1. First, let's use a Dynamic programming Algorithm to solve this question. let's define $T(i,j)$

$T(i,j)$ = whether we can use elements $a_i \cdots a_n$ to have a sum of j

base cases: $T(i,0)$ for all i is true, $T(n,j)$ for all $j>0$ is false

general recursive definition:

$$T(i,j) = \text{or} \begin{cases} T(i+1, j-a_i) & j-a_i \geq 0 \ \&\& \\ T(i+1, j) \end{cases}$$

then just return $T(0,T)$ will return the answer. The correctness of this DP should be straight. The complexity is that time $O(nT)$ since there are $O(nT)$ entries and $O(1)$ per entry, and since $T(i,j)$ only depends on entries in the $T(i+1, \cdot)$ columns, then it's okay to compute the $T(0,T)$ only memorizing $O(T)$ space, but still it's not enough we cannot output the subset. Therefore, we need to introduce a new function $div(n,k)$ and a new "table" $T'(i,j)$

$T'(i,j) \stackrel{def}{=}$ whether we can use element $a_0 \cdots a_i$ to have a sum of j.

Similarly $T'(i,j) = \text{or}\begin{cases} j-a_i \geq 0 \;\&\&\; T(i-1, j-a_i) \\ T(i-1, j) \end{cases}$ base cases: $T(i, 0) = \text{true}$ for all i

and $T(0, j) = \text{false}$ for all $j > 0$. Then $\text{div}(\{0,1,2 \cdots n, k\}, k)$ is a function that checks for a subset $S \subseteq \{a_0 \cdots a_n\}$ that sums to k, the sum S_1 that is the sum of all elements in $S \in \{a_0 \cdots a_{\lfloor n/2 \rfloor}\}$ and the sum S_2 that's the sum of all elements in $S \in \{a_{\lceil n/2 \rceil} \cdots a_n\}$, the code is:

```
div ({0,1,2 ....n}, k):
    for i from 0 to T
        if T(n/2, i) and T'(n/2 T-i) == True
            return i, T-i

    else: return -1, -1                    :// no such split.
```

then the complexity of $\text{div}(\{0,1, \cdots n\}, k)$ should be clear: For loop is $O(T)$ time the computation of $T(n/2, \cdot)$ is $O(nT)$ time $O(T)$ space, similarly for $T'(n/2, \cdot)$ it's $O(nT)$ time $O(T)$ space, so in total it's $O(nT)$ time with $O(T)$ space. then the function F to output the subset is just.

```
F({0,1,2 ... n}, k)   let's call it set T.
if |T| = 1:
    if a_T0 is k return a_T0
    else return None.
else:
    S1 S2 = div({0,1,2 ... n}, k)
    if S1, S2 == -1: return False:
    else:
        F({0,1,2 ... ⌊n/2⌋}, S1)
        F({⌈n/2⌉ ... n}, S2).
```

The complexity of F will be:

$$F(n, \cancel{k}\,T) = O(nT) + F(n/2, j) + F(n/2, \cancel{k}\,T-j)$$

guess that $F(n,k) \le \alpha \cdot nT$

$$F(n,T) \le \beta nT + \alpha \tfrac{n}{2} \cdot j + \alpha \tfrac{n}{2} \cdot (T-j) = (\beta + \tfrac{\alpha}{2}) nT$$

so it's valid as long as $\alpha > 2\beta$ then we know that

$F(n,T) \le \cancel{O(nT)}\ O(nT)$

Therefore computing the actual subset in total needs ~~O(nm)~~ $O(nT)$ time and $O(T)$ space.

Correctness: Since within function F, we recursively called F based on the output of div() then ~~so~~ if we can prove the correctness div( ) we can ~~so~~ still get the correct results. Then the div() is just checking the correct "contribution" ~~o~~ that the first half and the second half should make, and therefore we recursively cut the range into half and once we reach base case, when range is one, $div(\{a_i\}, k)$ we can check whether the ~~sum~~ k is a_i or k is zero to know a_i is selected or not. Therefore, via recursion we can return all the selected elements and thus output the subset.