

Our Solution(s)

Run Code

Run Code

Your Solutions

Run Code

Run Code

Solution 1 Solution 2

```

1 // Copyright © 2020 AlgoExpert, LLC. All rights reserved.
2
3 package main
4
5 // O(log(n)) time | O(1) space
6 func SearchForRange(array []int, target int) []int {
7     finalRange := []int{-1, -1}
8     alteredBinarySearch(array, target, 0, len(array)-1, finalRange, true)
9     alteredBinarySearch(array, target, 0, len(array)-1, finalRange, false)
10    return finalRange
11 }
12
13 func alteredBinarySearch(array []int, target, left, right int, finalRange []int, goLeft bool) {
14     for left <= right {
15         mid := (left + right) / 2
16         if array[mid] < target {
17             left = mid + 1
18         } else if array[mid] > target {
19             right = mid - 1
20         } else {
21             if goLeft {
22                 if mid == 0 || array[mid-1] != target {
23                     finalRange[0] = mid
24                     return
25                 } else {
26                     right = mid - 1
27                 }
28             } else {
29                 if mid == len(array)-1 || array[mid+1] != target {
30                     finalRange[1] = mid
31                     return
32                 } else {
33                     left = mid + 1
34                 }
35             }
36         }
37     }
38 }

```

Solution 1 Solution 2 Solution 3

```
1 package main
2
3 func SearchForRange(array []int, target int) []int {
4     // Write your code here.
5     return nil
6 }
7
```

Our Tests

```

1  package main
2
3  import (
4      "fmt"
5      "os"
6      "strconv"
7      "strings"
8  )
9
10 func main() {
11     args := os.Args
12     if len(args) != 2 {
13         fmt.Println("Usage: ./calculator ")
14         os.Exit(1)
15     }
16     expression := args[1]
17     // Parse the expression
18     tokens := tokenize(expression)
19     // Evaluate the expression
20     result := evaluate(tokens)
21     fmt.Println(result)
22 }
23
24 func tokenize(expression string) []string {
25     tokens := []string{}
26     currentToken := ""
27     for i, char := range expression {
28         if char == ' ' {
29             if len(currentToken) > 0 {
30                 tokens = append(tokens, currentToken)
31                 currentToken = ""
32             }
33         } else {
34             currentToken += string(char)
35         }
36     }
37     if len(currentToken) > 0 {
38         tokens = append(tokens, currentToken)
39     }
40     return tokens
41 }
42
43 func evaluate(tokens []string) float64 {
44     // Simple calculator logic
45     // This is a placeholder for a more complex parser
46     // For example, you could use a stack to handle operator precedence
47     // or a recursive descent parser for more complex expressions
48     // For now, we'll just return the first token as a placeholder
49     if len(tokens) == 0 {
50         return 0
51     }
52     return strconv.ParseFloat(tokens[0], 64)
53 }

```

Custom Output

Submit Code

Submit Code

```
11 def isPrime(n):
12     if n < 2:
13         return False
14     for i in range(2, int(n**0.5) + 1):
15         if n % i == 0:
16             return False
17     return True
18
19 def sieve(n):
20     isPrime = [True] * (n + 1)
21     for i in range(2, int(n**0.5) + 1):
22         if isPrime[i]:
23             for j in range(i*i, n + 1, i):
24                 isPrime[j] = False
25     return [i for i, prime in enumerate(isPrime) if prime]
```

Run or submit code when you're ready.