# Más Ejemplos de programación dinámica.

Ahora abordaremos más ejemplos de programación dinámica para que quede más claro.

## Longest Increasing subsequence

Dado un arreglo $A = \{a_0, a_1, \dots, a_{n-1}\}$ buscamos un secuencia de elementos $a_{l_1}, ..a_{l_k} \in A$ donde si $l_i < l_j \implies a_{l_i} < a_{l_j}$ tal que el número de elementos de dicha secuencia sea máximo.

Pongamos un ejemplo concreto. sea $A = \{6, 2, 5, 1, 7, 4, 8, 3\}$ se puede comprobar que la subsecuencia más larga es $\{2, 5, 7, 8\}$ por lo que en este caso la respuesta es 4.

¿Cómo resolver esto?....

Pues podemos definir a $LIS(i)$ como la subsecuencia creciente más grande que termina en $i$. Notemos que $LIS(0) = 1$ puesel cero mismo es la subsecuencia más larga y contiene un único elemento, ahora ¿cómo podriamos obtener el valor de $LIS(i)$? pues notemos que si hay algún indice $j < i$ tal que $a_j < a_i$ entonces un posible valor de $LIS(i)$ puede ser $LIS(j) + 1$ la longitud de la subsecuancia más larga anterior más un elemento que es $i$ pero podría exisitr más de un $j < i$ tal que $a_j < a_i$, entonces simpelemente buscamos entre el mayor de ellos por los que

$$LIS(i) = \max_{j<i}(LIS(j) + 1); a_j < a_i$$

Además notemos que el valor mínimo para $LIS(i) \geq 1$ pues siempre podamos tomar la subsecuancia como la secuencia $\{a_i\}$ así si no existe.

Tomese un tiempo para realizar una simulación a mano de lo anterior para que se convenzan.

COn esto ya estamos listo para programarlo.

### TOP DOWN

```
int LIS(int i){
        if(i == 0) return 0;
        if(memo[i] != -1) return memo[i];
        int maximo = 1;
        for(int j = 0 ; j < i; j++){
                if( A[j] < A[i]){
                        maximo = max(LIS(j) + 1, maximo);
                }
        }
        return memo[i] = maximo;
}
```

### BOTTOM UP

```
for(int i=0 ; i<n ; i++){
    lis[i] = 1;
    for(int j = 0; j<i ; j++){
        if(A[j] < A[i]){
            lis[i] = max(lis[i], lis[j] + 1);
        }
    }
}
```

Complejidad $\mathbb{O}(n^2)$

## Knapsack

La entrada es un conjunto $V = \{v_0, \ldots, v_n\}$ y $W = \{w_0, \ldots, w_n\}$ y un número entero $S$

Para entender mejor imaginemos que tenemos una mochila y tenemos $n$ items, donde el i-ésimo para ti tiene el valor $v_i$ pero un peso $w_i$ y tu mochila solo puede cargar $S$ unidades de peso.

Nuestro objetivo es obtener la mayor ganacia posible con nuestra condiciones.

Una estrategia inmediata es siempre tomar el que tiene mayor valor y de ahí y voler hacer lo mismo con los resultantes. pero.... no funciona 😦

Aquí un contraejemplo:

$V = \{100, 70, 50, 10\}$ , $W = \{10, 4, 6, 12\}$ y $S = 12$ con esta estragia tomamos $v_0 = 100, w_0 = 10$ con lo cual $S_{restante} = 2$ y ya no nos alcanza para nada y solo btuvimos un beneficio total $100$ la cual no es óptima porque pudimos haber obtenido una mejor que es tomar $v_1 = 70, v_2 = 50$ con consto de $w_{total} = w_1 + w_2 = 4 + 6 = 10$ y ganacia de 120 la cual es una mejor opción.

Podemos buscar en todo el espacio, es decir una fuerza bruta inteligente porque claramente hay traslapes de casos, pero preguntemosnos que información es suficiente para definir el estado, ¿únicamente el monto actual?, pues no, podemos llevar al igual que la segunda opción de la recursión del problema de las monedas lo podemos hacer de esa forma veamos como.

$$ganancia(S,k) = \begin{cases} \max(ganancia(S - w_k, k+1) + v_k, ganancia(S, k+1)) & si \quad S > 0 \\ -\infty & si \quad S < 0 \\ 0 & si \quad S = 0 \\ 0 & si \quad k >= n \end{cases}$$

Analicemos esto, Si tenemos aun capacidad lo que hacemos es tomar la mejor opción de tomar o no tomar el objetivo actual que es el k-ésimo.

- Si lo tomamos obtenemos esa ganancia perdemos capacidad y avanzamos al siguiente objeto.
- si no lo tomamos no perdemos capacidad y solo avanzamos al siguiente objeto.

- Si no tenemos capacidad, pues esa opción no debe ser tomada en cuenta por lo que retornamos menos infinito para que a la hora de tomar el máximo sea ignorada.

- Si tenemos cero capacidad es que el anterior si alcanzo por lo que solo retornamos ya no podemos obtener más.

- Y por ultimo si se nos acabaron las opciones puesya no podemos obtener más ganacia.

Sientánse libres de detenerse un moment para simular la recurrencia a mano.

Ahora veamos las implementaciones.

### TOP DOWN

```
int ganancia(int S, int k){
        if(S < 0) return -INF;
        if(S == 0)  return 0;
        if( k >= n) return 0;
        if(memo[S][k] != -1) return memo[S][k];

        return memo[S][k] = max(ganancia(S - W[k], k + 1) + V[k],
ganancia(S, k+1));
}
```

### BOTTOM UP

```
for(int i=0 ; i<= n ; i++) dp[0][i] = 0; // no es necesario solo para
recalcar los casos base

for(int s = 1 ; s <= S_ini; s++){
    for(int k = n - 1; k >= 0 ; k--){
        int x = -INF;
        if( s - W[k] >= 0) x = dp[s - W[k]][k+1] + V[k];
        int y = 0;
        if( k < n) y = dp[s][k+1];
        dp[s][k] = max(x,y);
    }
}
```

Complejidad $\mathbb{O}(Sn)$