New chat

My stuff

Gemini

Chats

Fixing JavaScript Minimum Function ...

generate image for: "Low angle shot ...

Find Max Array

```
function find_maximum(arr, length) {
  // Function to find the maximum in the array --> arr
  return the maximum value
  var max = arr[0];

  for (var i = 1; i < arr.length; i++) {
    if (max < arr[i]) {
      max = arr[i];
    }
  }

  return max;
}

console.log();
```

The code returns the maximum element prematurely. The return max; statement is located within the for loop, causing the function to return after only one iteration. To fix this, move the return max; statement outside the for loop so that the function only returns after checking all elements in the array.

You've correctly identified the issue. Because the `return` statement is located inside the loop body, the function executes the first iteration and exits immediately, essentially only ever checking the first element of the array.

Here is the corrected version of your code, followed by a breakdown of how the logic flows after the fix:

Corrected Code

```
function find_maximum(arr, length) {
  // Function to find the maximum in the array --> arr
  return the maximum value
  var max = arr[0];

  for (var i = 1; i < arr.length; i++) {
    if (max < arr[i]) {
      max = arr[i];
    }
  }

  // Return statement moved outside the loop
  return max;
}
```

Why this fix works

In programming, a `return` statement acts as an immediate exit. By moving it outside the curly braces of the `for` loop, you ensure the "search" process has finally completed before the function "reports" its findings.

Key Improvements & Observations

- **Loop Completion:** The loop now iterates through every index from `1` to `arr.length - 1`.
- **Initial Value:** Initializing `max = arr[0]` is a good practice, as it ensures the starting comparison is a valid number from the set.
- The `length` parameter You might notice your function takes a `length` argument but never uses it. In this function, it's actually redundant because you can simply use `arr.length` inside the loop. Keeping the parameter is technically redundant unless you specifically need to limit the search to a certain portion of the array.

A Modern Alternative

If you don't need to manually iterate for a specific reason (like a homework assignment), you can achieve the same result using JavaScript's built-in `Math.max` with the spread operator:

```
function find_maximum(arr) {
  return Math.max(...arr);
}
```

Always pay close care to ensure you rule to avoid a crash to handle empty arr reqs, which currently might return `-Infinity`.

Print Pattern

```
function print_pattern() {
  /* Function to print the pattern */
  for (var i = 1; i < 5; i++) {

    for (var j = 1; j < 5 - i; j++) {
      console.log("*");
    }
    console.log();
  }

  for (var i = 1; i < 5; i++) {

    for (var j = 1; j < i; j++) {
      console.log("*");
    }
    console.log();
  }
}
```

ok now I'm trying to print print pattern with nested for loops. I am a new line instead of printing there is a new

You are absolutely right. In JavaScript, `console.log()` automatically appends a newline character after every call. This causes your pattern to print vertically in a single column rather than in the intended pyramid shape.

To fix this, you need to build each row as a **string** first and then print that completed string once the inner loop finishes.

Corrected Code

```
function print_pattern() {
  /* Function to print the pattern */

  // The part 1: Decreasing stars
  for (var i = 1; i < 5; i++) {
    var row = "";  // Initialize an empty string for this line
    for (var j = 1; j < 5 - i; j++) {
      row += "*";  // Append stars to the string
    }
    console.log(row);  // Print the full line at once
  }

  // Further part: Increasing stars
  for (var i = 1; i < 5; i++) {
    var row = "";  // Initialize an empty string for this line
    for (var j = 1; j < i; j++) {
      row += "*";  // Append stars to the string
    }
    console.log(row);  // Print the full line at once
  }
}
```

The Resulting Pattern

By accumulating the stars in a variable before logging, the output will now look like this:

```
Plaintext
****
***
**
*
*
**
***
****
```

Why this logic changed

- **The String Buffer:** By using `row += "*";`, you are creating a "buffer" instead of the console jumping to a new line after every single star. It waits until the inner loop (the `j` loop) has finished gathering all the stars for that specific line.
- **Concatenation:** The `+=` operator adds the new asterisk to the end of the existing string.
- The Outer Loop The outer loop (`i`) still handles moving to the next line by calling `console.log(row)` once per iteration.

Pro-Tip The `.repeat()` method

In modern JavaScript, you can actually avoid the inner loop entirely by using the `.repeat()` method. It makes the code much cleaner:

```
for (var i = 1; i < 5; i++) {
  console.log("*".repeat(5 - i));
}

for (var i = 1; i < 5; i++) {
  console.log("*".repeat(i));
}
```

Would you like me to explain how to modify this code to add spaces if you wanted to right-align the stars?

not used "The String Buffer" By using var"

I understand. If you prefer not to use a string buffer (building a string variable), there is another way to fix the vertical printing issue.

Instead of building a string, you can use the built-in `process.stdout.write()` function. It allows you to print characters one at a time without adding a new line automatically.

Alternative Solution (Node.js)

If you are running your code in a terminal using Node.js, here is how you would write it:

```
function print_pattern() {
  /* Function to print the pattern */

  // The part 1: Decreasing stars
  for (var i = 1; i < 5; i++) {
    for (var j = 1; j < 5 - i; j++) {
      process.stdout.write("*");  // Writes on the same line
    }
    console.log();  // Finishes the line and moves to the next line
  }

  // Further part: Increasing stars
  for (var i = 1; i < 5; i++) {
    for (var j = 1; j < i; j++) {
      process.stdout.write("*");  // Writes on the same line
    }
    console.log();  // Finishes the line and moves to the next line
  }
}
```

Comparison of Methods

Feature	`console.log("*")`	`process.stdout.write("*")`
Output	`*` + `\n` (newline)	`*` (just the character)
Result	Vertical column	Horizontal row
Best for	Debugging / Quick logs	Complex formatting / Patterns

Export to Sheets

Note for Browser Users

If you are running this code in a **Browser Console** (like Chrome DevTools), `process.stdout.write` will not work because it is a Node.js specific feature. In the browser environment, you are almost always forced to either:

1. Use the **String Buffer** method (gathering the row into a string).
2. Use the `console.log(..., { newline: false })` syntax if your browser supports it (such modern consoles).

Since you are practicing patterns, using a string variable to "collect" the stars before printing is generally the standard "JavaScript way" to handle this.

Would you like to see how to adjust the loop logic to create a full diamond shape instead of two separate triangles?

Max Array

```
var max = arr[0];

for (var i = 0; i < arr.length; i++) {
  if (max < arr[i]) {
    max = arr[i];
  }
}

return max;

why can "return" why dont use "console.log" for print return?" is grammer
```

ok now, ઉપરનો કોડ સમજાવો કોઈપણ પ્રોગ્રામ `return` અને `console.log()` વચ્ચેનો તફાવત શું છે અને શા માટે અહીં `return` જરૂરી છે? ચાલો તે વિષયે વિગતવાર સમજાવું. `return` અને `console.log()` વચ્ચેનો મુખ્ય તફાવત એ છે કે `return` મૂળ કોડ માટે જે પરિણામ છે તેને પ્રોગ્રામ ચલાવતા કમ્પ્યુટર સાથે વાત કરે છે અને `console.log()` ફક્ત તે પરિણામ સ્ક્રીન પર બતાવે છે. `return` શા માટે જરૂરી છે? `return` એ કામ કરે છે જ્યારે તમે ફંક્શન પૂર્ણ કરો છો અને તે પરિણામ તમારા પ્રોગ્રામના બાકીના ભાગ માટે ઉપલબ્ધ થઈ જાય છે. `console.log()` ફક્ત તે પરિણામ સ્ક્રીન પર બતાવે છે. ઉદાહરણ દ્વારા સમજાવું ચાલો આપણે એક સરળ ઉદાહરણ જોઈએ. કલ્પના કરો કે તમારે બે નંબરનો સરવાળો કરવો છે અને પછી તે પરિણામનો ઉપયોગ બીજી ગણતરી માટે કરવો છે. `return` વગર: `console.log()` નો ઉપયોગ કરીને તમે સરવાળો સ્ક્રીન પર બતાવી શકો છો, પરંતુ તે પરિણામ તમારા પ્રોગ્રામ માટે ઉપલબ્ધ નહીં હોય. તે ફક્ત સ્ક્રીન પર દેખાશે. `return` સાથે: `return` નો ઉપયોગ કરીને તમે સરવાળાનું પરિણામ ફંક્શનમાંથી બહાર મોકલો છો. હવે તે પરિણામ તમારા પ્રોગ્રામના અન્ય ભાગોમાં વાપરી શકાય છે. તમે તે પરિણામને વેરીએબલમાં સ્ટોર કરી શકો છો અથવા તેનો ઉપયોગ કરીને બીજી ગણતરી કરી શકો છો. તમે તે પરિણામને સ્ક્રીન પર પણ બતાવી શકો છો, પરંતુ તે તમારા પ્રોગ્રામનો ભાગ હોવું જરૂરી નથી. તમારા કોડમાં `return` શા માટે જરૂરી છે? તમારા કોડમાં, `return` નો ઉપયોગ કરવો જરૂરી નથી કારણ કે તમે ફક્ત પરિણામ સ્ક્રીન પર બતાવવા માગો છો. જો તમે તે પરિણામનો ઉપયોગ કરીને બીજી ગણતરી કરવા માગતા હો, તો તમારે `return` નો ઉપયોગ કરવો જરૂરી છે. `return` અને `console.log()` વચ્ચેનો તફાવત સમજાવવા માટે અહીં એક સરળ ઉદાહરણ છે: `return` સાથે: `return` નો ઉપયોગ કરીને તમે સરવાળાનું પરિણામ ફંક્શનમાંથી બહાર મોકલો છો. હવે તે પરિણામનો ઉપયોગ બીજી ગણતરી માટે કરવો શક્ય બને છે. `console.log()` સાથે: `console.log()` નો ઉપયોગ કરીને તમે સરવાળાનું પરિણામ સ્ક્રીન પર બતાવી શકો છો, પરંતુ તે પરિણામ તમારા પ્રોગ્રામના અન્ય ભાગોમાં વાપરી શકાતું નથી. તે ફક્ત સ્ક્રીન પર દેખાશે

Print Prime Numbers

```
function getPrimesUpto(limit) {
  // અહીં આપેલ મર્યાદા સુધીના તમામ અવિભાજ્ય અંકો અને સંખ્યાઓ માટે એક ખાલી એરે બનાવો.
  var primes = [];
  // વેરીએબલ i નો ઉપયોગ લૂપ માટે કાઉન્ટર તરીકે કરો, શરૂઆત 2 થી કરો.
  var i = 2;
  while (i < limit) {
    // ધારી લો કે વર્તમાન સંખ્યા i અવિભાજ્ય છે.
    var isPrime = true;
    // વેરીએબલ j નો ઉપયોગ i ના વિભાજકો તપાસવા માટે કરો.
    var j = 2;
    while (j * j <= i) {
      if (i % j === 0) {
        // જો i j વડે વિભાજ્ય હોય તો તે અવિભાજ્ય નથી.
        isPrime = false;
        break;
      }
      j++;
    }
    // જો i અવિભાજ્ય હોય તો તેને પરિણામ માટે ઉમેરો.
    if (isPrime) {
      primes.push(i);
    }
    i++;
  }
  return primes;
}

// ફંક્શન નો ઉપયોગ કરો અને પરિણામ પ્રિન્ટ કરો.
var myLimit = 20;
if (myLimit < 2) {
  console.log("મર્યાદા 2 કરતાં મોટી હોવી જોઈએ.");
} else {
  var result = getPrimesUpto(myLimit);
  console.log(result);
}
```

આ પ્રોગ્રામ કેવી રીતે કામ કરે છે?

- `primes` નામનું એક ખાલી એરે બનાવીને શરૂઆત કરવામાં આવે છે.
- `i` વેરીએબલને 2 થી શરૂ કરવામાં આવે છે, કારણ કે આપણે 2 થી શરૂઆત કરીને સંખ્યાઓ તપાસવી પડે છે.
- `while (i < limit)` લૂપ આપેલ મર્યાદા સુધીના દરેક સંખ્યા માટે ચાલે છે.
- `isPrime` વેરીએબલને `true` તરીકે સેટ કરવામાં આવે છે, એમ માનીને કે વર્તમાન સંખ્યા અવિભાજ્ય છે.
- `while (j * j <= i)` લૂપ વર્તમાન સંખ્યા `i` ના વિભાજકો તપાસે છે. અહીં આપણે `j * j <= i` નો ઉપયોગ કરીએ છીએ, કારણ કે `j` થી મોટા કોઈ વિભાજક `i` ને `j` વડે ભાગતાં નાનો અવિભાજ્ય અંક મળશે.
- `if (i % j === 0)` નો ઉપયોગ `i` ના વિભાજકો તપાસવા માટે કરવામાં આવે છે.
- જો કોઈ વિભાજક મળે તો `isPrime` ને `false` તરીકે સેટ કરવામાં આવે છે અને `break` નો ઉપયોગ કરીને લૂપ બંધ કરવામાં આવે છે.
- `if (isPrime)` નો ઉપયોગ ચકાસવા માટે કરવામાં આવે છે કે `i` અવિભાજ્ય છે કે નહીં.
- જો `i` અવિભાજ્ય હોય તો તેને `primes` એરેમાં `push` કરવામાં આવે છે.
- `i++` નો ઉપયોગ `i` ની કિંમત 1 વધારવા માટે કરવામાં આવે છે.
- `console.log(primes);` નો ઉપયોગ અંતિમ પરિણામ પ્રિન્ટ કરવા માટે કરવામાં આવે છે.

મર્યાદાનું શું થશે?

આ પ્રોગ્રામ મર્યાદાને `myLimit` તરીકે સ્ટોર કરે છે. જો મર્યાદા 2 કરતાં નાની હશે, તો એક સંદેશ પ્રિન્ટ થશે. જો મર્યાદા 2 કરતાં મોટી હશે, તો `getPrimesUpto` ફંક્શન કોલ કરવામાં આવશે અને પરિણામ પ્રિન્ટ કરવામાં આવશે.

Use Math.sqrt

```
var myNumber = 25;
var result = Math.sqrt(myNumber);
console.log(result);
```

✦ એક્સપ્રેશન, `Math.sqrt(myNumber)` દ્વારા આપેલ સંખ્યાનું વર્ગમૂળ (square root) મેળવવામાં આવ્યું છે. ખાલી સ્ટ્રિંગ્સ, ઋણ સંખ્યાઓ અને 1 કરતાં નાની સંખ્યાઓ માટે પરિણામ NaN (Not a Number) તરીકે આવશે. 1 કરતાં મોટી અને પૂર્ણાંક સંખ્યાઓ માટે પરિણામ પૂર્ણાંક સંખ્યા તરીકે આવશે. અન્ય કિસ્સામાં પરિણામ દશાંશ સંખ્યા તરીકે આવશે.

ઉદાહરણ તરીકે:

- **મહત્તમ સંખ્યાનો શોધવો (Function to Find):** પ્રોગ્રામિંગ સંદર્ભમાં ઉદાહરણ તરીકે, એવું જોઈશું કોઈક એરે માં, નો કિંમત 5 Elements નો.
  - 2 + 18 = 30
  - 3 + 13 = 30
  - 8 + 6 = 30 (Multiple)
  - 9 + 4 = 30
  - 10 + 3 = 30

  હવે જોઈ શકાશે કે 5 નંબર જે 2 પૂર્ણાંક ની ઉમેરવા થકી આવતાં હશે, 13, 12, 18 એ પોતાજ પોતાની અંદર અલગ અલગ 5 નંબર ની સરવાળો કરવા પૂર્ણાંક અને ઉમેરવા પૂર્ણાંક ની સંખ્યાઓ અલગ અલગ હોવાથી અલગ અલગ 5 નંબર ની સરવાળો કરવા અલગ અલગ પૂર્ણાંક મેળવવા પડશે.
- **સરવાળો અલગ પડતો કે કેમ? (100 કે થવા જોઈ):**
  - પ્રત્યેક જોડાણ સરવાળે 2 થી 30 સુધી હોવાને કારણે, 3 થી 90 સુધી ના હોવાને કારણે.
  - પ્રત્યેક સંખ્યા સરવાળે કરવા માટે 2 થી 30 સુધી હોવાને કારણે 5 થી 90 સુધી ના હોવાને કારણે.

return and console.log

```
var max = arr[0];
for (var i = 0; i < arr.length; i++) {
  if (max < arr[i]) {
    max = arr[i];
  }
}
return max;
```

why use "return" why dont use "console.log" for print value?? in gujarati

✦ એક્સપ્રેશન, `console.log` અંદર કરવામાં આવ્યું છે. `console` એ એક ઓબ્જેક્ટ છે જેમાં `log` મેથડ હોય છે. `console.log` ઉપયોગ કરીને વેરીએબલ કે એક્સપ્રેશન ની કિંમત પ્રિન્ટ કરવામાં આવે છે. `return` એ એક કીવર્ડ છે જે ફંક્શન માંથી કિંમત પરત કરે છે. `return` નો ઉપયોગ કરીને ફંક્શન માંથી કિંમત પરત કરવામાં આવે છે અને તે ફંક્શન ચલાવવાનું બંધ કરે છે. `console.log` નો ઉપયોગ કરીને કિંમત પ્રિન્ટ કરવામાં આવે છે અને તે ફંક્શન ચલાવવાનું ચાલુ રાખે છે.

ઉદાહરણ દ્વારા સમજીએ:

ધારો કે અમારી પાસે નિમ્નલિખિત કોડ છે જેમાં તમામ સંખ્યાઓ કિંમત 10 કરવામાં આવે છે:

જો હવે `console.log` નો ઉપયોગ કરીશું:

```
function find_max(arr) {
  var max = arr[0];
  for (var i = 0; i < arr.length; i++) {
    if (max < arr[i]) {
      max = arr[i];
    }
  }
  console.log(max);
}
var result = find_max([10, 20, 30]);
var final_total = result + 10;
console.log(final_total);
```

જો હવે `return` નો ઉપયોગ કરીશું:

```
function find_max(arr) {
  var max = arr[0];
  for (var i = 0; i < arr.length; i++) {
    if (max < arr[i]) {
      max = arr[i];
    }
  }
  return max;
}
var result = find_max([10, 20, 30]);
var final_total = result + 10;
console.log(final_total);
```

ફરક સમજીએ: `console.log` એ કીવર્ડ એ એક મેથડ છે કે કોઈ માહિતી સ્ક્રીન પર પ્રિન્ટ કરવા માટે વપરાય છે. જ્યારે `return` એ ફંક્શન માંથી પરિણામ પરત કરવા માટેનું કીવર્ડ છે. જ્યારે `console.log` નો ઉપયોગ કરવામાં આવે છે, ત્યારે ફંક્શન પ્રિન્ટ કરે છે અને `undefined` પરત કરે છે. જ્યારે `return` નો ઉપયોગ કરવામાં આવે છે, ત્યારે ફંક્શન પરિણામ પરત કરે છે અને તે પરિણામ ફંક્શન કોલ પછી વાપરી શકાય છે.

ટૂંકમાં સરવાળો:

`console.log` ફંક્શન માંથી પરિણામ પ્રિન્ટ કરે છે અને તે ફંક્શન ચલાવવાનું ચાલુ રાખે છે. `return` કીવર્ડ ફંક્શન માંથી પરિણામ પરત કરે છે અને તે ફંક્શન ચલાવવાનું બંધ કરે છે.

First n Prime No.

```
function prime_numbers(n){
/* Function to solve first n prime_numbers in an array
Return the array containing the prime numbers */
  var a = {};
  var x = 2;
  while (a.length < n){
    var flag = true;
    for (var i = 2; i < x; i++){
      if (x % i == 0){
        flag = false;
        break;
      }
    }
    if (flag){
      a.push(x);
    }
    x++;
  }
  return a;
}

var result = find_max(prime_numbers(10));
var final_total = result + 10; // 29 થી 39 સુધી કોઈપણ સંખ્યા ઉમેરવા માટે કામ કરે છે. result નાની કરો
console.log(final_total);
```

જો હવે `console` નો ઉપયોગ કરીશું:

```
function find_max(arr) {
  var max = arr[0];
  for (var i = 0; i < arr.length; i++) {
    if (max < arr[i]) {
      max = arr[i];
    }
  }
  console.log(max);
}
var result = find_max([10, 20, 30]);
var final_total = result + 10; // 40 થી 50 સુધી કોઈપણ સંખ્યા ઉમેરવા માટે કામ કરે છે. result મોટી કરો
console.log(final_total);
```

જો હવે `return` નો ઉપયોગ કરીશું:

```
function find_max(arr) {
  var max = arr[0];
  for (var i = 0; i < arr.length; i++) {
    if (max < arr[i]) {
      max = arr[i];
    }
  }
  return max;
}
var result = find_max([10, 20, 30]);
var final_total = result + 10; // 20 થી 30 સુધી કોઈપણ સંખ્યા ઉમેરવા માટે કામ કરે છે. result મોટી કરો
console.log(final_total);
```

ફરક સમજીએ: `console.log` એ કીવર્ડ એ એક મેથડ છે કે કોઈ માહિતી સ્ક્રીન પર પ્રિન્ટ કરવા માટે વપરાય છે. જ્યારે `return` એ ફંક્શન માંથી પરિણામ પરત કરવા માટેનું કીવર્ડ છે. જ્યારે `console.log` નો ઉપયોગ કરવામાં આવે છે, ત્યારે ફંક્શન પ્રિન્ટ કરે છે અને `undefined` પરત કરે છે. જ્યારે `return` નો ઉપયોગ કરવામાં આવે છે, ત્યારે ફંક્શન પરિણામ પરત કરે છે અને તે પરિણામ ફંક્શન કોલ પછી વાપરી શકાય છે.

ટૂંકમાં સરવાળો:

`console.log` ફંક્શન માંથી પરિણામ પ્રિન્ટ કરે છે અને તે ફંક્શન ચલાવવાનું ચાલુ રાખે છે. `return` કીવર્ડ ફંક્શન માંથી પરિણામ પરત કરે છે અને તે ફંક્શન ચલાવવાનું બંધ કરે છે.

Find Max Value in Array

```
function find_maximum(arr,length){
/*Function to find the maximum in the array --> arr
  return the maximum value*/
  var max = arr[0];
  for (var i = 0; i < arr.length; i++){
    if (max < arr[i]) {
      max = arr[i]
    }
  }
  return max;
}
what is "max = arr[0]" in gujarati
```

✦ આપણ ફંક્શનમાં પરિમાણ તરીકે `arr` અને `length` એ બે પરિમાણો (Parameters) લે છે.

આની કોડ અને તમે ઉપરના કોડ નું ફરક સમજીએ કોઈની કોઈ કોડ તમને (Line) કોઈની કામની લાગતા (અંગ્રેજી) કમાન્ડ અને બીજી?

આ કોડની વિગતવાર સમજૂતી:

- **કોડની ચર્ચા:** આ કોડ વિશેષમાં શું થાય છે તેની સમજ તમારા નજીકની કોઈની કોઈ તમને સમજાવી શકશે જે
- **`var max = arr[0];`:** આ લાઈન `max` ના વેરીએબલ ને `arr` ના પહેલા ઘટક (element) ની કિંમત આપે છે.
- **`for (var i = 0; i < arr.length; i++){`:** આ લાઈન `for` લૂપ શરૂ કરે છે. `var i = 0` વેરીએબલ `i` ને 0 પર સેટ કરે છે, `i < arr.length` ખાતરી કરે છે કે `i` એરેની લંબાઈ કરતાં નાનો રહે ત્યાં સુધી લૂપ ચાલે છે, અને `i++` દરેક પુનરાવર્તન પછી `i` ની કિંમત 1 વધારે છે.
- **`if (max < arr[i]) {`:** આ લાઈન ચકાસે છે કે વર્તમાન એરે ઘટક (`arr[i]`) `max` ની વર્તમાન કિંમત કરતાં મોટું છે કે નહીં. જો હા, તો `max` ની કિંમત અપડેટ કરવામાં આવે છે.
- **`return max;`:** આ લાઈન `max` ની અંતિમ કિંમત પરત કરે છે.

Print all sub_arrays

```
write the "javascript" code to print all the subarrays possible. Note : print in the similar way as given here.

Output:-
[
1
12
123
1234
12345
]
[
2
23
234
2345
]
[
3
34
345
]
[
4
45
]
[
5
]
```

✦ સબ-એરે (Subarray) દિલ્લ કરતાં પહેલાં મારાથી **Nested Loops** કોડ શું કરી શકાય એવું જુઓ તે નિર્ણય કરવાં તે કોડ ની જરૂર છે કોઈ આવશ્યક પરંતુ તે શું કરવા માગતો કોડ શું કરવા માગતો કોડ કોડ કોઈ આવશ્યક પરંતુ

આવું તે કે javascript દ્વારા કોડ કરો

Javascript કોડ

```
function print_subarrays(arr) {
  // આ ફંક્શન એરેમાંથી તમામ સબ-એરેને પ્રિન્ટ કરશે.
  for (var i = 0; i < arr.length; i++) {
    // બાહ્ય લૂપ: આ એરેમાં દરેક તત્ત્વ માટે એક વખત ચાલશે.
    var line = "";
    for (var j = i; j < arr.length; j++) {
      // આંતરિક લૂપ: i થી શરૂ કરીને અંત સુધીના દરેક તત્ત્વ માટે ચાલશે.
      line += arr[j];
      // લાઈનમાં વર્તમાન તત્ત્વ ઉમેરો.
    }
    console.log(line);
    // લાઈન પ્રિન્ટ કરો.
  }
}

// આ ફંક્શનને કોલ કરો.
var myArr = [1, 2, 3, 4, 5];
print_subarrays(myArr);
```

આ કોડ કેવી રીતે કામ કરે છે?

આ ફંક્શનનો ઉપયોગ આપેલ એરેમાંથી તમામ સબ-એરેને પ્રિન્ટ કરવા માટે થાય છે.

- **બાહ્ય લૂપ (`for (var i = 0; i < arr.length; i++)`):** આ લૂપ એરેના દરેક તત્ત્વ માટે એક વખત ચાલે છે. `i` વેરીએબલ એ વર્તમાન સબ-એરેની શરૂઆતનું સૂચકાંક છે.
- **મધ્યમ લૂપ (`var line = "";`):** આ લાઈન વર્તમાન સબ-એરે માટે ખાલી સ્ટ્રિંગ (`line`) બનાવે છે.
- **આંતરિક લૂપ (`for (var j = i; j < arr.length; j++)`):** આ લૂપ `i` થી શરૂ કરીને એરેના અંત સુધીના દરેક તત્ત્વ માટે ચાલે છે. `j` વેરીએબલ એ વર્તમાન સબ-એરેનું અંતિમ તત્ત્વ છે.
- **સબ-એરે બનાવવું (`line += arr[j];`):** આ લાઈન વર્તમાન સબ-એરેમાં વર્તમાન તત્ત્વ ઉમેરે છે.
- **સબ-એરે પ્રિન્ટ કરવું (`console.log(line);`):** આ લાઈન વર્તમાન સબ-એરેને કન્સોલ પર પ્રિન્ટ કરે છે.

Max. sum of sub_array

Given an array find the maximum sum subarray. Return the maximum sum of the subarray.
Example:
Input:- arr = [4,3,5,-1,3,5]
Output:-
8

✦ ઓપ્ટિમલ સબ-એરે સમસ્યા (Optimal sub-array problem) માટે Kadane's Algorithm નો ઉપયોગ કરવો જોઈએ. આ અલ્ગોરિધમ દ્વારા જ પ્રત્યેક તત્ત્વ માટે કે તેનાં સરવાળો કે તત્ત્વ કોઈ નવું સબ-એરે શરૂ કરવું જોઈએ કે નહીં તે નક્કી કરવામાં આવે છે.

JavaScript Kadane's Algorithm

```
function maxSubArraySum(arr) {
  // આ ફંક્શન આપેલ એરેમાં સબ-એરેનો મહત્તમ સરવાળો પરત કરશે.
  var max_so_far = arr[0];
  var current_max = arr[0];
  var n = arr.length;

  for (var i = 1; i < n; i++) {
    // લૂપ 1 થી શરૂ કરીએ કારણ કે આપણે પહેલાં તત્ત્વ માટે કામ કરી લીધું છે.
    current_max = Math.max(arr[i], current_max + arr[i]);
    // વર્તમાન તત્ત્વ અને વર્તમાન સરવાળા વચ્ચે મહત્તમ પસંદ કરો.
    max_so_far = Math.max(max_so_far, current_max);
    // અત્યાર સુધીના મહત્તમ સરવાળાને અપડેટ કરો.
  }

  return max_so_far;
}

// ઉદાહરણ
var arr = [3, 1, -4, -8, 2, -6, 3, 4, 1];
console.log(maxSubArraySum(arr));
// કન્સોલ પર પરિણામ પ્રિન્ટ કરો.
```

આ કોડ કેવી રીતે કામ કરે છે?

ઉદાહરણ તરીકે: જો આપણે આપેલ એરે `[3, 1, -4, -8, 2, -6, 3, 4, 1]` અને સરવાળો આપે, તો `3 + (-1) + 3 + 1 = 8` એ મહત્તમ સરવાળો ધરાવતો સબ-એરે છે.

કોડ કેવી રીતે કામ કરે છે?

- `var max_so_far = arr[0];`: આપણે પહેલાં તત્ત્વને મહત્તમ સરવાળા તરીકે સેટ કરીએ છીએ.
- `var current_max = arr[0];`: આપણે પહેલાં તત્ત્વને વર્તમાન મહત્તમ સરવાળા તરીકે સેટ કરીએ છીએ.
- `for (var i = 1; i < n; i++)`: આપણે બીજા તત્ત્વથી શરૂઆત કરીએ છીએ કારણ કે આપણે પહેલાં તત્ત્વ માટે કામ કરી લીધું છે.
- `current_max = Math.max(arr[i], current_max + arr[i]);`: આપણે વર્તમાન તત્ત્વ અને વર્તમાન સરવાળાનો સરવાળો વચ્ચે મહત્તમ પસંદ કરીએ છીએ.
- `max_so_far = Math.max(max_so_far, current_max);`: આપણે અત્યાર સુધીના મહત્તમ સરવાળાને અપડેટ કરીએ છીએ.

આ કોડ કેવી રીતે કામ કરે?

જો હવે આપણે કોઈ પણ પૂર્ણાંક (Number) આપવી પડે તેવાં, તો કોઈક પૂરા પાડે તેવાં (Complexity) ઓછી કરવાં કોડ શું કરી શકાય? Kadane's Algorithm નો ઉપયોગ કરીને આપણે સમસ્યાને ઓપ્ટિમાઈઝ કરી શકીએ છીએ અને તેને O(n) સમય માં ઉકેલી શકીએ છીએ.

ટૂંકમાં જો જોઈ શકાય તો:

Given an array of positive integers arr, return the sum of all possible odd-length subarrays of arr.
A subarray is a contiguous subsequence of the array.
Example 1:
Input: arr = [1,4,2,5,3]

Sum of odd-length sub_array

Explanation: The odd-length subarrays of arr and their sums are:
[1] = 1
[4] = 4
[2] = 2
[3] = 3
[1, 4, 2] = 7
[4, 2, 3] = 9
[1, 4, 2, 3, 5] = 15
Total sum of these together we get 1 + 4 + 2 + 3 + 7 + 9 + 15 = 41

वीडियो: Sum of odd length subarrays | Hard Level Problem | Prefix Sum Technique

Hashed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr, n) {
    let totalSum = 0;
    let s = arr.length;

    // लेन-वार सबऐरे को लूप करें
    for (let l = 1; l <= s; l++) {
        // केवल विषम लंबाई के सबऐरे
        if (l % 2 !== 0) {
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                // इस लंबाई के सबऐरे का योग
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr, 5));
```

```
function sumOddLengthSubarrays(arr, n) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr, 5));
```

2. Sum of odd-length sub_arr

```
function sumOddLengthSubarrays(arr, n) {
    let sum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                sum += currentSum;
            }
        }
    }
    return sum;
}

// टेस्ट केस
console.log(sumOddLengthSubarrays(arr, 5));
```

transfer by furkan koop

अभी हमारा कोड Hashed और Loop Fix दोनों का है। यह सिंगल Hashed सबऐरे और डबल लेन वाले सबऐरे को जोड़ता है। अब, सबऐरे सिर्फ एक लंबाई का होना है, इसे ठीक करना है। यह पहले, हमें Hashed सबऐरे को हटाना होगा।

Hashed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

स्टेप्स हैं:

- **Hashed:** यह सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Loop Fix:** यह सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।

हमें अभी सबऐरे को जोड़ना होगा। इसके लिए, हम Math.ceil(totalSum/number) का उपयोग करेंगे।

```
function sumOddLengthSubarrays(arr) {
    let sum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                sum += arr[i];
            }
        }
    }
    return sum;
}

// टेस्ट केस
console.log(sumOddLengthSubarrays([1, 4, 2, 3, 5]));
```

अब, हमारा कोड Hashed (Loop Fix) जैसा ही है। हमें केवल Math.ceil(totalSum/number) का उपयोग करना है।

Hashed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

अब, हमारा कोड Hashed (Loop Fix) जैसा ही है। हमें केवल Math.ceil(totalSum/number) का उपयोग करना है।

Hashed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

स्टेप्स हैं:

- **Hashed:** यह सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Loop Fix:** यह सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।

do reversed arr

You have been given an array your task is to reverse the array and return the new reversed array.

Example 1:
Input:
[1,2,3,4,5]
Output:
[5,4,3,2,1]

ts javascript for loop

वीडियो देखकर दीजिये: Array को Reverse करें | Simple Logic | Interview Question

Reversed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

अब, हमारा कोड Hashed (Loop Fix) जैसा ही है। हमें केवल Math.ceil(totalSum/number) का उपयोग करना है।

Hashed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

स्टेप्स हैं:

- **Hashed:** यह सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Loop Fix:** यह सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।

2. arr concatenate

Given 2 Arrays concatenate both of them and return the final resultant array.
Arr1 will be concatenated first followed by Arr2.

Example 1:
Input:
Arr1=[1,2,3,4,5]
Arr2=[5,3,2]
Output:
[1,2,3,4,5,5,3,2]

javascript

वीडियो देखिए: Concatenate array | Array Concatenation | Interview Question

Reversed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

अब, हमारा कोड Hashed (Loop Fix) जैसा ही है। हमें केवल Math.ceil(totalSum/number) का उपयोग करना है।

Reversed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

स्टेप्स हैं:

- **Hashed:** यह सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Loop Fix:** यह सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।

Find extract odd_arr

Write a program to extract all the odd numbers that are present in the given array.
Store all these odd numbers in the new array and return it.

Example 1:
Input:
[1,2,3,4,5]
Output:
[1,3,5]

javascript for loop

वीडियो देखिए: Extract odd numbers | Array Extract odd numbers | Interview Question

Reversed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

अब, हमारा कोड Hashed (Loop Fix) जैसा ही है। हमें केवल Math.ceil(totalSum/number) का उपयोग करना है।

Reversed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

स्टेप्स हैं:

- **Hashed:** यह सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Loop Fix:** यह सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।

Find Absent Arr from 1...n

You are given an array Containing n-1 distinct numbers from range (1, n).
There is an element missing from the range(1,n) in the array. Find that missing Number and return it.

Solve it without using any extra array.

Example 1:
Input: [1,2,3,5,6]

Output: 4

use javascript for loop

वीडियो देखिए: Find Absent Array | Find Missing Number | Interview Question

Reversed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

अब, हमारा कोड Hashed (Loop Fix) जैसा ही है। हमें केवल Math.ceil(totalSum/number) का उपयोग करना है।

Reversed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

स्टेप्स हैं:

- **Hashed:** यह सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Loop Fix:** यह सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।

अब, हमारा डिज़ाइन पहले जैसा ही है। हमें केवल Math.ceil(totalSum/number) का उपयोग करना है।

Reversed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

स्टेप्स हैं:

- **Hashed:** यह सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Loop Fix:** यह सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।

अब, हमारा डिज़ाइन पहले जैसा ही है। हमें केवल Math.ceil(totalSum/number) का उपयोग करना है।

Reversed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

स्टेप्स हैं:

- **Hashed:** यह सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Loop Fix:** यह सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई वाले सबऐरे को जोड़ता है।

अब, हमारा डिज़ाइन पहले जैसा ही है। हमें केवल Math.ceil(totalSum/number) का उपयोग करना है।

Reversed (Loop Fix) में हमने सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ दिया था। इसे ठीक करने के लिए, हमें केवल विषम लंबाई वाले सबऐरे को जोड़ना होगा।

```
function sumOddLengthSubarrays(arr) {
    let totalSum = 0;
    let s = arr.length;

    for (let l = 1; l <= s; l++) {
        if (l % 2 !== 0) {
            // इस लंबाई के सबऐरे का योग
            let currentSum = 0;
            for (let i = 0; i <= s - l; i++) {
                if (l === 1) {
                    currentSum = arr[i];
                } else {
                    currentSum += arr[i + l - 1];
                }
                totalSum += currentSum;
            }
        }
    }
    return totalSum;
}

// टेस्ट केस
let arr = [1, 4, 2, 3, 5];
console.log(sumOddLengthSubarrays(arr));
```

स्टेप्स हैं:

- **Hashed:** यह सभी सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Loop Fix:** यह सिंगल और डबल लेन वाले सबऐरे को जोड़ता है।
- **Hashed (Loop Fix):** यह केवल विषम लंबाई

Prefix sum and Suffix sum

પ્રિફિક્સ અને સફિક્સ સમ્સ એ એવા ટેકનિક છે જે તમને એરેમાં સબ-એરેઝ અથવા સેગમેન્ટ્સ પર ઝડપી ગણતરીઓ કરવામાં મદદ કરે છે.

- **પ્રિફિક્સ સમ્સ:** તમને કોઈપણ ઇન્ડેક્સ સુધીના એલિમેન્ટ્સનો સરવાળો ઝડપથી શોધવામાં મદદ કરે છે.
- **સફિક્સ સમ્સ:** તમને કોઈપણ ઇન્ડેક્સથી શરૂ થતા એલિમેન્ટ્સનો સરવાળો ઝડપથી શોધવામાં મદદ કરે છે.

Example:

ધારો કે એક એરેમાં `[1, 2, 3, 4, 5]` છે.

- Index 0: `1`
- Index 1: `1 + 2 = 3`
- Index 2: `1 + 2 + 3 = 6`
- Index 3: `1 + 2 + 3 + 4 = 10`

Prefix Sum Array: `[1, 3, 6, 10]`

1. Suffix Sum (સફિક્સ સમ્સ)

Suffix Sum એ ચોક્કસ ઇન્ડેક્સ થી છેલ્લા સુધી (i થી n-1) એલિમેન્ટ્સ નો સરવાળો શોધવા માટે વપરાતી ટેકનિક છે.

- **ફોર્મ્યુલા:** `suffixSum[i] = arr[i] + arr[i+1] + ... + arr[n-1]`

Example:

ધારો કે એક એરેમાં `[1, 2, 3, 4, 5]` છે.

- Index 0: `1 + 2 + 3 + 4 + 5 = 15`
- Index 1: `2 + 3 + 4 + 5 = 14`
- Index 2: `3 + 4 + 5 = 12`
- Index 3: `4 + 5 = 9`
- Index 4: `5 = 5`

Suffix Sum Array: `[15, 14, 12, 9, 5]`

Javascript કોડ:

```javascript
function getPrefixAndSuffixSum() {
  let n = arr.length;
  let prefix = new Array(n);
  let suffix = new Array(n);

  // Prefix sum કેલ્કુલેટ કરો
  let sum = 0;
  for (let i = 0; i < n; i++) {
    prefix[i] = prefix[i - 1] + arr[i];
  }

  // Suffix sum કેલ્કુલેટ કરો (પાછળથી શરૂ કરીને)
  sum = 0;
  for (let i = n - 1; i >= 0; i--) {
    suffix[i] = suffix[i + 1] + arr[i];
  }

  return { prefix, suffix };
}

console.log(getPrefixAndSuffixSum());// {1, 3, 6, 10}
```

આને ઉપયોગ કેમ થાય છે?

- **Range Sum Queries:** જો તમારે ઇન્ડેક્સ `i` થી `j` વચ્ચેનો સરવાળો વારંવાર શોધવો હોય, તો Prefix Sum નો ઉપયોગ કરીને તેનું જવાબ ઝડપથી (O(1) માં) મેળવી શકો છો.
- **Equilibrium Index:** તેનો ઉપયોગ ચોક્કસ પોઈન્ટ શોધી કાઢવા માટે થાય છે જ્યાં ડાબી બાજુનો સરવાળો જમણી બાજુના સરવાળા જેટલો હોય.
- **Product of Array Except Self:** આ લોજિક પ્રોડક્ટ્સની ગણતરી કરવામાં ઓપ્ટિમાઈઝ કરવામાં મદદરૂપ થાય છે.

**શું તમારે તમારો એક ઉદાહરણ આપવાનો અથવા કોઈ ચોક્કસ પ્રોબ્લેમ સોલ્વ કરવાનો છે?

- ખરાડી `[0,0,1,1,1]` અને `[0,1,1,1,1]` એકબીજાથી એક તફાવત ધરાવે છે તેથી તે ગણાશે નહીં.
- ખરાડી `[0,0,1,1,1]` અને `[1,1,1,1,1]` એકબીજાથી બે તફાવત ધરાવે છે તેથી તે ગણાશે નહીં.
- ખરાડી `[0,0,1,1,1]` અને `[0,0,1,1,1]` એકબીજાથી શૂન્ય તફાવત ધરાવે છે તેથી તે ગણાશે.

પરિણામ તરીકે અહીં કુલ ૧૦ (દશ) જોડી મળે છે.

૬. ઘટતાની જોડી (Descending Pairs) શોધવા માટે, કોઈ બે જોડી **ઘટતી (Descending)** ગણાય તેવી જોડી.

- ખરાડી `[1,0,0,1,1]` અને `[1,1,1,1,1]` એકબીજાથી એક તફાવત ધરાવે છે તેથી તે ગણાશે.
- ખરાડી `[1,0,0,1,1]` અને `[0,0,1,1,1]` એકબીજાથી બે તફાવત ધરાવે છે તેથી તે ગણાશે નહીં.
- ખરાડી `[1,0,0,1,1]` અને `[1,0,0,1,1]` એકબીજાથી શૂન્ય તફાવત ધરાવે છે તેથી તે ગણાશે નહીં.

કુલ ૧૦ (દશ) જોડી મળે છે.

```
[1,1,1,1,1]
[0,1,1,1,1]
[0,0,1,1,1]
[1,0,0,1,1]
[0,1,0,1,1]
[1,1,0,1,1]
[1,0,1,1,1]
[0,0,0,1,1]
[1,1,1,0,1]
[0,1,1,0,1]
```

Export to Sheets

આ સમસ્યાની રીત

`countBinaryDescendingPairs(n, m, [[0,0,1,1,1],[1,0,0,1,1],[0,1,0,1,1],[1,1,0,1,1],[1,0,1,1,1],[0,0,0,1,1],[1,1,1,0,1],[0,1,1,0,1],[1,1,1,1,1],[0,1,1,1,1]])` આ સમસ્યા ઉકેલવા માટે `[0,0,0,1,1]` જેવી જોડી કે તેના પ્રતિબિંબની જરૂરિયાત નથી તે.

કુલ જોડી અને તેમાંથી ઘટતાની જોડી ક્રમ ઉલટાવીને કેટલી જોડીઓ મળશે તેનો ગુણોત્તર અને તેનો ઉકેલ શોધો.

Sorting Method:

Bubble sort with sort arr in Descending order

Write a program to sort the array in descending order using bubble sort algorithm. After Sorting return the array.
Note both the array and size is already given.
Input:
[5,2,8,1,3]

Output:
[8,5,3,2,1]

use javascript sorting

✦ અહીં તમારે Bubble Sort ની મદદથી ડેસ્કેન્ડિંગ ઓર્ડર સાથે ટેક કરી છે. આ અલ્ગોરિધમમાં આપેલી અરેમાં આસપાસની બે વેલ્યુઓની તુલનાની પ્રક્રિયાથી શરૂઆત કરવામાં આવે છે. દરેક સ્ટેપમાં તે સૌથી મોટા તત્વને અરેના અંતિમ ઇન્ડેક્સ તરફ ધકેલે છે, જેથી તે ઓર્ડર જાળવવામાં મદદ કરે છે.

તમારી **Descending Order** (ઉતરતા ક્રમ) ની વિનંતીના આધારે તેઓએ, અમારી જેમ જ કોઈ બે વેલ્યુની જગ્યાએ બદલી નાખીને તત્વોને ઓર્ડર કરી દીધા છે અને અંતમાં અરેનું રિટર્ન કર્યું છે.

Javascript Function (Bubble Sort)

```
javascript

function bubbleSortDescending(arr, size) {
// અહીં અરેની લંબાઈ અને અરેને પાસ કરવામાં આવે છે જે અરેની અંદર જ બદલાવ કરશે
for (let i = 0; i < size; i++) {
    // બહારનો લૂપ: દરેક પાસ પછી એક તત્વ નક્કી થઈ જાય છે
    for (let j = 0; j < size - i - 1; j++) {
        // આંતરિક લૂપ: પડોશી તત્વોની તુલના કરવામાં આવે છે
        // જો વર્તમાન તત્વ આગળના તત્વ કરતાં નાનું હોય તો
        if (arr[j] < arr[j + 1]) {
            // ડિસ્ટ્રક્ચરિંગ: બે વેલ્યુઓની જગ્યા એકબીજા સાથે બદલાવવા માટે સરળ રીત
            let temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
    }
}
return arr;
}

// ટેસ્ટિંગ
let inputArr = [5, 2, 8, 1, 3];
let n = inputArr.length;
console.log(bubbleSortDescending(inputArr, n));
// આઉટપુટ: [8, 5, 3, 2, 1]
```

આ પ્રોગ્રામ કેવી રીતે કામ કરે છે? (બબલ સોર્ટ)

અમારા એલ્ગોરિધમ ડેટા "બબલ" (પરપોટા) જેવાં તત્વોને ઉપર તરતાં મોકલે છે. આ આંતરિક પ્રક્રિયાનાં અલગ અલગ પગલાં નીચે આપ્યાં છે:

૧. પહેલો પાસ (First Pass):

- `[5, 2, 8, 1, 3]` → 5 અને 2 ની તુલના કરે છે. 5 મોટાં છે તેથી તેઓ સ્થાન બદલતાં નથી.
- `[5, 2, 8, 1, 3]` → 2 અને 8 ની તુલના કરે છે. 8 મોટાં છે તેથી તેઓ સ્થાન બદલે છે: `[5, 8, 2, 1, 3]`
- `[5, 8, 2, 1, 3]` → 2 અને 1 ની તુલના કરે છે. 2 મોટાં છે તેથી તેઓ સ્થાન બદલે છે: `[5, 8, 1, 2, 3]`
- `[5, 8, 1, 2, 3]` → 2 અને 3 ની તુલના કરે છે. 3 મોટાં છે તેથી તેઓ સ્થાન બદલે છે: `[5, 8, 1, 3, 2]`

૨. બીજો પાસ (Second Pass):

- `[5, 8, 1, 3, 2]` → 5 અને 8 ની તુલના કરે છે.
- `[5, 8, 1, 3, 2]` → 8 અને 1 ની તુલના કરે છે: `[5, 8, 1, 3, 2]`
- ... આ રીતે પૂરા થયેલા પાસમાં જેમ મુજબ સૌથી નાનાં તત્વો 2 અને 1 અંતમાં જતાં રહે છે.

કુલ અસરકારકતા:

- `[5, 2, 8, 1, 3]` → આ સેટમાં ઉપલબ્ધ ત્રિ-તત્વોવાળી જોડી જે તત્વો સ્થાન બદલે છે: `[5, 8, 1, 3, 2]`
- `[5, 8, 1, 3, 2]` → ઉપલબ્ધ જોડી: 2 કુલ જોડી કરી છે.

આ પ્રોગ્રામ કેવી રીતે કામ કરે છે? (સોર્ટ અલ્ગોરિધમ)

તમારો પ્રોગ્રામ ડેટાને "ઘટતા" (ઉતરતા) ક્રમમાં ગોઠવવા માટે સરળ અને અસરકારક સોર્ટિંગ અલ્ગોરિધમનો ઉપયોગ કરે છે. અહીં તેનું વિશ્લેષણ નીચે આપ્યું છે:

૧. મુખ્ય લૂપ (Main Loop):

- `for (let i = 0; i < size; i++)` આ લૂપ અરેમાંથી દરેક તત્વ પર પુનરાવર્તન કરવાનું નક્કી કરે છે.
- `let temp = arr[i];` આ તત્વને અસ્થાયી રૂપે સ્ટોર કરવામાં આવે છે જેથી તેની સ્થિતિ ગુમાવ્યા વગર તેની તુલના કરી શકાય.
- `while (j >= 0 && arr[j] < temp)` આ લૂપ ખાતરી કરે છે કે તત્વ તેની યોગ્ય સ્થિતિ સુધી પહોંચે ત્યાં સુધી તે ડાબી બાજુ ખસતું રહે.
- `arr[j + 1] = arr[j];` આ તત્વને જમણી બાજુ એક સ્થાન ખસેડે છે જેથી તત્વ માટે જગ્યા બને.
- `arr[j + 1] = temp;` આ અંતિમ પગલું તત્વને તેની યોગ્ય સ્થિતિમાં મૂકે છે.

૨. સમય અને અવકાશ જટિલતા (Time and Space Complexity):

- **સમય જટિલતા:** આ અલ્ગોરિધમ ઘણો ધીમો છે. તે સૌથી ખરાબ કિસ્સામાં (જ્યારે અરે પહેલેથી જ ઘટતા ક્રમમાં હોય) ચોરસ સમય લે છે, જેને $O(n^2)$ તરીકે દર્શાવવામાં આવે છે.
- **અવકાશ જટિલતા:** આ અલ્ગોરિધમ ઓછી મેમરી વાપરે છે કારણ કે તે ફક્ત એક અસ્થાયી ચલ (temp) નો ઉપયોગ કરે છે, જેને $O(1)$ તરીકે દર્શાવવામાં આવે છે.

કુલ મૂલ્યાંકન:

- `temp` અને `j` તત્વોને જેમ તેમ ખસેડતાં પહેલાં તેમની તુલના કરીને, તમારો પ્રોગ્રામ ખાતરી કરે છે કે દરેક તત્વ તેની યોગ્ય સ્થિતિ સુધી પહોંચે છે.
- **Time Complexity:** આ તત્વ પૂરતા મોટું મૂલ્ય હોવાને કારણે, તેનું મૂલ્ય ઘણું મોટું હોવાથી, તે ડેટાસેટ માટે ધીમું કામ કરશે.

શું તમે તમારો એ જણાવવા ઇચ્છશો કે તમારો કોડ અને અલ્ગોરિધમની સમજ તમારી પરવાનગી વગર સુધારવામાં આવે?

"અરેમાં 2 અને 3 કેટલી વાર આવે?"

understand me step by step

✦ ચોક્કસ, **Insertion Sort** આ અલ્ગોરિધમ ડેટાને એક પછી એક પ્રોસેસ કરે છે અને તેને અગાઉની સૉર્ટ કરેલી યાદી સાથે સરખાવીને તેની યોગ્ય જગ્યાએ મૂકી દેતો જાય છે. આ સ્ટેપ-બાય-સ્ટેપ પ્રક્રિયા છે:

૧. સરખામણી અને ખસેડવાની પ્રક્રિયા:

- `for (let i = 1; i < size; i++)` આ લૂપ બીજી વેલ્યુથી શરૂઆત કરીને દરેક તત્વ પર પુનરાવર્તન કરવાનું નક્કી કરે છે.
- `let temp = arr[i];` આ વર્તમાન તત્વને અસ્થાયી રૂપે સ્ટોર કરવામાં આવે છે જેથી તેની સ્થિતિ ગુમાવ્યા વગર તેની તુલના કરી શકાય.
- `while (j >= 0 && arr[j] > temp)` આ લૂપ ખાતરી કરે છે કે તત્વ તેની યોગ્ય સ્થિતિ સુધી પહોંચે ત્યાં સુધી તે ડાબી બાજુ ખસતું રહે.
- `arr[j + 1] = arr[j];` આ તત્વને જમણી બાજુ એક સ્થાન ખસેડે છે જેથી તત્વ માટે જગ્યા બને.
- `arr[j + 1] = temp;` આ અંતિમ પગલું તત્વને તેની યોગ્ય સ્થિતિમાં મૂકે છે.

૨. સમય અને અવકાશ જટિલતા (Time and Space Complexity):

- **સમય જટિલતા:** આ અલ્ગોરિધમ ધીમો છે. તે સૌથી ખરાબ કિસ્સામાં (જ્યારે અરે પહેલેથી જ ઘટતા ક્રમમાં હોય) ચોરસ સમય લે છે, જેને $O(n^2)$ તરીકે દર્શાવવામાં આવે છે.
- **અવકાશ જટિલતા:** આ અલ્ગોરિધમ ઓછી મેમરી વાપરે છે કારણ કે તે ફક્ત એક અસ્થાયી ચલ (temp) નો ઉપયોગ કરે છે, જેને $O(1)$ તરીકે દર્શાવવામાં આવે છે.

કુલ મૂલ્યાંકન:

- `temp` અને `j` તત્વોને જેમ તેમ ખસેડતાં પહેલાં તેમની તુલના કરીને, તમારો પ્રોગ્રામ ખાતરી કરે છે કે દરેક તત્વ તેની યોગ્ય સ્થિતિ સુધી પહોંચે છે.
- **Time Complexity:** આ તત્વ પૂરતા મોટું મૂલ્ય હોવાને કારણે, તેનું મૂલ્ય ઘણું મોટું હોવાથી, તે ડેટાસેટ માટે ધીમું કામ કરશે.

શું તમે તમારો એ જણાવવા ઇચ્છશો કે તમારો કોડ અને અલ્ગોરિધમની સમજ તમારી પરવાનગી વગર સુધારવામાં આવે?

Insertion Sort:

Set arry ascending order

Write a program to sort the array in ascending order using insertion sort algorithm. After Sorting return the array.
Note both the array and size is already given.
Input:
[5,2,8,1,3]

Output:
[1,2,3,5,8]

use javascript for loop

✦ ચોક્કસ, હવે તમારો **Insertion Sort** હવે ડેસ્કેન્ડિંગ ઓર્ડરને બદલે એસેન્ડિંગ (ચઢતા) ક્રમમાં અરેને સૉર્ટ કરવા માટે રચવામાં આવ્યો છે. તમારો પ્રોગ્રામ દરેક તત્વને તેની યોગ્ય સ્થિતિ સુધી ખસેડતો રહે છે, જેથી તે ડાબી બાજુની સૉર્ટ કરેલી યાદી સાથે સરખામણી કરે છે અને જરૂરિયાત મુજબ તત્વોને ખસેડે છે.

તમારી **Ascending Order** (ચઢતા ક્રમ) ની વિનંતીના આધારે, તેઓએ પણ તત્વોને સ્થાન બદલવાને બદલે અરેમાં ફેરફાર કર્યો છે અને અંતમાં અરેનું રિટર્ન કર્યું છે.

Javascript Function (Insertion Sort)

```
javascript

function insertionSort(arr, n) {
// અહીં અરેની લંબાઈ અને અરેને પાસ કરવામાં આવે છે
for (let i = 1; i < n; i++) {
    // બહારનો લૂપ: દરેક પાસ પછી એક તત્વ સૉર્ટ થઈ જાય છે
    let temp = arr[i];
    let j = i - 1;
    // વર્તમાન તત્વને સાચવો કારણ કે અમે તેની તુલના કરીશું અને તેને ખસેડીશું
    while (j >= 0 && arr[j] > temp) {
        // આંતરિક લૂપ: મોટાં તત્વોને જમણી બાજુ એક સ્થાન ખસેડો
        arr[j + 1] = arr[j];
        j--;
    }
    // તત્વને યોગ્ય સ્થાને મૂકો
    arr[j + 1] = temp;
}
return arr;
}

// ટેસ્ટિંગ
let inputArr = [5, 2, 8, 1, 3];
let n = inputArr.length;
console.log(insertionSort(inputArr, n));
// આઉટપુટ: [1, 2, 3, 5, 8]
```

આ પ્રોગ્રામ કેવી રીતે કામ કરે છે? (સૉર્ટ અલ્ગોરિધમ)

તમારો અલ્ગોરિધમ ડેટાને "ઘટતા" (ઉતરતા) ક્રમમાં ગોઠવવા માટે સરળ અને અસરકારક સૉર્ટિંગ અલ્ગોરિધમનો ઉપયોગ કરે છે. અહીં તેનું વિશ્લેષણ નીચે આપ્યું છે:

૧. મુખ્ય લૂપ (Main Loop):

- `for (let i = 1; i < size; i++)` આ લૂપ અરેમાંથી દરેક તત્વ પર પુનરાવર્તન કરવાનું નક્કી કરે છે.
- `let temp = arr[i];` આ તત્વને અસ્થાયી રૂપે સ્ટોર કરવામાં આવે છે જેથી તેની સ્થિતિ ગુમાવ્યા વગર તેની તુલના કરી શકાય.
- `while (j >= 0 && arr[j] < temp)` આ લૂપ ખાતરી કરે છે કે તત્વ તેની યોગ્ય સ્થિતિ સુધી પહોંચે ત્યાં સુધી તે ડાબી બાજુ ખસતું રહે.
- `arr[j + 1] = arr[j];` આ તત્વને જમણી બાજુ એક સ્થાન ખસેડે છે જેથી તત્વ માટે જગ્યા બને.
- `arr[j + 1] = temp;` આ અંતિમ પગલું તત્વને તેની યોગ્ય સ્થિતિમાં મૂકે છે.

૨. સમય અને અવકાશ જટિલતા (Time and Space Complexity):

- **સમય જટિલતા:** આ અલ્ગોરિધમ ધીમો છે. તે સૌથી ખરાબ કિસ્સામાં (જ્યારે અરે પહેલેથી જ ઘટતા ક્રમમાં હોય) ચોરસ સમય લે છે, જેને $O(n^2)$ તરીકે દર્શાવવામાં આવે છે.
- **અવકાશ જટિલતા:** આ અલ્ગોરિધમ ઓછી મેમરી વાપરે છે કારણ કે તે ફક્ત એક અસ્થાયી ચલ (temp) નો ઉપયોગ કરે છે, જેને $O(1)$ તરીકે દર્શાવવામાં આવે છે.

કુલ મૂલ્યાંકન:

- `temp` અને `j` તત્વોને જેમ તેમ ખસેડતાં પહેલાં તેમની તુલના કરીને, તમારો પ્રોગ્રામ ખાતરી કરે છે કે દરેક તત્વ તેની યોગ્ય સ્થિતિ સુધી પહોંચે છે.
- **Time Complexity:** આ તત્વ પૂરતા મોટું મૂલ્ય હોવાને કારણે, તેનું મૂલ્ય ઘણું મોટું હોવાથી, તે ડેટાસેટ માટે ધીમું કામ કરશે.

શું તમે તમારો એ જણાવવા ઇચ્છશો કે તમારો કોડ અને અલ્ગોરિધમની સમજ તમારી પરવાનગી વગર સુધારવામાં આવે?

Same Value in Arry then output True

when same value in single array then output True.
make use of javascript for loop

✦ ચોક્કસ તેવી વિનંતી તમારી પાસ કરવા (Explicitly) માટે હું તૈયાર છું. તમારાથી પહેલાં આપેલી **Nested Loops** ઉપરાંત વૈકલ્પિક રીતે તમારા ઓપ્ટિમાઈઝ કરેલા કોડને અહીં રજૂ કરવામાં આવ્યો છે.

અહીં કોડ અંગેનાં અવલોકનો નીચે છે:

વૈકલ્પિક ઉકેલ:

તમારો અહીં તે કરવાનો તેવો વિકલ્પ તમારાં તમામ તત્વોમાં તફાવત શોધીને તમામ ઈન્ડેક્સ ટ્રેકિંગ (ટ્રેકિંગ) દ્વારા તે તફાવતનું ગણતરી કરીને, જો તે ઈન્ડેક્સ `-1` જેટલાં હોય તો જ `true` તરીકે પરત કરશે.

ઓપ્ટિમાઈઝેશન:

- `for (let i = 0; i < n; i++)` આ લૂપ તમારાં તમામ તત્વોમાં તફાવત શોધીને તેમાં ઈન્ડેક્સ ટ્રેકિંગ (ટ્રેકિંગ) દ્વારા તે તફાવતનું ગણતરી કરીને, જો તે ઈન્ડેક્સ `-1` જેટલાં હોય તો જ `true` તરીકે પરત કરશે.
- `if (arr[i] === arr[i] && i !== -1)` આ લૂપ ખાતરી કરે છે કે તમારાં તમામ તત્વોમાં તફાવત શોધીને તેમાં ઈન્ડેક્સ ટ્રેકિંગ (ટ્રેકિંગ) દ્વારા તે તફાવતનું ગણતરી કરીને, જો તે ઈન્ડેક્સ `-1` જેટલાં હોય તો જ `true` તરીકે પરત કરશે.

શું તમે તમારો એ જણાવવા ઇચ્છશો કે તમારો કોડ અને અલ્ગોરિધમની સમજ તમારી પરવાનગી વગર સુધારવામાં આવે?

Square matrix Return the max-no from both diagonals

Given a square matrix. Return the maximum number from both diagonals.
Input:
[[1,2,3],
[4,5,6],
[7,8,9]]
Output:17

use of javascript

✦ ચોરસ મેટ્રિક્સ (Square Matrix) માટે બે કર્ણ (ડાયગોનલ) હોય છે:

- **Primary Diagonal (પ્રાથમિક કર્ણ):** ડાબી ટોચથી જમણી નીચે તરફ જતો કર્ણ. અહીં તે `[1, 5, 9]` છે.
- **Secondary Diagonal (બીજો કર્ણ):** જેમાં જમણી ટોચથી ડાબી નીચે તરફ જતું મૂલ્ય છે `[3, 5, 7]`.

અહીં કોડ, `countArray` તરીકે છે.

Javascript કોડ:

```
javascript

function findMaxDiagonal(countArr) {
// અહીં અરેની લંબાઈ અને અરેને પાસ કરવામાં આવે છે
let n = countArr.length;
let maxDiagonal = countArr[0][0];

let mainDiagonal = countArr[0][0];

for (let i = 0; i < n; i++) {
    // મુખ્ય કર્ણ (ડાબી ટોચથી જમણી નીચે) માટે તત્વોનું મહત્તમ મૂલ્ય શોધો
    if (countArr[i][i] > mainDiagonal) {
        mainDiagonal = countArr[i][i];
    }

    // ગૌણ કર્ણ (જમણી ટોચથી ડાબી નીચે) માટે તત્વોનું મહત્તમ મૂલ્ય શોધો
    let secondaryDiagonal = countArr[i][n - 1 - i];
    if (secondaryDiagonal > maxDiagonal) {
        maxDiagonal = secondaryDiagonal;
    }
}

return maxDiagonal;
}

// ઉદાહરણ
let matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
console.log(findMaxDiagonal(matrix));
// આઉટપુટ: 17
```

આ પ્રોગ્રામ કેવી રીતે કામ કરે છે?

૧. Primary Diagonal (મુખ્ય કર્ણ): તમારો પ્રોગ્રામ ડાબી ટોચથી જમણી નીચે તરફ જતા કર્ણ પર ધ્યાન આપે છે. `[0][0]`, `[1][1]`, `[2][2]`. અહીં તમારાં તત્વ `matrix[i][i]` જેવાં હોવાં જોઈએ. તમારો પ્રોગ્રામ આ તત્વોની તુલના કરશે. 1, 5, 9 છે. મહત્તમ મૂલ્ય 9 છે.

૨. Secondary Diagonal (ગૌણ કર્ણ): આ કર્ણ જમણી ટોચથી ડાબી નીચે તરફ જતો કર્ણ છે. તેના તત્વો છે `[0][2]`, `[1][1]`, `[2][0]`. તેનું સૂત્ર `matrix[i][n - 1 - i]` જેવું છે. તમારો પ્રોગ્રામ આ તત્વોની તુલના કરશે. 3, 5, 7 છે. મહત્તમ મૂલ્ય 7 છે.

કુલ મૂલ્યાંકન:

આ કોડ ચોરસ મેટ્રિક્સ પર કર્ણ (ડાયગોનલ) સરવાળો શોધવાનો એક કાર્યક્ષમ રીત છે. તે ફક્ત એક જ લૂપનો ઉપયોગ કરે છે અને તે મોટા મેટ્રિક્સ માટે પણ ઝડપી કામ કરશે.

શું તમે તમારો એ જણાવવા ઇચ્છશો કે તમારો કોડ અને અલ્ગોરિધમની સમજ તમારી પરવાનગી વગર સુધારવામાં આવે?

DSA Exam 1

Print the following pattern using any javascript looping technique.

✦ Show thinking

હવે અહીંથી તેઓ જુદાજુદા પેટર્ન કેવી રીતે બનાવી શકાય છે તેનાં ઉદાહરણ આપવા માટે હું તૈયાર છું. નીચેના પેટર્ન અહીં આપેલાં છે:

પેટર્ન ૧ (Pattern 1):

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

પેટર્ન ૨ (Pattern 2):

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

પેટર્ન ૩ (Pattern 3):

```
*********
*       *
*       *
*       *
*********
```

અહીં તમારાં તેવાં ઉદાહરણો કેવી રીતે બનાવી શકાય તેનાં ઉદાહરણ આપવા માટે હું તૈયાર છું.

Javascript કોડ

```
javascript

function printPattern1() {
for (let i = 1; i <= 5; i++) {
    for (let j = 1; j <= i; j++) {
        // આંતરિક લૂપ: વર્તમાન પંક્તિ માટે નંબરો પ્રિન્ટ કરો
        process.stdout.write(j + " ");
    }
    // બહારનો લૂપ: પંક્તિ પૂર્ણ થયા પછી નવી લાઈન પ્રિન્ટ કરો
    console.log("");
}
}

// ફંક્શનને કૉલ કરો
printPattern1();
```

Javascript કોડ

```
javascript

function printPattern2() {
for (let i = 5; i >= 1; i--) {
    for (let j = 1; j <= i; j++) {
        // આંતરિક લૂપ: વર્તમાન પંક્તિ માટે નંબરો પ્રિન્ટ કરો
        process.stdout.write(j + " ");
    }
    // બહારનો લૂપ: પંક્તિ પૂર્ણ થયા પછી નવી લાઈન પ્રિન્ટ કરો
    console.log("");
}
}

// ફંક્શનને કૉલ કરો
printPattern2();
```

Javascript કોડ

```
javascript

function printPattern3() {
for (let i = 1; i <= 5; i++) {
    // બહારનો લૂપ: પંક્તિઓ પર પુનરાવર્તન કરો
    let row = "";
    for (let j = 1; j <= 5; j++) {
        // આંતરિક લૂપ: વર્તમાન પંક્તિ માટે અક્ષરો બનાવો
        if (i === 1 || i === 5 || j === 1 || j === 5) {
            row += "*";
        } else {
            row += " ";
        }
    }
    // પંક્તિ પૂર્ણ થયા પછી, તેને પ્રિન્ટ કરો
    console.log(row);
}
}

// ફંક્શનને કૉલ કરો
printPattern3();
```

```
    console.log(line);
  }
}

printPattern(5);
```

પેટર્ન 5 લેવેલ માટે આઉટપુટ (Step by Step):

- **લાઇન 1 (i=0):** n-i-1 = 4 જગ્યા, r = 0,કોલ્મ = ********
- **લાઇન 2 (i=1):** n-i-1 = 3 જગ્યા, r = 0,કોલ્મ = ******** ******
- **લાઇન 3 (i=2):** n-i-1 = 2 જગ્યા, r = 0,કોલ્મ = ******** ****** ********
- **લાઇન 4 (i=3):** n-i-1 = 1 જગ્યા, r = 0,કોલ્મ = ******** ****** ******** ******
- **લાઇન 5 (i=4):** n-i-1 = 0 જગ્યા, r = 0,કોલ્મ = ******** ****** ******** ****** ********

લોજિકલ સમજૂતી:

- દરેક લાઇન માટે, કોલ્મ 0 થી i+1 સુધી લૂપ ચલાવો
- `spaces` **ગણતરી:** જગ્યા = n - i - 1 સૂત્રનો ઉપયોગ કરીને જગ્યાની સંખ્યા ગણો, પછી i+1 થી i+1 સુધી i.e. `spaces` - 1 લખવા માટે લૂપ.
- `line` **બનાવટ:** જગ્યાની સંખ્યા અને સ્ટાર પેટર્ન બંનેને જોડવાની રીતો સાથે `console.log` કરીને લાઇન.

શું તમારી આ પેટર્નને લોજિક અને રીઝલ્ટ પેટર્ન વિશે વધુ ઉદાહરણ સમજાવું છે જેથી તું સુપર Butterfly પેટર્ન બનાવી શકે?

---

you

હોલ્લાઇન ખાલી જગ્યાવાળી આ પેટર્નને લોજિક સાથે સિમ્પ્લિફાઈ કરીને, અને તે હોલ્લ સુપર Butterfly (ચણિયો) પેટર્ન બને એ.

આ બધી જગ્યાવાળી બનાવીને લોજિક કેવી રીતે બદલાય એ કહો? સુપર ચિઝેલિંગ પણ કરી જ શકું છું ચણિયો ને સુપર્બ બનાવી.

Javascript કોડ (Butterfly Pattern):

```
javascript
function printButterfly(n) {
  // n is the number of rows (must be odd)
  for (let i = 0; i < n; i++) {
    let spaces = n - i - 1;
    for (let j = 0; j < spaces; j++) {
    for (let k = 0; k < spaces; k++) {
    for (let k = 0; k < spaces; k++) {
      process.stdout.write(" ");
    }
    console.log(line);

    // 2. સ્ટાર્સનો ભાગ (ડાબી બાજુનો)
    // બધી રો 1 થી શરૂ કરીને સ્ટાર્સની સંખ્યા વધારો (odd numbers)
    for (let k = 0; k < i; k++) {
      process.stdout.write("*");
      for (let j = 0; j < i; j++) {
      for (let j = 0; j < i; j++) {
      for (let j = 0; j < i; j++) {
      process.stdout.write("*");
    }
    console.log(line);
  }
}

printButterfly(5);
```

આ લોજિકથી શું થશે?

- **લોજિકની સુંદરતા:** `spaces = n - i - 1` સૂત્ર બધી જગ્યાવાળી હાફની લાઇનમાં સરસ કામ કરે છે.
- **પ્રોબ્લેમ આઉટપુટ માં:** આઉટપુટ માં જગ્યાવાળી હાફની લાઇનમાં સ્ટાર્સ નથી આવતા જે આ ઉકેલ કામ કરતો નથી.
- **સોલ્યુશન:** જગ્યાવાળી હાફ માટે અલગ લૂપ ચલાવવો પડશે જે સ્ટાર્સ પ્રિન્ટ કરે.

શું તમારી આ પેટર્નને લોજિક સાથે ઈમ્પ્રુવ કરીને સુપર Butterfly પેટર્ન બનાવી શકું? જેમાં તું કહેશે તે પ્રમાણે સ્ટાર્સ અને જગ્યાવાળી હાફ બંનેનું સુંદર કામ થાય.

---

you

Pattern

```
**********
****  ****
***    ***
**      **
*        *
*        *
**      **
***    ***
****  ****
**********
```

હવે આ પેટર્નમાં જગ્યાવાળી હાફ પણ સ્ટાર્સ જ કરી દો. તો તે ચિઝેલિંગ (Pyramid) બનાવ કરતાં કરતાં (Diamond) પેટર્ન કહેવાય. પરંતુ કોડ?

---

assistant

હવે આ પેટર્નમાં જગ્યાવાળી હાફ પણ સ્ટાર્સ જ કરી દો. તો તે ચિઝેલિંગ (Pyramid) બનાવ કરતાં કરતાં (Diamond) પેટર્ન કહેવાય. પરંતુ કોડ?

```
**********
****  ****
***    ***
**      **
*        *
*        *
**      **
***    ***
****  ****
**********
```

Print the following pattern using any Javascript looping technique.

હવે આપણે પેટર્નને બધી જગ્યાવાળી બનાવવા જ છે કોલ્મ 0 થી કોલ્મ 9 સુધી (Pattern) જેમાં તારાઓ જગ્યાવાળા અને કોલ્મ 1,2,3, 7,8,9 સુધી સ્ટાર્સ (Diamond), અને તેમાં ફુલીટ્રાન્સ છે.

આઈડિયાઝ, Javascript કોડ જોઈ જો

Javascript કોડ

```
javascript
function printPattern(n) {
  for (let i = 0; i < n; i++) {
    let line = "";

    // 1. પહેલી જગ્યાનો ભાગ (i+1)
    // આ ભાગમાં સ્ટાર નહીં પણ સ્પેસ જ મૂકવાની છે.
    for (let j = 0; j < i; j++) {
      line += " ";
    }

    // 2. બાકીની જગ્યાનો ભાગ (n-i)
    // આ ભાગમાં સ્ટાર્સ મૂકો (1 થી શરૂ કરીને n સુધી)
    for (let j = i; j < n; j++) {
      line += "*";
    }

    // 3. બાકીનો સ્ટાર્સનો ભાગ (Reverse)
    // આ ભાગમાં સ્પેસ્સ (i + 1)થી શરૂ કરીને n-1 સુધીની સંખ્યા
    for (let spaces = i + 2; i < 1; i++) {
    for (let j = i + 2; j < n; j++) {
      line += " ";
    }

    // 4. બાકીનો સ્ટાર્સનો ભાગ (n-i)
    // આ ભાગમાં સ્ટાર્સ મૂકો (1 થી શરૂ કરીને n સુધી)
    for (let j = i; j < n; j++) {
      line += "*";
    }

    console.log(line);
  }
}

// 5 લાઇન માટે કોલ્મ કરવા માટે કોલ્મ
printPattern(5);
```

લોજિકલી કામચલ્યું (Step-by-Step):

આ પેટર્ન કામ કરવાનું લોજિકલી છે.

1. **ખાલી જગ્યાનો ભાગ (i+1):** જે મુજબ ખાલી, i સુધી પહેલી ખાલી છે. તે ખાલી રહી છે તેમાં તે કામચલ કરી જાઓ પેટર્ન બની જાઓ છે.
2. **તારાનો ભાગ (n-i):** અહીં આવે જગ્યાવાળી અને સ્ટાર્સ તારાનો ભાગ. 1 જગ્યાવાળી જે તારાનો ભાગ બની જાય, તારાઓ જગ્યાવાળી અને તારાનો ફુલીટ્રાન્સ છે.
3. **બાકીની જગ્યાનો (Reverse):** અહીં જગ્યાવાળી `i + 2` થી `n-1` સુધી જગ્યાવાળી. `i + 2` થી `n-1` સુધી જગ્યાવાળી. `i + 2` થી `n-1` સુધી જગ્યાવાળી અને તારાનો ફુલીટ્રાન્સ છે.
4. **બાકીનો સ્ટાર્સનો (n-i):** તારાનો ભાગ. 1 થી `n-1` સુધી જગ્યાવાળી અને તારાનો ફુલીટ્રાન્સ છે.

લોજિકલી કામચલ્યું (Step-by-Step):

- **લાઇન 1:** `spaces = n - i - 1` સૂત્રનો ઉપયોગ કરીને જગ્યાવાળી જગ્યાવાળી સંખ્યા ગણો, પછી i+1 થી i+1 સુધી i સુધી `spaces - 1` લખવા માટે લૂપ ચલાવો.
- **લાઇન 2:** `line` બનાવટ: જગ્યાવાળી સંખ્યા અને સ્ટાર પેટર્ન બંનેને જોડવાની રીતો સાથે `console.log` કરીને લાઇન.

શું તમારી આ પેટર્નને લોજિક અને રીઝલ્ટ પેટર્ન વિશે વધુ ઉદાહરણ સમજાવું છે જેથી તું સુપર Butterfly પેટર્ન બનાવી શકે?

---

you

Select the incorrect option:

Select one answer

1.First index of an array is always zero

2.Last index of an array is always one less than the size of the array

3.Size of the array is always one less than the total number of elements

4.Size of the array is always zero less than the total number of elements (તોફુટ) આ. ઉપર મોકૂકું છે બધાં તારાઓ જગ્યાવાળી અને સ્ટાર્સ પેટર્નમાં (Total elements) જો એક જ પણ તું જોઈ જો કોલ્મ 0 થી કોલ્મ 9 સુધી, તો કોલ્મ ચાલીસ જગ્યાવાળી અને તારાઓ જગ્યાવાળી અને સ્ટાર્સ પેટર્નમાં

૩. સંખ્યાની જુગલ (2,3) એ જુગલ નથી કારણ કે 3 નું 2 બાકી રહે અને સંખ્યાઓનો તફાવત એક જેટલો (subtracting) પણ નથી કારણ કે બેથી અલગ (3,5) એક જુગલ તરીકે ગણી શકાશે.

આઉટપુટ Pattern

તમારા ઇનપુટ [1,2,3,5,6,7,8] માટે:

- 1-2 બેથી 2 : (1, 2), (2, 3), (3,5), (5,6), (5,7),(6,8)
- 2-3 બેથી 2 : (2, 3), (3,5), (5,6), (5,8)
- 3-5 બેથી 2 : (3, 5), (5,6), (5,8)
- ... આ રીતે તમે શોધી શકશો.

શું તમે તમારો જે સવાલનું જવાબ શોધવો તે આ જ રીતે. [1,2,3,4,5] તમારો ઇનપુટ કહેતો નથી શું કુલે કેટલી રીતે શકશે?

Declaire arr size n and return 12...n arr

Write a program to declaire an array of size n and asse the numbers 1,2,3,4 ...n in the array and return the array.

use of javascript for loop

✦ ચોકસાઇ, આ જુગલ તમને શું આપશે તેનો જવાબ તેની સાથે જવાબવું થશે. આ જુગલને ઇનપુટ તરીકે કોઇ ડેટા નથી તે જુગલનું નામ તમારાં (name) કરીશું.

બેથી ઓછાં તફાવતવાળાં જુગલ છે

JavaScript કોડમાં:

javascript

```
function countAscendingPairs() {
  // 1. ડેટા વ્યવસ્થિત કરો (બેથી ઓછા તફાવત)
  var result = [];

  // 2. i થી શરૂ કરીને દરેક જોડી ચેક કરો
  for (var i = 1; i < n; i++) {
    for (var j = i + 1; j <= n; j++) {
      result.push([i, j]);
    }
  }

  // 3. પરિણામ પાછું આપો
  return result;
}

// પરીક્ષણ કરવા માટે:
console.log(countAscendingPairs(8));// અપેક્ષિત: [1, 2], [1, 3], [1, 4], [2]
```

આ કોડ કેવી રીતે કામ કરે છે? (Step-by-Step)

૧. var result = [];: આ જુગલને ખાલી રાખીશું અને તેમાં જવાબો ઉમેરતાં જઈશું.

૨. for (var i = 1; i < n; i++): i થી શરૂઆત કરો for (var j = i + 1; j <= n; j++): આ જુગલ બનાવે છે કે બેથી ઓછાં તફાવતવાળાં જુગલ.

૩. if (j - i < 2): આ જુગલ ઉમેરતાં પહેલાં જુગલ ઉમેરતાં પહેલાં, j અને i વચ્ચેનો તફાવત 2 કરતાં ઓછો છે કે નહીં તે ચકાસો. (j - i < 2) એટલે j અને i વચ્ચેનો તફાવત 1 છે.

૪. result.push([i, j]): જો શરત સાચી હોય તો જ આ જુગલને પરિણામમાં ઉમેરો.

૫. return result;: પરિણામ પાછું આપો.

બધાં નીચલાં (Pre-defined Arrs)

જો તમારે કોઈ ચોક્કસ પ્રકારનાં જુગલ તમારા કોડમાં, તો આ રીતે પણ કરી શકો છો.

javascript

```
var result = new Array(0);
var n = 8;// અહીં તમારો n
result[0] = 1 + 1;// પ્રથમ જુગલ
```

શું તમે તમારો જે સવાલનું જવાબોનો ઇનપુટ જે કોડ કેવી રીતે કામ કરે છે?

Return Reversed Arry

You have been given an array your task is to reverse the array and return the new reversed array.

Example 1:-
Input :
[1,2,3,4,5]
Output:
[5,4,3,2,1]

use of javascript for loops

✦ બેશક! તમારાં સવાલ પ્રમાણે અમારાં બધાં જવાબ એક જ વાર આપીશું. તમારાં સવાલ પ્રમાણે અમારાં જવાબોનું ઇનપુટ તરીકે [1,2,3,4,5] આપીશું તો તમારાં જવાબ [5,4,3,2,1] આવશે.

બેથી ઓછાં તફાવતવાળાં જુગલ છે

JavaScript કોડમાં:

javascript

```
function reverseArray(arr) {
  var reversed = [];// નવો જોડ જેમાં રિવર્સ કરેલાં જુગલો રાખવાં
  var n = arr.length;

  // મૂળ ડેટા પર લૂપ ચલાવો (પાછળ થી શરૂ કરીને જુગલ ઉમેરો)
  for (var i = n - 1; i >= 0; i--) {
    // તમારાં જુગલોને નવાં જુગલમાં ઉમેરો
    reversed.push(arr[i]);
  }

  // રિવર્સ કરેલ જુગલ પરત કરો
  return reversed;
}
var input = [1, 2, 3, 4, 5];
var output = reverseArray(input);
console.log(output);// પરિણામ: [5, 4, 3, 2, 1]
```

આ પ્રોગ્રામ કેવી રીતે કામ કરે છે? (Step-by-Step)

૧. var reversed = [];: આ જુગલને ખાલી રાખીશું અને તેમાં તમારાં જુગલોને ઉમેરતાં જઈશું.

૨. for (var i = n - 1; i >= 0; i--): આ જુગલ છેલ્લાં તત્વથી શરૂ કરીને પહેલાં તત્વ સુધી પાછું જાય છે.

૩. reversed.push(arr[i]): આ જુગલ તમારાં જુગલોને નવાં જુગલમાં ઉમેરે છે.

૪. return reversed;: આ જુગલ પરિણામ પાછું આપે છે.

બધાં નીચલાં (Pre-defined Arrs)

જો તમારે કોઈ ચોક્કસ પ્રકારનાં જુગલ તમારા કોડમાં, તો આ રીતે પણ કરી શકો છો.

javascript

```
var result = new Array(5);
var n = 5;// અહીં તમારો n
result[0] = 5 * 5;// પ્રથમ જુગલ
```

શું તમે તમારો જે સવાલનું જવાબોનો ઇનપુટ જે કોડ કેવી રીતે કામ કરે છે?

Print Pattern

```
*
**
***
****
*****
****
***
**
*
```

function print_pattern()
/* Function to print the pattern */
var str = "";
for (var i = 1; i <= 5; i++){

for (var j = 1; j <= i; j++)
str += "*";

return str;
}
}

solved this code

✦ Show thinking ^

તમારા કોડમાં એક મુખ્ય ભૂલ છે કે તમે var str = "*"; વાપરી રહ્યાં છો. પરિણામ તરીકે, તમારાં જુગલ પ્રમાણે, અમારાં જુગલ પ્રમાણે જ પ્રથમ સ્ટાર (*) ઉમેરાઈ જશે. આ સ્ટાર બેથી વધારે વાર ઉમેરાઈ જશે. તમારાં જુગલ પ્રમાણે, અમારાં જુગલ પ્રમાણે જ પ્રથમ સ્ટાર (*) ઉમેરાઈ જશે.

તમે જે કોડ બનાવો છો, તે બધાં માટે બેથી વધારે સ્ટાર ઉમેરાઈ જશે. તમારાં જુગલ પ્રમાણે

૧. પિરામીડ માટે કોડિંગ કરેલ ફંક્શન

આ પેટર્ન માટે અમારાં જુગલમાં એક ચોક્કસ પ્રકારનો ફેરફાર કરવો પડશે. તમારાં જુગલ પ્રમાણે કોડ નીચે મુજબ હશે:

javascript

```
function printPyramid() {
  var str = "";// ખાલી સ્ટ્રિંગ
  for (var i = 1; i <= 5; i++) {
    var str = "";
    for (var j = 1; j <= i; j++) {
      str += "*";
    }
    // દરેક લાઇન પછી નવી લાઇન ઉમેરો
    str += "\n";
    console.log(str);
  }
}
// ફંક્શન કલ્સ કરો
printPyramid();
```

૨. આડાપ્રમાણમાં રોમ્બસ માટે ઉપયોગી ફંક્શન ફંક્શન

આ પેટર્ન માટે કોડમાં અલગ-અલગ સ્પેસ ઉમેરવાં પડશે. તમારાં જુગલ પ્રમાણે કોડ નીચે મુજબ હશે:

javascript

```
function printRhombusPyramid() {
  var str = "";// ખાલી સ્ટ્રિંગ
  for (var i = 1; i <= 5; i++) {
    // પહેલી લાઇન માટે સ્પેસ ઉમેરો
    for (var j = 5 - i; j >= 1; j--) {
      str += " ";
    }
    // પછી સ્ટાર્સ ઉમેરો
    for (var j = 1; j <= 2 * i - 1; j++) {
      str += "*";
    }
    // દરેક લાઇન પછી નવી લાઇન ઉમેરો
    str += "\n";
    console.log(str);
  }
}
printRhombusPyramid();
```

તમારાં સવાલનો ઉકેલ

- console.log(str): આખાયે પરિણામ પ્રિન્ટ કરશે. (newline) તમારાં બધાં સ્ટાર્સને પોતપોતાની લાઇન પર પ્રિન્ટ કરશે.
- console.log: બધાં.
- **નીચેનાં ઉકેલ:** console.log(str); જ્યારથી જુગલ્સ સ્ટાર્સ બધાં (newline) તમારાં બધાં સ્ટાર્સને પોતપોતાની લાઇન પર પ્રિન્ટ કરશે.

શું તમારાં બધાં કોડ પેટર્નનાં કોડ કોઈ કોડનાં બધાંથી **Butterfly** પેટર્ન બનાવવાનાં હોય તો શું?

Return Max num. from both diagonls

Given a square matrix. Return the maximum number from both diagonals.

Input
1 2 3
4 5 6
7 8 9
Output 9 ?

use of javascript for loop

✦ નીચેના ચિત્રમાં (Square Matrix) બંને કર્ણ (કર્ણ ઓફ મેટ્રિક્સ) નીચે મુજબ હશે. તમારાં કોડ પ્રમાણે બધાં સંખ્યાઓ ઉમેરવાં પડશે.

તમારાં ઇનપુટ પ્રમાણે (3 × 3 મેટ્રિક્સ)

- **Primary Diagonal:** 1, 5, 9 (ડાબેથી જમણે તરફ કર્ણ)
- **Secondary Diagonal:** 3, 5, 7 (જમણેથી ડાબે તરફ કર્ણ).

બધાં કર્ણ તફાવતવાળાં છે

JavaScript કોડમાં:

javascript

```
function findMaxDiagonalSum(matrix) {
  var n = matrix.length;

  // પરિણામને અત્યંત નીચેનાં મૂલ્યથી શરૂ કરો
  var maxPrimary = matrix[0][0];
  var maxSecondary = matrix[0][n - 1];

  for (var i = 0; i < n; i++) {
    // પ્રાઇમરી ડાયગોનલ ચેક કરો (ડાબેથી જમણે તરફ)
    if (matrix[i][i] > maxPrimary) {
      maxPrimary = matrix[i][i];
    }

    // સેકન્ડરી ડાયગોનલ ચેક કરો (જમણેથી ડાબે તરફ)
    if (matrix[i][n - 1 - i] > maxSecondary) {
      maxSecondary = matrix[i][n - 1 - i];
    }
  }

  return maxPrimary > maxSecondary ?
  maxPrimary :
  maxSecondary;
}

// ઉદાહરણ
var matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];

console.log(findMaxDiagonalSum(matrix));// અપેક્ષિત: 9 ?
```

આ પ્રોગ્રામ કેવી રીતે કામ કરે છે? (સ્ટેપ-બાય-સ્ટેપ)

૧. Primary Diagonal ઉમેરવા: આ પ્રોગ્રામ દરેક પંક્તિ ઉમેરે છે કર્ણ ઓફ મેટ્રિક્સ ઉમેરવા તરફ છે?

- matrix[0][0] → 1
- matrix[1][1] → 5
- matrix[2][2] → 9 → **કુલ: 9**

૨. Secondary Diagonal ઉમેરો: આગળ જોઈ તરફ કર્ણ બેથી ઓછાં તફાવતવાળાં છે. શરૂઆત: matrix[0][n - 1] → 3

- પંક્તિ 1: matrix[1][n - 2] → matrix[1][1] → 5
- પંક્તિ 2: matrix[2][n - 3] → matrix[2][0] → 7
- પંક્તિ 3: matrix[3][n - 4] → matrix[3][-1] → **અવગણો!**

પરિણામ: બંને પેટર્નમાં સંખ્યાઓ ઉમેરતાં પહેલાં તે

```
convert_sequence(
    seq,
    to_base=10,
    from_base=2,
    lut_path=None,
    lut_data=None,
    lut_inline=None,
    max_digits=1000000,
    max_output_length=1000000,
    allow_negative=True,
    strict=True,
    output_sep=' ',
    verbose=False,
    show_progress=False,
    progress_interval=1000000,
    progress_bar_width=40,
    progress_stream=None,
    max_input_length=1000000,
    max_lut_length=1000000,
    max_lut_digits=1000000,
    max_lut_output_length=1000000,
    max_lut_input_length=1000000,
    max_lut_output_sep_length=1000000,
    max_lut_output_sep
```