

Usamos la notación:

- A : conjunto original
- B_k : Subconjunto de tamaño k de A .
- $C_k = A \setminus B_k$

Proponemos el algoritmo:

1. Ordenamos A .
2. Generamos todos los subconjuntos de tamaño $k - 1$ y los guardamos en una lista S , junto con los elementos restantes de A . Es decir guardamos en S todas las tuplas posibles (B_{k-1}, C_{k-1}) .
3. Para cada tupla (B_{k-1}, C_{k-1}) de S :
 - i. Sumamos todos los elementos de B_{k-1} , llamando a esta suma s .
 - ii. Buscamos en C_{k-1} un número p tal que $s + p = m$ (o, equivalentemente, $p = m - s$). Dado que ordenamos a A previamente, podemos utilizar búsqueda binaria.
 - iii. Si encontramos p , terminamos y regresamos el conjunto $B_{k-1} \cup \{p\}$.
4. Si ya revisamos todas las tuplas y no encontramos ninguna válida, A no tiene elementos que sumen m .

Ahora, hacemos el análisis de complejidad:

1. Asumiendo que usamos heapsort o un algoritmo similar, el ordenamiento toma tiempo $O(n \log n)$.
2. Si A tiene n elementos, el número de subconjuntos de tamaño $k - 1$ es:

$$\begin{aligned}
 & \binom{n}{k-1} \\
 &= \frac{n!}{(k-1)! (n-k+1)!} \\
 &= \frac{n(n-1) \cdots (n-k+2)}{(k-1)!} \\
 &= \frac{n^{k-1} + O(n^{k-2})}{(k-1)!} \\
 &= O(n^{k-1})
 \end{aligned}$$

Por lo tanto, el arreglo S contiene $O(n^{k-1})$ tuplas. Asumiendo que la creación de listas se realiza en tiempo $O(1)$, dado que tenemos que hacer asignaciones para cada tupla, este paso tiene complejidad $O(n^{k-1})$.

3. El arreglo C_k tiene $n - k$ elementos, y por lo tanto la complejidad de la búsqueda binaria sobre él es $O(\log(n - k))$. Como en el peor de los casos (la última tupla es la tupla válida o no hay números que sumen m) tenemos que iterar sobre todas las tuplas, la complejidad será $O(n^{k-1} \log(n - k))$. Si $k \ll n$, esto es equivalente a $O(n^{k-1} \log n)$.

Por lo tanto, la complejidad total del algoritmo es:

$$\begin{aligned} & O(n \log n) + O(n^{k-1}) + O(n^{k-1} \log n) \\ &= O(n^{k-1} \log n) \end{aligned}$$

Para el caso $k = 3$, esto se vuelve $O(n^2 \log n)$, que era lo pedido.

Por otro lado, para la complejidad en espacio:

1. De nuevo, usando heapsort el espacio es $O(1)$.
2. Cada tupla que guardamos en S es una copia del arreglo original A , simplemente partido de distintas maneras; esto ocupa espacio $O(n)$. Como hay $O(n^{k-1})$ tuplas, el espacio total usado por S es $O(n \cdot n^{k-1}) = O(n^k)$.
3. s y p ocupan espacio $O(1)$.

Por lo tanto, la complejidad en espacio es:

$$\begin{aligned} & O(1) + O(n^k) + O(n) \\ &= O(n^k) \end{aligned}$$

Como extra, podemos escribir este algoritmo en pseudocódigo:

Algoritmo 1: Comprueba si existe un subconjunto de tamaño k que sume a un cierto número m .

Entrada: A : Arreglo de números, k : Número de elementos, m : Suma deseada

Salida: Subconjunto de A de tamaño k tal que su suma es m .

inicio

```

     $A \leftarrow \text{sort}(A)$ 
     $S \leftarrow \text{tuples}(A, k)$ 
    para  $t \in S$  hacer
         $s \leftarrow \text{sum}(t[0])$ 
         $p \leftarrow \text{find}(m - s, t[1])$ 
        si  $p \neq \text{NULL}$  entonces
            devolver  $\text{union}(t[0], p)$ 
        fin
    fin

```

fin

Donde:

- `sort(A)`: Ordena el arreglo A de menor a mayor.
- `tuples(A, j)`: Regresa una lista de todas las tuplas posibles (B_k, C_k) previamente definidas.
- `sum(A)`: Suma todos los elementos de A .
- `find(x, A)`: Encuentra el valor x en el arreglo A mediante búsqueda binaria. A debe de estar ordenado de menor a mayor. Si A no contiene x , regresa NULL.
- `union(A, B)`: Une los conjuntos A y B .