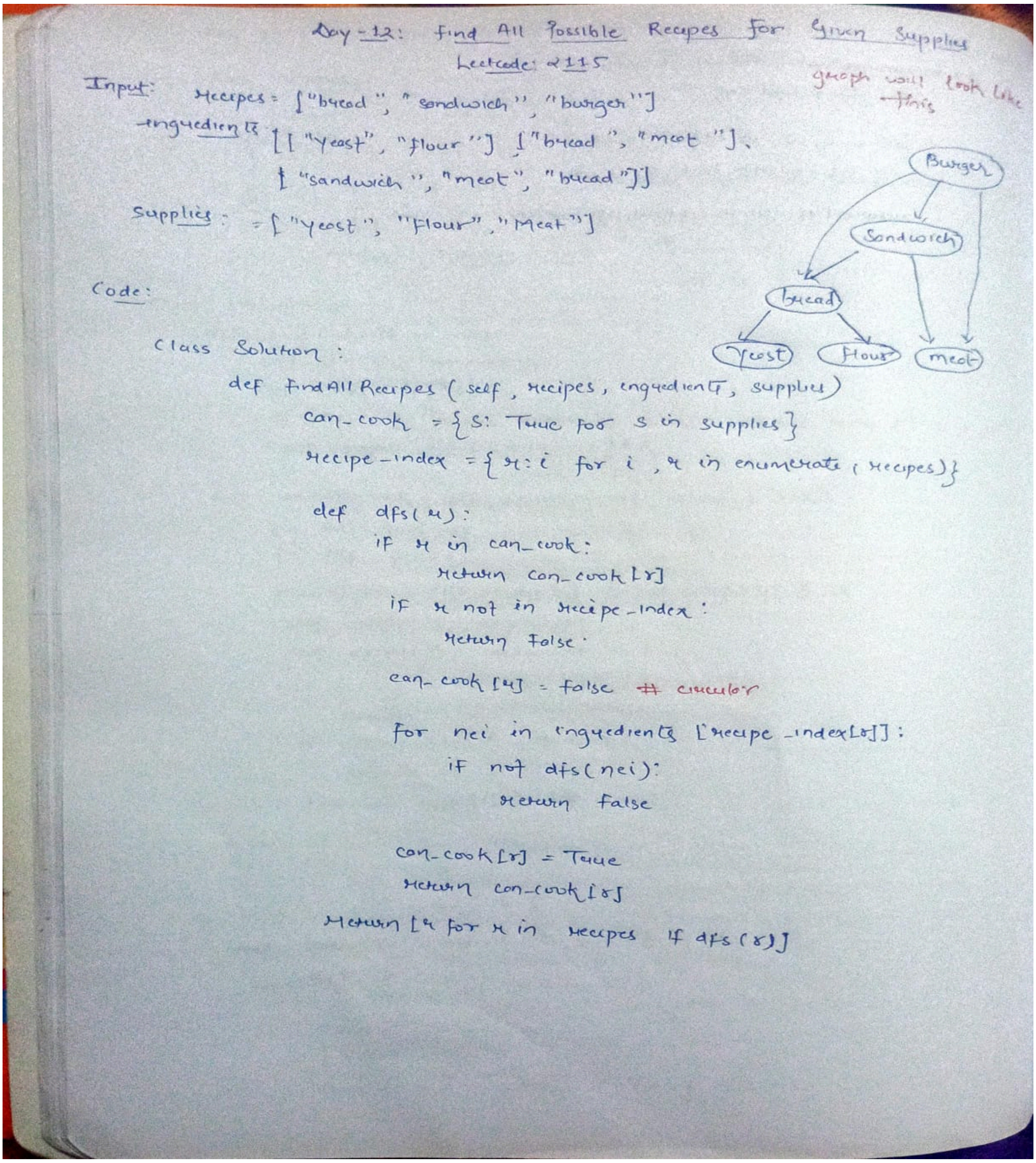
Day_12: DSA



Day-12: find All Possible Recipes for Given Supplies

Leetcode: 2115

Input: recipes = ["bread", "sandwich", "burger"]

ingredients [["Yeast", "flour"] ["bread", "meat"].
["sandwich", "meat", "bread"]]

supplies : = ["yeast", "Flour", "Meat"]

graph will look like this

Burger
Sandwich
bread
Yeast
Flour
meat

Code:

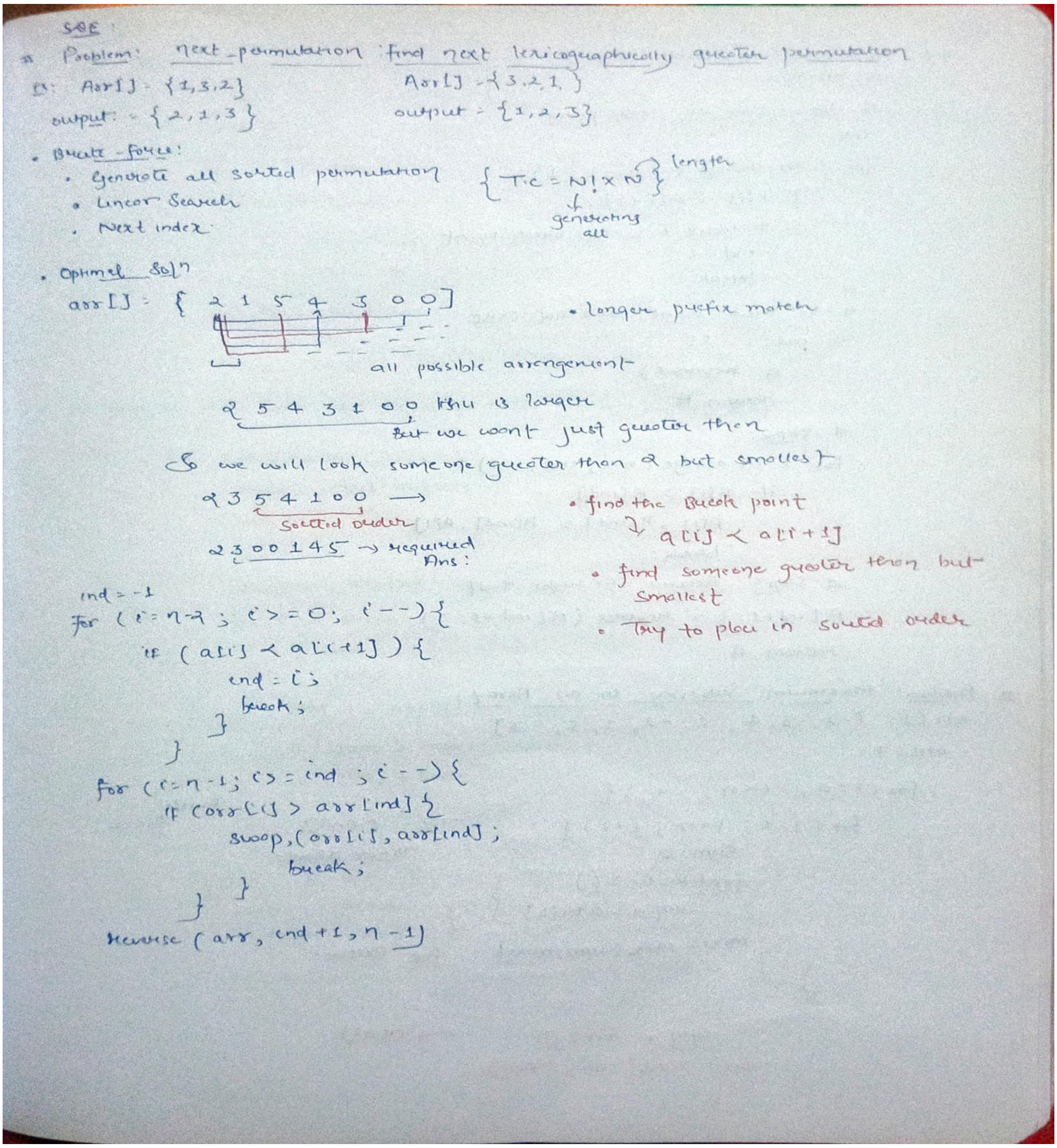
```
class Solution :
    def findAllRecipes ( self, recipes, ingredients, supplies)
        can-cook = {S: True for s in supplies}
        recipe-index = {r:i for i ,r in enumerate ( recipes)}

        def dfs(r):
            if r in can_cook:
                return can-cook[r]
            if r not in recipe-index:
                return False.

            can-cook[r] = false  # circular

            for nei in ingredients [recipe-index[r]]:
                if not dfs(nei):
                    return False

            can-cook[r] = True
            return can-cook[r]
        return [r for r in recipes if dfs(r)]
```



SDE :

* Problem: next_permutation find next lexicographically greater permutation

Ex: Arr[] = {1,3,2} Arr[] = {3,2,1}

output: = {2,1,3} output = {1,2,3}

- Brute - force:
  - Generate all sorted permutation   { T.C = N! x N } length ↓ generating all
  - Linear Search
  - Next index

- Optimal Soln

arr[] = { 2 1 5 4 3 0 0]   • longer prefix match

all possible arrangement

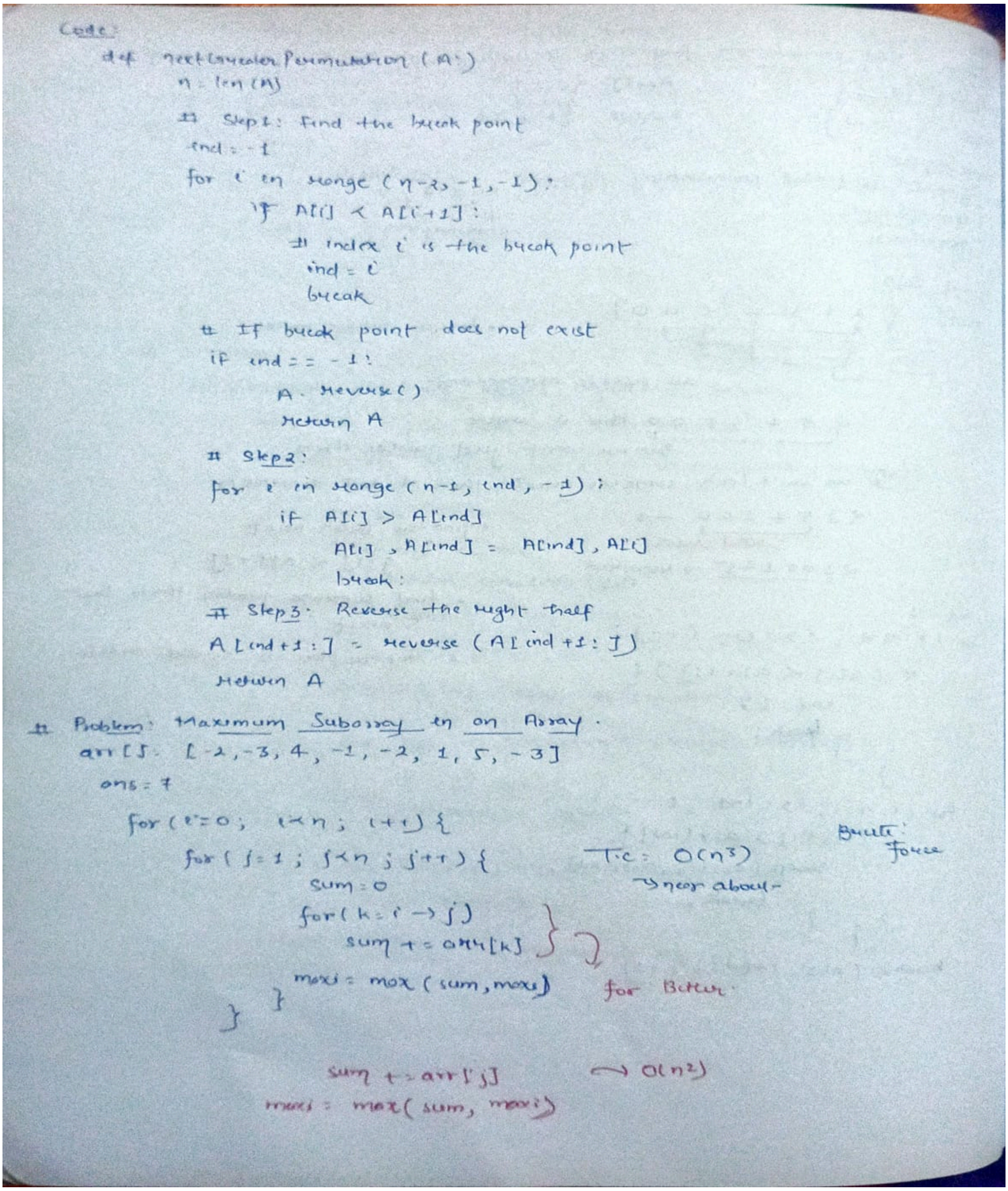
2 5 4 3 1 0 0 this is larger
But we want just greater then

So we will look someone greater than 2 but smallest

2 3 5 4 1 0 0 ⟶   • find the Break point
sorted order   ↳ a[i] < a[i+1]

2 3 0 0 1 4 5 → required Ans: • find someone greater than but smallest
• Try to place in sorted order

```
ind = -1
for ( i = n-2 ; i >= 0; i--){
        if ( a[i] < a[i+1] ) {
                ind = i;
                break;
        }
}
for ( i=n-1; i>= ind ; i--) {
        if (arr[i] > arr[ind] {
                swap,(arr[i], arr[ind] ;
                        break;
        }
}
reverse ( arr, ind + 1, n - 1)
```



Code:

```
def  nextGreaterPermutation (A:)
        n = len (A)

        # Step1: Find the break point
        ind = -1
        for i in range (n-2, -1, -1):
            if A[i] < A[i+1]:
                # index i is the break point
                ind = i
                break

        # If break point does not exist
        if ind == -1:
            A. reverse()
            return A

        # Step2:
        for i in range (n-1, ind, -1);
            if A[i] > A[ind]
                A[i] , A[ind] = A[ind], A[i]
                break:
        # Step3: Reverse the right half
        A [ind+1 :] = reverse (A[ind +1 : ])
        return A
```

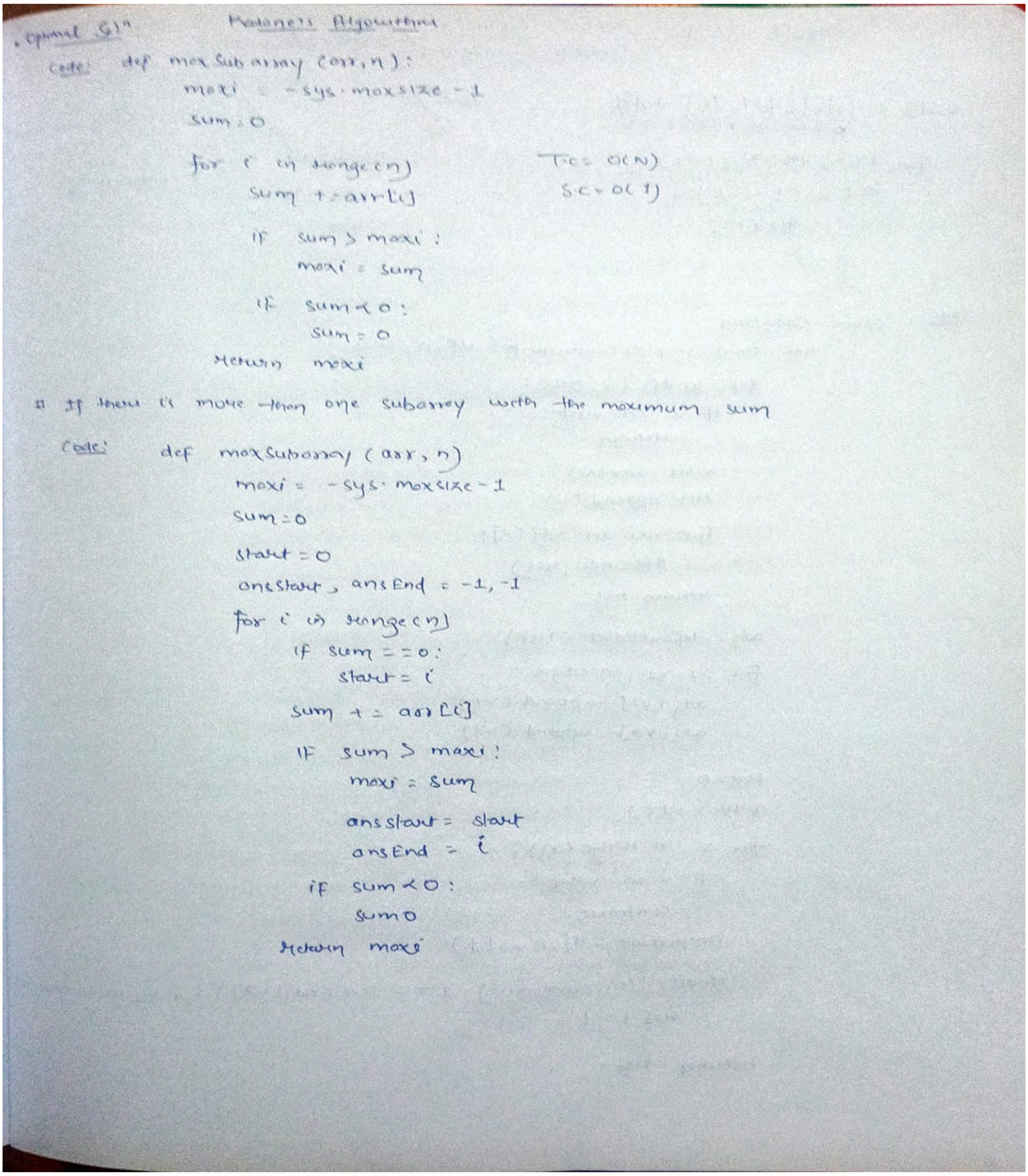
Problem: Maximum Subarray in an Array.

arr[]. [-2, -3, 4, -1, -2, 1, 5, -3]

ans = 7

```
for (i=0;   i<n;   i++) {
    for ( j=1 ;  j<n ;  j++) {            T.C: O(n^3)          Brute
        sum = 0                             → near about-       Force
        for( k = i → j)          }
            sum += arr[k]        }  ⤵
        maxi = max (sum, maxi)        for Better.
    }
}

        sum += arr[j]          ↪ O(n^2)
    maxi = max( sum, maxi)
```



• Optimal Sol^n — Kadane's Algorithm

Code:

```
def maxSubarray (arr, n):
        maxi = -sys.maxsize - 1
        sum = 0

        for i in range(n)                     T.C = O(N)
            sum += arr[i]                     S.C = O(1)

            if sum > maxi :
                maxi = sum

            if sum < 0 :
                sum = 0
        return maxi
```

If there is more than one subarray with the maximum sum

Code:

```
def maxSubarray (arr, n)
        maxi = -sys.maxsize - 1
        sum = 0
        start = 0
        ansStart, ansEnd = -1, -1
        for i in range(n)
            if sum == 0:
                start = i
            sum += arr[i]
            if sum > maxi:
                maxi = sum
                ansstart = start
                ansEnd = i
            if sum < 0:
                sum0
        return maxi
```