m, double **C) {

        for (int i= 0; i<n; i++) {
                for (int j= 0; j<m; j++) {
                        C[i][j]= 0;
                        for (int h= 0; h<k; h++)
                                C[i][j]+= A[i][k]*B[k][j];
                }
        }
}
```

c)

```
void QuickSort(int* v, int ini, int fin) {

    int pospivote;
    if(ini < fin){
        pospivote= pivotar(v, ini, fin);
        QuickSort(v, ini, pospivote-1);
        QuickSort(v, pospivote+1, fin);
    }
}
int pivotar(int *v, int ini, int fin) {
    int i= ini, j= fin, aux, pivote;

    pivote= v[ini];
    do { i++; } while (v[i]<=pivote && i<=fin);
    do { j--; } while (v[j]>pivote);
    while (i<j) {
        aux=v[i]; v[i]=v[j]; v[j]=aux;
        do { i++; } while (v[i]<=pivote && i<=fin);
        do { j--; } while (v[j]>pivote);
    }
    aux=v[ini]; v[ini]=v[j]; v[j]=aux;
    return j;
}
```

d)

```
int fibonacciIteracion (int n){

  int f, fn_1=0, fn_2=0;

  if (n<=1) return n;
  else{
    fn_1=1;
    fn_2=0;

    for (int i = 2; i<=n; i++) {
      f= fn_1+fn_2;
      fn_2 = fn_1;
      fn_1 = f;
    }
  }
  return f;
}
```

e)

```
int maximo(int v[], int n) {
  int resul, aux;
  if (n==0) resul= v[0];
  else {
         aux= maximo(v, n-1);
         if (aux>v[n]) resul= aux;
         else resul= v[n];
  }
  return resul;
}
```

f)

```
  long ejemplo2 (int n) {
    int  i, j, k;
    long total = 0;

    for (i = 1; i < n; i++)
        for (j = i+1; j <= n; j++)
            for (k = 1; k <= j; k++)
                total += k*i;
   return total;
 }
```

g)

```
double ElevarDyV(double r, int n) {

    if (n==0) return 1;
    else if (n==1) return r;
    else {
        double subsolucion= ElevarDyV(r, n/2);
        if (n%2 == 0)
            return subsolucion*subsolucion;
        else return r*subsolucion*subsolucion;
    }
}
```

h)

```
int BuscarBinario(double *v, const int ini, const int fin,
                  const double x) {
    int centro;
    if (ini>fin) return -1;

    centro= (ini+fin)/2;
    if (v[centro] == x) return centro;
    if (v[centro]>x) return BuscarBinario(v, ini, centro-1, x);
    return BuscarBinario(v, centro+1, fin, x);
}
```

i)

```
double Seleccion(double *v, const int ini, const int fin, const int i) {

    if (ini == fin) return v[ini];
    int posPivote= pivotar(v, ini, fin);
    if (posPivote == i) return v[posPivote];
    if (posPivote > i) return Seleccion(v, ini, posPivote-1, i);
    return Seleccion(v, posPivote+1, fin, i);
}
```

```
int pivotar(double *v, const int ini, const int fin) {

    double pivote= v[ini], aux;
    int i= ini+1, j= fin;

    while (i<=j) {
        while (v[i]<pivote && i<=j) i++;
        while (v[j]>=pivote && j>=i) j--;

        if (i<j) {
            aux= v[i]; v[i]= v[j]; v[j]= aux;
        }
    }

    if (j>ini) {
        v[ini]= v[j]; v[j]= pivote;
    }
    return j;
}
```

Resolución de recurrencias

1. Indique el orden de eficiencia de algoritmos recursivos cuya ecuación en recurrencias es:

    a) $T(n)= T(n-1)+T(n-2)$
    b) $T(n)= 5T(n-1)-8T(n-2)+4T(n-3)$
    c) $T(n)= 2T(n-1)+1$
    d) $T(n)= 2T(n-1)+n$
    e) $T(n)= 2T(n-1)+n+2^n$
    f) $T(n)= 4T(n/2)+n$
    g) $T(n)= 4T(n/2)+n^2$
    h) $T(n)= 2T(n/2)+n\cdot\log(n)$
    i) $T(n)= 3T(n/2)+c\cdot n$
    j) $T(n)= 2T(n/2)+\log(n)$
    k) $T(n)= 5T(n/2)+(n\cdot\log(n))^2$
    l) $T(n)= T(n/2)\cdot T^2(n/4)$