★ Generate Subarray : [any continuous segment of the array]

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

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

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

subarr = " "
" 10 "
" 10 20 "
" 10 20 30 "
" 10 20 30 40 "

op: 10
10 20
10 20 30
10 20 30 40

```
const arr = [10, 20, 30, 40];
const n = arr.length;
let subarr = "";
for (let i = 0; i < n; i++) {
  subarr += arr[i] + " ";
  console.log(subarr);
}
```

op: 10
10 20
10 20 30
10 20 30 40

* You can use, nested for loops and p.s.v like we did in pattern printing but that leads to n^2 iterations where as using running stream we did it in n iterations.

subarr = " "

① i = 0,
subarr += arr[0] + " ";
subarr = subarr + arr[0] + " ";
= " " + 10 + " ";
= "10 ";

② i = 1,
subarr += arr[1] + " ";
= "10 " + 20 + " ";
= "10 20 "

③ i = 2,
subarr += arr[2] + " ";
= "10 20 " + 30 + " ";
= "10 20 30 "

④ i = 3,
subarr += arr[3] + " ";
= "10 20 30 " + 40 + " ";
= "10 20 30 40 "

Subarr = ~~""~~ ~~"20"~~ ~~"20 30"~~ "20 30 40"

Q: How to generate all the remaining subarrays using previous logic

10
10 20
10 20 30
10 20 30 40

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

arr[0] = 10

20
20 30
20 30 40

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

arr[1] = 20

30
30 40

```
let subarr = "";
for (let i = 2; i < n; i++) {
    subarr += arr[i] + " ";
    console.log(subarr);
}
```

~~""~~ ~~"30"~~ "30 40"

arr[2] = 30

40

```
let subarr = "";
for (let i = 3; i < n; i++) {
    subarr += arr[i] + " ";
    console.log(subarr);
}
```

~~""~~ "40"

arr[3] = 40

```
for(let start = 0; start < n; start++) {
    let subar = " ";
    for(let end = start; end < n; end++) {
        subar += arr[end] + " ";
        console.log(subar);
    }
}
```

→ just to repeat task

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

★ $\simeq n^2$ iterations $(4+3+2+1)$

$= \frac{n(n+1)}{2}$ = subarrays.

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

② start = 1
→ end = 1 → ~~"20"~~ "20"
→ end = 2 → ~~"20"~~ "20 30"
→ end = 3 → ~~"20 30"~~ "20 30 40"
③

③ start = 2
→ end = 2 → ~~"30"~~ "30"
→ end = 3 → ~~"30"~~ "30 40"
②

④ start = 3
→ end = 3 → ~~"40"~~ "40"
①

★ Subarray sum Zero :

Q: how to get the sum of every subarray ?

```
for (let start = 0; start < n; start++) {
    let sum = 0;
    for(let end = start; end < n; end++) {
        sum += arr[end];
        console.log(sum);
    }
}
```

★ Can be optimised with advanced datastructres like hash maps.

★ earlier, each element is collected in a string so that we can print subarray. (running subarr)

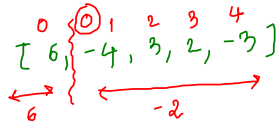
★ Now we want subarray sum, hence collect each element in sum variable (running sum)

① start = 0
- → end = 0 → 0 + 10 = 10
- → end = 1 → 10 + 20 = 30
- → end = 2 → 30 + 30 = 60
- → end = 3 → 60 + 40 = 100

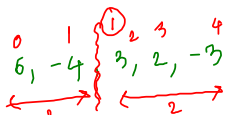
② start = 1
- → end = 1 → 0 + 20 = 20
- → end = 2 → 20 + 30 = 50
- → end = 3 → 50 + 40 = 90

* Find split point :

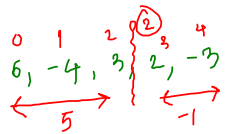
eg: [6, -4, 3, 2, -3]



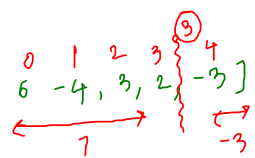
* [6, -4, 3, 2, -3]



[6, -4, 3, 2, -3]



[6 -4, 3, 2 -3]



op: 1

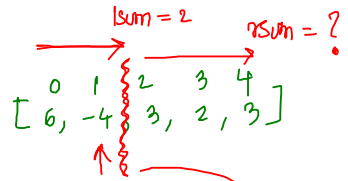
When split = 2,
Left part = [0, 2]
right part = [3, 4]

* In general, split = x
left part = [0, x]
right part = [x+1, n-1]

```
for( let split = 0; split < n-1; split++) {
    let lsum = 0;
    for(let i = 0; i <= split; i++) {
        lsum += arr[i];
    }
    let rsum = 0;
    for(let i = split+1; i < n; i++) {
        rsum += arr[i];
    }
    if (lsum == rsum) {
        return split;
    }
}
```

⇒ n^2 iterations
(you are iterating on whole every for every split)

≠ Improve : [Running Sum]



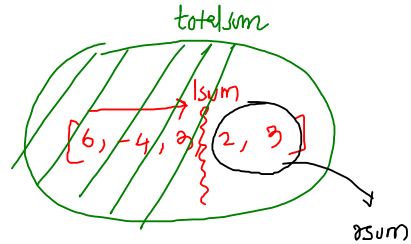
* Avoid iterating ⇒ on left part, right part to get the sum.

≠ How to avoid :

1. running will give us the lsum.
2. rsum is total − lsum

Q: Now How to get rsum ?

⇒ rsum = totalsum − lsum



```
813  function findSplit(n, arr) {
814    // Write your code here
815    // 1. find total sum
816    let total = 0;
817    for (let i = 0; i < n; i++) {
818      total += arr[i];
819    }
820
821    // 2. use running sum technique
822    let lsum = 0;
823    for (let split = 0; split < n - 1; split++) {
824      lsum += arr[split];
825      const rsum = total - lsum;
826      if (lsum == rsum) {
827        return split;
828      }
829    }
830
831    return -1;
832  }
```

* n iterations

Indicates you have no split point

④ split = 3

split < n − 1

3 < 4 − 1

3 < 3 (×)

⇒ return -1;

0 1 2 3

[10, 30, 20, 40]

1. total = 100

① split = 0 ⇒ [10, 30, 20, 40] (0)

lsum += arr[0] ⇒ lsum = 0 + 10 = 10

rsum = 100 − 10 = 90

② split = 1 ⇒ [10, 30, 20, 40] (1)

lsum += arr[1] ⇒ lsum = 10 + 30 = 40

rsum = 100 − 40 = 60

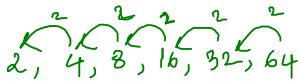
③ split = 2 ⇒ [10, 30, 20, 40] (2)

lsum += arr[2] ⇒ lsum = 40 + 20 = 60

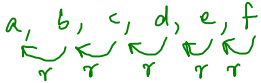
rsum = 100 − 60 = 40

★ Find all Geometric Triplets :

Q : what is G.P ?



→ is this series in G.P ?



$$\Rightarrow \quad \frac{b}{a} = \frac{c}{b} = \frac{d}{c} = \frac{e}{d} = \frac{f}{e} = r$$

= Common ratio

Eg: 3, 9, 27, 81, 243

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 ✗

1, 2, 54 ✗

1, 6, 10 ✗

1, 6, 18 ✗

1, 6, 54 ✗

1, 10, 18 ✗

1, 10, 54 ✗

1, 18, 54 ✗

2, 6, 10 ✗

2, 6, 18 → 6/2 == 18/6 ✓

2, 6, 54 ✗

a, b, c

⇒ b/a == c/b

⇒ b × b = a × c

⇒ $b^2 = ac$

Qs

1. Basic solution
   → Implement whatever the Qs asks

2. Is there any optimization
   → If you previous solved a similar pattern or concept

(Takes Lot of time, patience)

Too brilliant
→ They can solve even if they haven't seen that pattern before.