Code

Here is what our algorithm will look:

Java | Python3 | C++ | JS

```cpp
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;
}

```

Output 1.085s

```
Smallest subarray length: 2
Smallest subarray length: 1
Smallest subarray length: 3
```

Time Complexity

The time complexity of the above algorithm will be $O(N)$. The outer `for` loop runs for all elements and the inner `while` loop processes each element only once, therefore the time complexity of the algorithm will be $O(N + N)$ which is asymptotically equivalent to $O(N)$.

Space Complexity

The algorithm runs in constant space $O(1)$.