# Análisis y diseño de algoritmos

---

## Complejidad temporal: Cálculo analítico II

### Solución del ejercicio 1 de la práctica 4

---

**Ejercicio 1**

Realiza un análisis **analítico** de la complejidad temporal de la siguiente función del lenguaje C++. En el supuesto de que existan los casos mejor y peor identifica las instancias que pertenecen a cada caso y obtén las correspondientes cotas de complejidad.

```
1  float
2  Mochila(vector <float> &v,
3           vector<unsigned> &p,
4           unsigned P, int i) {
5   float S1, S2;
6   if (i>=0){
7    if (p[i] <= P)
8         S1= v[i]+Mochila(v,p,P-p[i],i-1);
9    else S1= 0;
10   S2= Mochila(v,p,P,i-1);
11   return max(S1,S2);
12  }
13  return 0;
14 }
```

<u>**Solución:**</u>

El tamaño del problema viene dado por el número de elementos de ambos vectores. Llamémosle $n$. Asumimos que en la llamada inicial el parámetro $i$ toma valor $n-1$

El algoritmo presenta caso mejor y peor (¿Sabrías decur por qué?)

**Complejidad temporal en el mejor de los casos:**

Las instancias que pertenecen al caso más favorable son aquellas en las que $p_i > P\ \forall i$. De esta manera, la llamada recursiva de la línea 8 nunca se realiza. Aún así, la de la línea 10 se realizará siempre. Por lo tanto, podemos expresar la complejidad mediante una relación de recurrencia:

$$T(n) = \begin{cases} 1 & n < 0 \\ 1 + T(n-1) & n \geq 0 \end{cases}$$

El siguiente paso es aplicar el método sustitución para resolver la recurrencia.

$$\begin{aligned} T(n) &\stackrel{1}{=} 1 + T(n-1) \\ &\stackrel{2}{=} 2 + T(n-2) \\ &\stackrel{3}{=} 3 + T(n-3) \\ &\dots \\ &\stackrel{k}{=} k + T(n-k) \end{aligned}$$

Por lo tanto, $T(n) = k + T(n-k)\ \forall k : 1 \leq k \leq n+1;\ T(-1) = 1$
Tomando $k = n+1$ tenemos $T(n) = n + 1 + T(-1) = n + 2$.
De esta manera $c_i(n) \in \Omega(n)$.

**Complejidad temporal en el peor de los casos:**

En el peor de casos están todas las instancias para las que siempre se realizan ambas llamadas recursivas. para que esto ocurra se debe cumplir: $\sum_{i=0}^{n-1} p_i \leq P$. La complejidad puede expresarse mediante la recurrencia:

$$T(n) = \begin{cases} 1 & n < 0 \\ 1 + 2T(n-1) & n \geq 0 \end{cases}$$

Resolviendo:

$$\begin{aligned} T(n) &\stackrel{1}{=} 1 + 2T(n-1) \\ &\stackrel{2}{=} 1 + 2 + 4T(n-2) \\ &\stackrel{3}{=} 1 + 2 + 4 + 8T(n-3) \\ &\dots \\ &\stackrel{k}{=} \Big(\sum_{i=0}^{k-1} 2^i\Big) + 2^k T(n-k) \end{aligned}$$

Por lo tanto, $T(n) = 2^k - 1 + 2^k T(n-k)\ \forall k : 1 \leq k \leq n+1;\ T(-1) = 1$.

Tomando $k = n+1$ tenemos $T(n) = 2^{n+1} - 1 + 2^{n+1} T(-1) = 4 \cdot 2^n - 1$. De esta manera $c_s(n) \in O(2^n)$.