Η αναδρομική συνάρτηση βασίζεται στη λογική της δυναμικής επιλογής:  
Για κάθε στοιχείο του πίνακα, έχουμε **δύο επιλογές**:

1. **Να το αγνοήσουμε** και να ελέγξουμε αν μπορούμε να βρούμε το άθροισμα sum στα υπόλοιπα στοιχεία.
2. **Να το συμπεριλάβουμε** στο υποσύνολο, αφαιρώντας την τιμή του από το sum, και να ελέγξουμε αν μπορούμε να βρούμε το νέο άθροισμα sum - array[n - 1] στα υπόλοιπα στοιχεία.

Αυτό οδηγεί στον αναδρομικό ορισμό:

return subsetSum(array, n - 1, sum) || subsetSum(array, n - 1, sum - array[n - 1]);

* **subsetSum(array, n - 1, sum)** → Αν το υποσύνολο υπάρχει χωρίς να πάρουμε το array[n-1].
* **subsetSum(array, n - 1, sum - array[n - 1])** → Αν το υποσύνολο υπάρχει όταν συμπεριλάβουμε το array[n-1].

Αν **έστω μία από τις δύο κλήσεις επιστρέψει 1 (true)**, σημαίνει ότι βρέθηκε υποσύνολο με άθροισμα 0, οπότε επιστρέφουμε **1**. Αν καμία δεν το βρει, επιστρέφουμε **0**.

**Παράδειγμα εκτέλεσης:**

Για τον πίνακα {3, -1, 7, 2, -6} και sum = 0:

1. Ξεκινάμε με όλα τα στοιχεία και sum = 0.
2. Προσπαθούμε είτε να χρησιμοποιήσουμε είτε να αγνοήσουμε το -6.
3. Αυτό διακλαδώνεται σε υποπροβλήματα μέχρι να βρούμε υποσύνολο με άθροισμα 0 ή να εξαντλήσουμε τις επιλογές.

Η αναδρομική συνάρτηση subsetSum(array, n, sum) εξετάζει **δύο περιπτώσεις** για κάθε στοιχείο του πίνακα:

1. **Να αγνοήσουμε το στοιχείο** → subsetSum(array, n - 1, sum)
2. **Να το συμπεριλάβουμε** → subsetSum(array, n - 1, sum - array[n - 1])

Αυτό σημαίνει ότι σε κάθε επίπεδο της αναδρομής, δημιουργούνται **δύο νέες υποκλήσεις**.  
Αν n είναι το πλήθος των στοιχείων του πίνακα, τότε το μέγιστο βάθος της αναδρομής είναι n και ο αριθμός των κόμβων στο δέντρο αναδρομής είναι περίπου:

T(n)=2⋅T(n−1)=2nT(n) = 2 \cdot T(n - 1) = 2^nT(n)=2⋅T(n−1)=2n

Άρα, η **χρονική πολυπλοκότητα** είναι:

O(2n)O(2^n)O(2n)

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, λογισμικό, λογισμικό πολυμέσων

Το περιεχόμενο που δημιουργείται από τεχνολογία AI ενδέχεται να είναι εσφαλμένο.

**Χωρική Πολυπλοκότητα**

Η χωρική πολυπλοκότητα εξαρτάται από το βάθος της αναδρομής, το οποίο είναι **O(n)** λόγω της στοίβας κλήσεων.

**Σύγκριση με άλλες λύσεις**

Η εκθετική πολυπλοκότητα καθιστά αυτή τη λύση μη πρακτική για μεγάλα n (π.χ. n > 30).  
**Βελτίωση:**

* **Memoization (Δυναμικός Προγραμματισμός)** → Μειώνει την πολυπλοκότητα σε **O(n \* sum)**.
* **Bitmasking (Meet in the Middle)** → Μειώνει την πολυπλοκότητα σε **O(2^(n/2))**, καλύτερο για n ≈ 40.