JS Code:

function smallest\_subarray\_with\_given\_sum(s, arr) {

let windowSum = 0,

minLength = Infinity,

windowStart = 0;

for (windowEnd = 0; windowEnd < arr.length; windowEnd++) {

windowSum += arr[windowEnd]; // add the next element

// shrink the window as small as possible until the 'window\_sum' is smaller than 's'

while (windowSum >= s) {

minLength = Math.min(minLength, windowEnd - windowStart + 1);

windowSum -= arr[windowStart];

windowStart += 1;

}

}

if (minLength === Infinity) {

return 0;

}

return minLength;

}

console.log(`Smallest subarray length: ${smallest\_subarray\_with\_given\_sum(7, [2, 1, 5, 2, 3, 2])}`);

console.log(`Smallest subarray length: ${smallest\_subarray\_with\_given\_sum(7, [2, 1, 5, 2, 8])}`);

console.log(`Smallest subarray length: ${smallest\_subarray\_with\_given\_sum(8, [3, 4, 1, 1, 6])}`);

C++

using namespace std;

#include <iostream>

#include <limits>

#include <vector>

class MinSizeSubArraySum {

public:

static int findMinSubArray(int S, const vector<int>& arr) {

int windowSum = 0, minLength = numeric\_limits<int>::max();

int windowStart = 0;

for (int windowEnd = 0; windowEnd < arr.size(); windowEnd++) {

windowSum += arr[windowEnd]; // add the next element

// shrink the window as small as possible until the 'windowSum' is smaller than 'S'

while (windowSum >= S) {

minLength = min(minLength, windowEnd - windowStart + 1);

windowSum -= arr[windowStart]; // subtract the element going out

windowStart++; // slide the window ahead

}

}

return minLength == numeric\_limits<int>::max() ? 0 : minLength;

}

};

int main(int argc, char\* argv[]) {

int result = MinSizeSubArraySum::findMinSubArray(7, vector<int>{2, 1, 5, 2, 3, 2});

cout << "Smallest subarray length: " << result << endl;

result = MinSizeSubArraySum::findMinSubArray(7, vector<int>{2, 1, 5, 2, 8});

cout << "Smallest subarray length: " << result << endl;

result = MinSizeSubArraySum::findMinSubArray(8, vector<int>{3, 4, 1, 1, 6});

cout << "Smallest subarray length: " << result << endl;

}

Python:

import math

def smallest\_subarray\_with\_given\_sum(s, arr):

window\_sum = 0

min\_length = math.inf

window\_start = 0

for window\_end in range(0, len(arr)):

window\_sum += arr[window\_end] # add the next element

# shrink the window as small as possible until the 'window\_sum' is smaller than 's'

while window\_sum >= s:

min\_length = min(min\_length, window\_end - window\_start + 1)

window\_sum -= arr[window\_start]

window\_start += 1

if min\_length == math.inf:

return 0

return min\_length

def main():

print("Smallest subarray length: " + str(smallest\_subarray\_with\_given\_sum(7, [2, 1, 5, 2, 3, 2])))

print("Smallest subarray length: " + str(smallest\_subarray\_with\_given\_sum(7, [2, 1, 5, 2, 8])))

print("Smallest subarray length: " + str(smallest\_subarray\_with\_given\_sum(8, [3, 4, 1, 1, 6])))

main()

Java:

class MinSizeSubArraySum {

public static int findMinSubArray(int S, int[] arr) {

int windowSum = 0, minLength = Integer.MAX\_VALUE;

int windowStart = 0;

for (int windowEnd = 0; windowEnd < arr.length; windowEnd++) {

windowSum += arr[windowEnd]; // add the next element

// shrink the window as small as possible until the 'windowSum' is smaller than 'S'

while (windowSum >= S) {

minLength = Math.min(minLength, windowEnd - windowStart + 1);

windowSum -= arr[windowStart]; // subtract the element going out

windowStart++; // slide the window ahead

}

}

return minLength == Integer.MAX\_VALUE ? 0 : minLength;

}

public static void main(String[] args) {

int result = MinSizeSubArraySum.findMinSubArray(7, new int[] { 2, 1, 5, 2, 3, 2 });

System.out.println("Smallest subarray length: " + result);

result = MinSizeSubArraySum.findMinSubArray(7, new int[] { 2, 1, 5, 2, 8 });

System.out.println("Smallest subarray length: " + result);

result = MinSizeSubArraySum.findMinSubArray(8, new int[] { 3, 4, 1, 1, 6 });

System.out.println("Smallest subarray length: " + result);

}

}