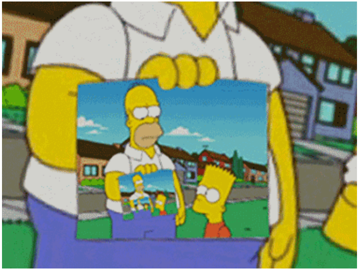
Recursión y Cálculo de complejidad

Introducción a la computación (Matemática)



Ejercicio 1
Exponenciación rápida

```
def naivePower(a,n):
    result = 1
    for i in range(n):
        result = a * result
    return result
```

1. ¿Qué complejidad tiene este algoritmo? O(n)
2. Implementar un algoritmo para calcular la n-ésima potencia de a más rápido (recursión, D&C)

Ejercicio 1
Exponenciación rápida

```
def recursivePower(a, n):
    if n == 0:
        return 1
    y = recursivePower(a, n/2)
    if n%2 == 0:
        return y*y
    else:
        return y*y*a
```

1. ¿Qué complejidad tiene este algoritmo? O(log n)

$T(1) = 3$
$T(n) = T(\lfloor \frac{n}{2} \rfloor) + 8$ para $n > 0$.

Podemos probar por inducción que $T(n) \in O(log_2(n))$.
Queremos ver que $\exists c, n_0 \in \mathbb{N}_{>0}$ tal que $T(n) \leq c \cdot log_2(n), \forall n \geq n_0$.
Elegimos $n_0 = 2$, y $c = 11$.

- Caso base: $T(2) = T(1) + 8 = 11 \leq c \cdot log_2(2) = 11 \cdot 1$ ✓
- Paso inductivo:
  Para $n \geq 3$, sup. $T(n) \leq c \cdot log_2(n)$, quiero ver que $T(n+1) \leq c \cdot log_2(n+1)$.
  $T(n+1) \stackrel{def}{=} T(\lfloor \frac{n+1}{2} \rfloor) + 8 \stackrel{HI}{\leq} 11 \cdot log_2(\lfloor \frac{n+1}{2} \rfloor) + 8 \leq 11 \cdot log_2(\frac{n+1}{2}) + 8 \leq$
  $11 \cdot log_2(n+1) - 11 \cdot log_2(2) + 8 \leq 11 \cdot log_2(n+1) - 3 \leq$
  $11 \cdot log_2(n+1) = c \cdot log_2(n+1)$ ✓

Entonces, $T(n) \in O(log(n))$.

- Un arreglo de enteros pico esta compuesto por una secuencia estrictamente creciente seguida de una estrictamente decreciente.
- Suponemos que hay al menos un elemento a la izquierda y uno a la derecha del máximo.
- Por ejemplo, el arreglo (-1, 3, 8, 22, **30**, 20, 9, 4, 2, 1).

(Problema)

Dado un arreglo pico de longitud n, queremos encontrar al máximo. La complejidad del algoritmo que resuelva el problema debe ser O(log n).

Ejercicio 2

Máximo en arreglo pico

```python
#Precondicion: a es un arreglo pico
def maximo_en_pico(a):
    if len(a) == 3:
        return a[1]

    medio = (len(a)-1)/2

    if a[medio] > a[medio - 1] and a[medio] > a[medio + 1]:
        # a[medio] es el pico
        return a[medio]
    elif a[medio - 1] < a[medio] and a[medio] < a[medio + 1]:
        # a[medio] esta en la parte creciente
        return maximo_en_pico(a[medio:])
    elif a[medio - 1] > a[medio] and a[medio] > a[medio + 1]:
        # a[medio] esta en la parte decreciente
        return maximo_en_pico(a[:medio+1])
```

Dado un arreglo de n enteros, se desea encontrar el maximo valor que se puede obtener sumando elementos consecutivos.

- Por ejemplo, para la secuencia (3, -1, 4, 8, -2, 2, -7, 5), este valor es 14, que se obtiene de la subsecuencia (3, -1, 4, 8).
- Si una secuencia tiene todos números negativos, se entiende que su subsecuencia de suma máxima es la vacía, por lo tanto el valor es 0.

```
def suma_maxima_1(list):
    n= len(list)
    maxsum=0

    for i in range(n):
        for j in range(i,n):

            s=0
            for k in range(i,j+1):
                s+= list[k]

            if s > maxsum:
                maxsum= s

    return maxsum
```

- ¿Cuál es la complejidad? O(n^3)

Ejercicio 3
Subsecuencia de suma máxima

```python
def suma_maxima_2(list):
    n= len(list)
    maxsum=0

    for i in range(n):
        s=0
        for j in range(i,n):
            s+= list[j]
            if s > maxsum:
                maxsum= s

    return maxsum
```

- ¿Cuál es la complejidad? O(n^2)
- Implementar una versión de menor complejidad utilzando la técnica D&C, y luego calcular la complejidad

Ejercicio 4
Matriz creciente

Se tiene una matriz A de n*n números naturales, de manera que $A[i,j]$ representa al elemento en la fila i y columna j $(1 \leq i,j \leqslant n)$.
Se sabe que el acceso a un elemento cualquiera se realiza en tiempo O(1).
Se sabe también que todos los elementos de la matriz son distintos y que todas las filas y columnas de la matriz están ordenadas de forma creciente
(es decir, $i < n \implies A[i,j] < A[i+1,j]$ y $j < n \implies A[i,j] < A[i,j+1]$).

- Implementar, utilizando la técnica de dividir y conquistar, la función: está(n: $\mathbb{N}$, A: $\mathbb{N}[][]$, e: $\mathbb{N}$) $\rightarrow$ bool

  que decide si un elemento e dado aparece en alguna parte de la matriz A.
- El algoritmo debe realizar una menor cantidad de operaciones que recorrer todos los elementos de la matriz. Notar que el tamaño de la entrada es O(n^2).