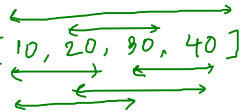
* Generate Subarray : [any continuous segment of the arr]

Eg : [10, 20, 30, 40]



Op :

10
10, 20
10, 20, 30
10, 20, 30, 40

20
20, 30
20, 30, 40

30
30, 40

40

* It uses running stream only (Running subarray)

Q : How to generate all subarrays starting with arr[0] ⇒ 10 ?

```
let subarr = "";  // empty string
for(let i = 0; i < n; i++) {
    subarr += arr[i] + " ";
    console.log(subarr);
}
```

"10" ⟶ cl("10")

"10 20" ⟶ cl("10 20")

"10 20 30" ⟶ cl("10 20 30")

"10 20 30 40" ⟶ cl("10 20 30 40")

Q: Now generate all subarrays using previous logic

```
let subarr = "";  // empty string
for(let i = 0; i < n; i++) {
    subarr += arr[i] + " ";
    console.log(subarr);
}
```

```
let subarr = "";  // empty string
for(let i = 1; i < n; i++) {
    subarr += arr[i] + " ";
    console.log(subarr);
}
```

```
for (let start = 0; start < n; start++) {
    let subarr = "";
    for (let end = start; end < n; end++) {
        subarr += arr[end] + " ";
        cl(subarr);
    }
}
```

1. start = 0 → end = 0 → 10 ...
2. start = 1 → end = 1 → 20 ...
3. start = 2 → end = 2 → 30 ...
4. start = 3 → end = 3 → 40 ...

```
819  const arr = [10, 20, 30, 40];
820  const n = arr.length;
821  for (let start = 0; start < n; start++) {
822    //let subarr = [];
823    let subarr = "";
824    for (let end = start; end < n; end++) {
825      //subarr.push(arr[end]);
826      subarr += arr[end] + " ";
827      console.log(subarr);
828    }
829  }
```

① start = 0

→ subarr = " "

→ end = start = 0, subarr += arr[0] + " " = "10 "

→ end = 1, subarr += arr[1] + " " = "10 20 "

→ end = 2, subarr += arr[2] + " " = "10 20 30 "

→ end = 3, subarr += arr[3] + " " = "10 20 30 40"

② start = 1

→ subarr = " "

→ end = start = 1, subarr += arr[1] + " "
= "20 "

→ end = 2, subarr += arr[2] + " "
= "20 30 "

→ end = 3, subarr += arr[3] + " "
= "20 30 40 "

③ start = 2

→ subarr = " "

→ end = start = 2, subarr += arr[2] + " "
= "30 "

→ end = 3, subarr += arr[3] + " "
= "30 40 "

④ start = 3

→ end = 3, subarr += arr[3] + " "
= "40 "

★ Zero Sum Subarrays :

Eg:

0	1	2	3	4	5
3	4	-7	3	1	3

Op:

[3, 4, -7] → 0 (0 to 2)

[4, -7, 3] → 0 (1 to 3)

[-7, 3, 1, 3] → 0 (2 to 5)

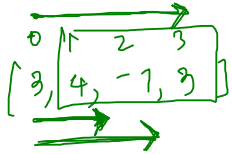
⇒ Involves, subarray
running stream/sum
flag variable

→ In previous concept we need to print each subarray hence we used **running subarray** by taking empty string.

→ Here we need the sum of each subarray, we can use **running sum**.

```
function zeroSubarray(arr) {
  //Write your code here
  const n = arr.length;
  let isFound = false;
  for (let start = 0; start < n; start++) {
    let sum = 0;
    for (let end = start; end < n; end++) {
      sum += arr[end];
      if (sum == 0) {
        console.log(`Subarray found from Index ${start} to ${end}`);
        isFound = true;
      }
    }
  }

  if (isFound == false) {
    console.log(-1);
  }
}
```



(1) s = 0 sum = 0 3 7 0 3

→ i = 0, sum += arr[0]
+= 3

→ i = 1, sum += arr[1]
+= 4

→ i = 2, sum += arr[2]
= 0

→ i = 3, sum += arr[3]
= 3

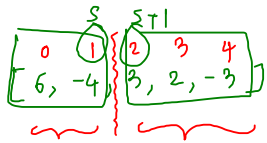
(2) s = 1 sum = 0 4 -3 0

→ i = 1, sum += arr[1]
+ = 4

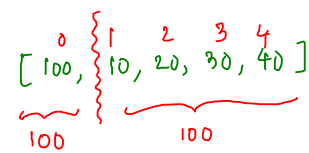
→ i = 2, sum += arr[2]
+ = -1

→ i = 3, sum += arr[3]
+= 3

* Find Split point :

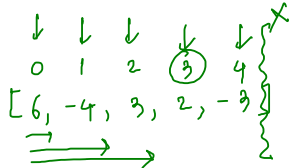


op : 1

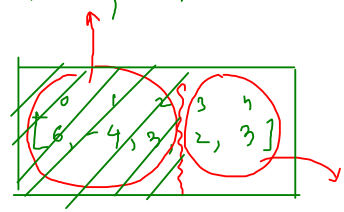


op: 0

op: 6 -4 (0 to split)
3 2 -3 (split+1, n-1)



① running sum,



? total - runningsum
↓
Lpartsum

* RS at i=2, ⇒ RI=5
Indicates left part sum

→ we got Lpartsum, Rpartsum?

⇒ total - Left partsum
⇒ total - runningSum

RS = 6 (i=0)
RS = 2 (i=1)
RS = 5 (i=2)
RS = 7 (i=3)
RS = 4 (i=4) ✗ Split not possible

```
function findSplit(arr, N) {
  let total = 0;
  for (let i = 0; i < N; i++) {
    total += arr[i];
  }

  let lsum = 0;
  for (let split = 0; split < N - 1; split++) {
    lsum += arr[split];
    const rsum = total - lsum;
    if (lsum == rsum) {
      return split;
    }
  }

  // you will reach here only when no split possible
  return -1;
}
```

[10, 20, 30, 40]
0 1 2 3

1. total = 100

2. split = 0, [10, {20, 30, 40}] (90)
lsum += arr[0] ⇒ lsum = 0 + 10 = 10
rsum = 100 - 10 = 90

3. split = 1 ⇒ [10, 20, {30, 40}] (70)
lsum += arr[1] ⇒ lsum = 10 + 20 = 30
rsum = 100 - 30 = 70

4. split = 2 ⇒ [10, 20, 30, {40}] (40)
lsum += arr[2] ⇒ lsum = 30 + 30 = 60
rsum = 100 - 60 = 40

~~5~~. split = 3, 3 < N-1
3 < 4-1
3 < 3 ✗
[10, 20, 30, {40}] ✗

return -1;

★ Geometric triplets :

Q: what is G.P ?

Eg: 2, 4, 8, 16, 32, 64

4/2 = 2 2 2 2 2

a, b, c, d, e

r r r r

⇒ b/a = c/b = d/c = e/d

Eg: 3, 9, 27, 81, 243

3 3 3 3

0 1 2 3 4 5

[1, 2, 6, 10, 18, 54]

1, 2, 6 ⇒ 2/1 == 6/2 ×

1, 2, 10 ⇒ 2/1 == 10/2 ×

1, 2, 18 ⇒ 2/1 == 18/2 ×

1, 2, 54 ⇒ 2/1 == 54/2 ×

1, 6, 10 ⇒ 6/1 == 10/6 ×

2, 6, 18 ⇒ 6/2 == 18/6 ✓

★ Generate all triplets, check whether they are in GP.

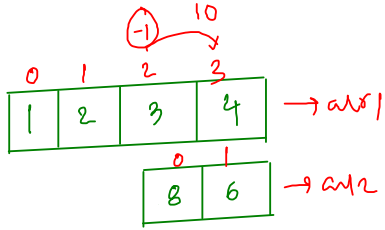
a, b, c → GP

b/a == c/b

$b^2 = ac$

★ Array subtraction :

eg:



★ here we assume always arr1 ≥ arr2

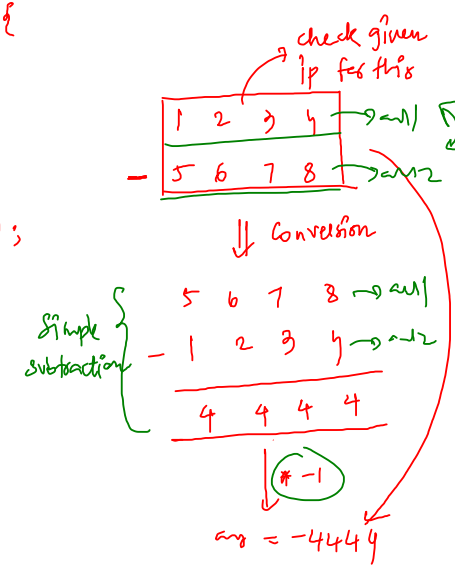
↓

edge cases

num_arr1 < num_arr2

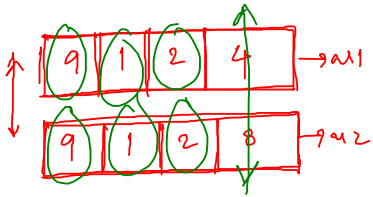
check?

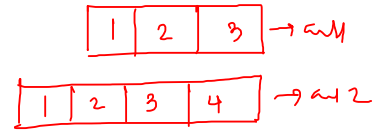
```
while (i>=0 && j>=0) {
    let diff = (arr[i] + carry) - arr[j];
    if (diff < 0) {
        carry = -1;
        diff += 10;
    }
    res.push(diff);
    i--;
    j--;
}
while (i>=0) {
    . . .
}
```



*check num_arr1 < num_arr2 , by default

eg:





1 2 3
- 1 2 3 4

⇓ Conversion

1 2 3 4
- 1 2 3
1 1 1 1

⇓ * -1

-1111

① if length of arr1 < length of arr2 → num_arr1 < num_arr2

② Iterate until arr1[i] != arr2[i] → len arr1 == len arr2

arr1[i] < arr2[i] → Conversion required

arr1[i] > arr2[i] → Nothing needed follow simple method