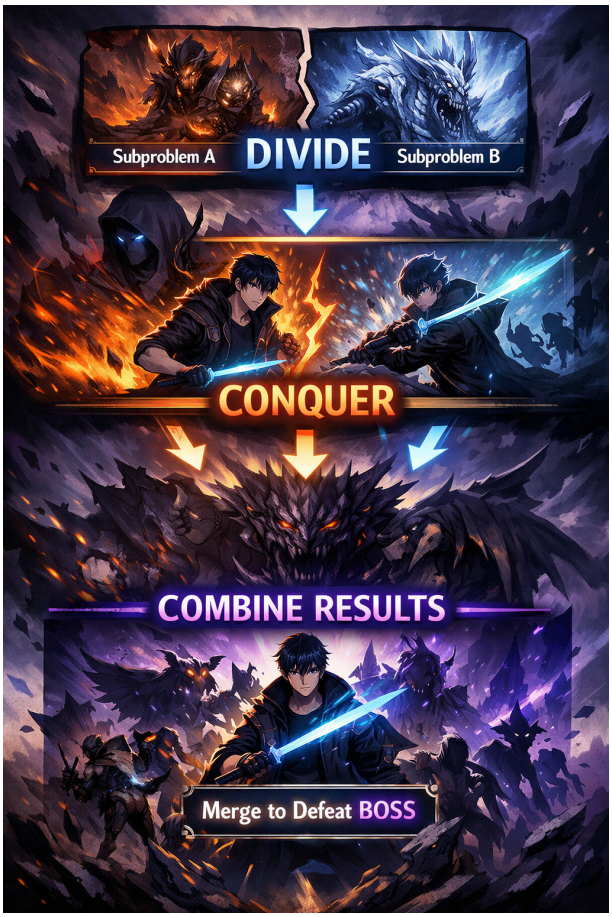
Memoria ejemplo práctica 1



José Antonio Hernández López

Departamento de Informática y Sistemas
Universidad de Murcia

13 de febrero de 2026

Índice

1. Enunciado del problema 📝

Dada una secuencia de $n \geq 1$ números enteros $a_1, a_2, a_3, \ldots, a_n$, se desea encontrar la longitud de la subsecuencia contigua más larga en la que los elementos estén en orden (no estrictamente) ascendente.

Entrada:

- Un entero n, que indica el número de elementos de la secuencia.
- Una secuencia S de n números enteros $[a_1, a_2, a_3, \ldots, a_n]$.

Salida:

- Un único entero que indique la longitud de la subsecuencia contigua más larga donde los elementos estén en orden estrictamente ascendente.

Ejemplo:

- Entrada: $n = 8$ y $S = [2, 2, 9, 3, 4, 4, 1, 6]$.
- Salida: 3
- Explicación: La subsecuencia contigua más larga estrictamente ascendente es $[3, 4, 4]$, que tiene longitud 3.

2. Diseño 💡

En este apartado se presenta el diseño de dos algoritmos: un algoritmo iterativo y un enfoque basado en divide y vencerás.

2.1. Algoritmo iterativo

El Algoritmo 1 muestra el algoritmo que resuelve el problema de manera iterativa y por fuerza bruta. Para cada elemento del array se calcula la mayor secuencia ascendente que empieza él y se coge la mayor de todas. Este algoritmo intuitivo se usará para validar el algoritmo DyV.

Algorithm 1 Subsecuencia más larga no decreciente (iterativo)

```
1:  function SUBSECUENCIAMASLARGAITERATIVO(S, n)
2:      maxLen ← 1
3:      for i ← 0 hasta n − 1 do
4:          currLen ← 1
5:          for j ← i + 1 hasta n − 1 do
6:              if S[j] ≥ S[j − 1] then
7:                  currLen ← currLen + 1
8:                  maxLen ← máx(maxLen, currLen)
9:              else
10:                 break
11:     return maxLen
```

2.2. Algoritmo DyV

El Algoritmo 2 contiene el esquema de DyV adaptado al problema. La idea es la siguiente: si el array es pequeño, se devuelve directamente la solución. En caso contrario, se divide el problema en dos subproblemas, se llama recursivamente al procedimiento principal en cada una de las dos mitades del array y se devuelve la combinación de los dos subproblemas.

Algorithm 2 Esquema DyV adaptado

```
1: function SUBSECUENCIA_MAS_LARGA(S, p, q)
2:     if PEQUEÑO(p, q) then
3:         return SOLUCIÓN_PEQUEÑA()
4:     m ← (p + q)//2
5:     izq ← SUBSECUENCIA_MAS_LARGA(S, p, m)
6:     der ← SUBSECUENCIA_MAS_LARGA(S, m + 1, q)
7:     return COMBINAR(S, p, m, q, izq, der)
```

El Algoritmo 3 y el Algoritmo 4 contienen las definiciones de la función PEQUEÑO y SOLUCIÓN_PEQUEÑA respectivamtente. Un array es considerado

pequeño si tiene solo un elemento. En ese caso, la mayor subcadena ascentente es el propio array de longitud 1.

Algorithm 3 Función PEQUEÑO

```
1: function PEQUEÑO(p, q)
2:     return p == q
```

Algorithm 4 Función SOLUCIÓN_PEQUEÑA

```
1: function SOLUCIÓN_PEQUEÑA( )
2:     return 1
```

Finalmente, Algoritmo 5 define la función de combinación. Primero se comprueba si existe una cadena ascendente que cruza la frontera. Si es así, se calcula la longitud de la secuencia que cruza llevando dos punteros i_{izq} e i_{der} que se muevan hacia la izquierda y derecha respectivamente hasta que se encuentre un elemento que rompa la tendencia creciente.

Algorithm 5 Función COMBINAR

```
1: function COMBINAR(S, p, m, q, izq, der)
2:     if S[m] > S[m + 1] then
3:         return MÁXIMO(izq, der)
4:     i_izq ← m
5:     while i_izq − 1 ≥ p y S[i_izq] ≥ S[i_izq − 1] do
6:         i_izq ← i_izq − 1
7:     i_der ← m + 1
8:     while i_der + 1 ≤ q y S[i_der] ≤ S[i_der + 1] do
9:         i_der ← i_der + 1
10:    cent ← i_der − i_izq + 1
11:    return MÁXIMO(izq, der, cent)
```

3. Implementación ⚙️

El proyecto asociado a esta memoria tiene los siguientes componentes principales[1]:

- Ficheros `dyv.h` y `dyv.cpp`. Estos ficheros contienen las cabeceras y las implementaciones de los algoritmos propuestos.
- Ficheros `tests_unitarios.cpp` y `tests_oraculo.cpp`. El código de estos ficheros se encarga de testear y validar el algoritmo de DyV en forma de

[1] el código fuente está en `https://github.com/AED-UMU/ejemplo-practica-dyv`

tests unitarios y usando la implementación directa como oráculo sobre entradas aleatorias.

- Fichero `tiempos.cpp`. Este fichero contiene la generación de entradas que den lugar al mejor y peor caso junto con la medida de tiempos.
- Fichero `regresion.py`. El código de este fichero se encarga de generar gráficas a partir de los tiempos y de hacer los ajustes de regresión.
- Fichero `makefile`. Describe cómo se compila un proyecto.

4. Validación 🔍

Para validar el algoritmo de DyV se han utilizado dos estrategias: validación con casos de prueba (`tests_unitarios.cpp`) y validación usando la función iterativa como oráculo (`tests_oraculo.cpp`).

4.1. Casos de prueba

Se plantean los siguientes casos de prueba puntuales:

- Mayor subcadena al principio.

  Entrada: $[\mathbf{1, 2, 3}, 2, 3, 4, 1, 2]$

  Salida esperada: 3

- Mayor subcadena al final.

  Entrada: $[1, 2, 3, 2, 3, 4, 1, 2, \mathbf{1, 2, 3, 4}]$

  Salida esperada: 4

- Mayor subcadena en la mitad.

  Entrada: $[1, 2, 3, 4, \mathbf{1, 2, 3, 4, 5, 6}, 1, 2, 3, 4]$

  Salida esperada: 6

- Cadena decreciente (ejemplo de mejor caso).

  Entrada: $[\mathbf{5}, 4, 3, 2, 1]$

  Salida esperada: 1

- Cadena creciente (ejemplo peor caso).

  Entrada: $[\mathbf{1, 2, 3, 4, 5}]$

  Salida esperada: 5

4.2. Oráculo

Para hacer un testeo más exhaustivo, hemos comparado las salidas del algoritmo iterativo con las salidas del algoritmo DyV dado un conjunto aleatorio de entradas. Más concretamente, hemos rellenado con valores aleatorios entre 0 y 100 arrays de longitudes $5, 10, 20, 50, 100, 1000$. Para cada uno de estos seis arrays, hemos ejecutado ambos algoritmos y, posteriormente, hemos comprobado que las entradas coincidían.

5. Análisis teórico 💹

El tiempo del Algoritmo 2 viene dado por la siguiente ecuación. Si el problema original tiene tamaño

$$n = q - p + 1,$$

al dividir en $m = \lfloor (p+q)/2 \rfloor$ se generan dos subproblemas: el izquierdo $[p, m]$ y el derecho $[m+1, q]$. Por tanto, el coste cumple

$$t(n) = t(n_1) + t(n_2) + f(n),$$

donde n_1 y n_2 son los tamaños de los subproblemas izquierdo y derecho, respectivamente, y $f(n)$ es el coste de la combinación.

En concreto,

$$n_1 = (m - p + 1), \qquad n_2 = (q - m),$$

y como $q = p + n - 1$ se tiene

$$m = \left\lfloor \frac{p+q}{2} \right\rfloor = \left\lfloor \frac{p + (p+n-1)}{2} \right\rfloor = \left\lfloor p + \frac{n-1}{2} \right\rfloor = p + \left\lfloor \frac{n-1}{2} \right\rfloor.$$

Sustituyendo:

$$n_1 = m - p + 1 = \left\lfloor \frac{n-1}{2} \right\rfloor + 1 = \left\lceil \frac{n}{2} \right\rceil,$$

y además

$$n_2 = q - m = (p + n - 1) - \left(p + \left\lfloor \frac{n-1}{2} \right\rfloor \right) = n - 1 - \left\lfloor \frac{n-1}{2} \right\rfloor = \left\lfloor \frac{n}{2} \right\rfloor.$$

En consecuencia,

$$t(n) = t\Big(\left\lceil \frac{n}{2} \right\rceil\Big) + t\Big(\left\lfloor \frac{n}{2} \right\rfloor\Big) + f(n).$$

Vamos a asumir que n es potencia de 2 ($n = 2^k$ para algún k entero). Bajo esta suposición, se tiene que:

$$t(n) = 2t(n/2) + f(n).$$

Como el problema se divide en 2 subproblemas del mismo tamaño tenemos $2t(n/2)$. Por otro lado, $f(n)$ engloba todos los cálculos que se hacen en cada llamada a la recursión.

El **mejor caso** viene dado cuando la cadena de entrada está ordenada de manera decreciente. En este caso la operación COMBINAR es constante pues no se ejecutan ninguno de los bucles `while`. Por tanto, $f(n) \in \Theta(1)$. Finalmente, aplicando el teorema maestro, se sigue que $t_m(n) \in \Theta(n)$.

El **peor caso** viene dado cuando la cadena de entrada está ordenada de manera creciente. En este caso la operación COMBINAR es $\Theta(n)$ pues los bucles `while` se ejecutan completos recorriendo todo el array. Así pues, $f(n) \in \Theta(n)$. Finalmente, aplicando el teorema maestro, se sigue que $t_M(n) \in \Theta(n \log n)$.

En el análisis anterior, estamos suponiendo que n es potencia de 2. Sin embargo, podemos generalizarlo para todo n usando el teorema de la $b-$armónica:

- $t(n)$ (con el techo y suelo) es eventualmente no decreciente
- n y $n \log n$ son crecientes
- n y $n \log n$ son $2-$armónicas

6. Análisis experimental 🔬

Los tamaños de entrada considerados son $n_i = c_0 \cdot 2^i$ con $i = 0, \ldots, 10$ y $c_0 = 1000$. c_0 se ha tomado 1000 para un tamaño de inicio no trivial. Se toma un salto mutiplicativo para tener una mejor cobertura del dominio (se cubren varios órdenes de magnitud). Con estos saltos, cada punto es *comparable* al anterior ($n_{i+1}/n_i = 2$).

Siguiendo las recomendaciones de la clase de teoría, para cada tamaño de entrada, se tomaron 10 medidas de tiempo y se calculó la mediana.

La Figura 1 muestra las medidas de tiempo (en ms) tomadas para cada tamaño de entrada n y cada caso. La Figura 2 muestra la misma figura pero usando escala logarítmica en el eje x.

Para la generación de las cadenas que simulen el peor y mejor caso se ha usado el Algoritmo 6 y el Algoritmo 7 respectivamente. El primero devuelve, dado un n, secuencias de la forma $[0, 1, 2 \ldots, n-1]$ y el segundo $[n-1, n-2, \ldots, 1, 0]$.

Algorithm 6 Generar peor caso

```
1: function GENERAR_PEOR_CASO(n)
2:     crear vector secuencia de tamaño n
3:     for i ← 0 hasta n − 1 do
4:         secuencia[i] ← i
5:     return secuencia
```

Algorithm 7 Generar mejor caso

```
1: function GENERAR_MEJOR_CASO(n)
2:     crear vector secuencia de tamaño n
3:     for i ← 0 hasta n − 1 do
4:         secuencia[i] ← n − i
5:     return secuencia
```

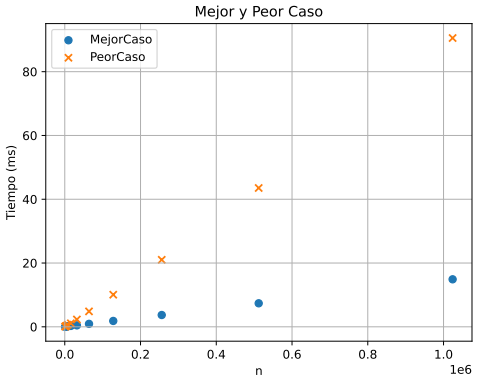


Figura 1: Medidas tomadas para el mejor y peor caso.

7. Contraste ⚖

Para contrastar los órdenes obtenido en el análisis teórico con las medidas obtenidas en el análisis experimental, se ha utilizado una simple regresión lineal.

Se va a explicar únicamente cómo se ha contrastado el tiempo en el peor caso, el mejor caso es análogo. Como $t_M(n) \in \Theta(n \log n)$, eso quiere decir que, para un n suficientemente grande, $t_M(n)$ se comportará de manera similar a $c \cdot n \log n + d$ ($t_M(n) \sim c \cdot n \log n + d$). Esto en la práctica significa que podemos ejecutar un modelo de regresión lineal usando los puntos $(n_i \log n_i, t_i)$ para cada medida tomada (n_i, t_i). De este modelo de regresión lineal podemos sacar el coeficiente de determinación $R^2 \in [0, 1]$ que indica cómo de bueno es el ajuste (cuanto más cerca de 1 mejor).

La Figura 3a y Figura 3b muestran los resultados de la regresión de manera visual para el peor y mejor caso respectivamente (usando escala logarítmica en el eje x). Puede observarse que los ajustes son casi perfectos: los valores R^2 obtenidos son 0,9999 y 1,0000. Con esto concluimos que el análisis teórico cuadra bien con el análisis experimental.

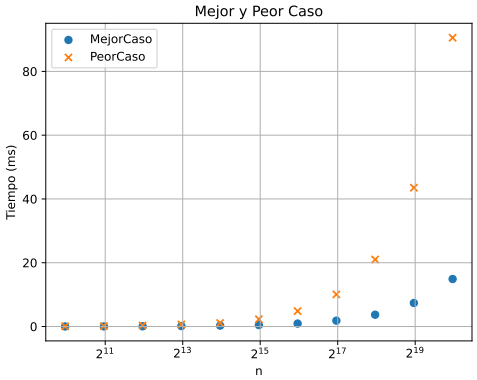
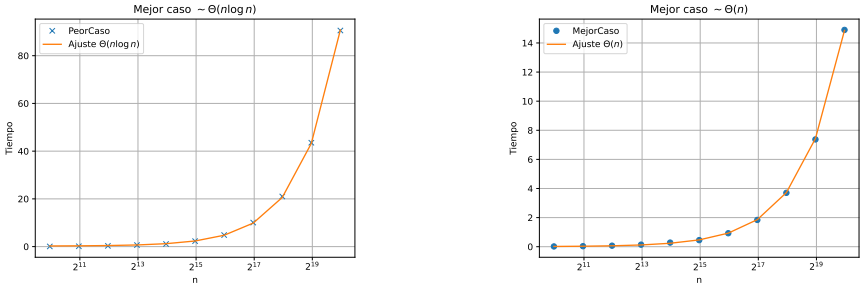


Figura 2: Medidas tomadas para el mejor y peor caso (escala logarítmica).



(a) Ajuste del peor caso.

(b) Ajuste del mejor caso.

Figura 3: Ajustes del mejor y peor caso.

8. Consideraciones importantes ⚠️

- El algoritmo propuesto se puede optimizar llevando dos punteros que indican donde empieza y termina la mejor solución encontrada. Esos punteros se podrían usar en el combinar para optimizarlo.
- No copiéis ni reutilicéis directamente este algoritmo para resolver el problema que tengáis asignado en la práctica 1. Lo más habitual es que no resulte válido, ya que los problemas planteados en dicha práctica son de naturaleza diferente.
- El objetivo de este ejemplo es mostraros qué se espera en cada uno de los apartados solicitados en la práctica 1 (enunciado, diseño, implementación, etc.).
- Se puede usar la IA (siempre y cuando se documente en la memoria) para: generar código de compilación (*makefile*), generar código para medir

tiempos, generar casos de prueba unitarios, documentar el código (siempre que se revise después) y generar código que genere la recta de regresión y gráficas. En vez de usar ChatGPT o similares, os recomiendo usar la extensión de VS Code GitHub Copilot que tiene autocompletado y agentes. Además, los estudiantes de la universidad tienen licencias pro gratuitas.

- En ningún caso la IA se puede utilizar para generar la memoria del proyecto. Se hará especial hincapié a la hora de corregir en la parte del análisis teórico.