```
@Test
fun popEmpty() {
    var output = ""
    try {
        problem1.pop( stackNum: 0)
    } catch(e: Exception){output = e.localizedMessage.toString()
    }
    assertEquals( expected: "Trying to pop an empty stack.",output)
}
@Test
fun push_data() {
    problem1.push( stackNum: 0,  value: 1)
    assertEquals( expected: 1,  problem1.peek( stackNum: 0))
}
@Test
fun checkIsEmpty() {
    var res = problem1.isEmpty( stackNum: 0)
    assertEquals( expected: true, res)
}
@Test
fun push_data2() {
    problem1.push( stackNum: 1,  value: 3)
    assertEquals( expected: 3,  problem1.peek( stackNum: 1))
}
@Test
fun get_Value() {
    problem1.push( stackNum: 1,  value: 3)
    assertEquals( expected: 3, problem1.peek( stackNum: 1))
}
@Test
fun pop_data() {
    problem1.push( stackNum: 0,  value: 2)
    problem1.pop( stackNum: 0)
    assertEquals( expected: 0, problem1.peek( stackNum: 1))
}
```

```kotlin
@Test
fun array_is_not_empty() {
    problem1.push( stackNum: 0,  value: 30)
    assertEquals( expected: false, problem1.isEmpty( stackNum: 0))
}
@Test
fun push_not_empty() {
    problem1.push( stackNum: 1,  value: 3)
    problem1.push( stackNum: 1,  value: 4)
    assertEquals( expected: 4, problem1.peek( stackNum: 1))
}

@Test
fun peek_out_of_range() {
    try {
        problem1.peek( stackNum: 1000)
    }
    catch (e: Exception)  {
        assertEquals( expected: "Index 1000 out of bounds for length 3", e.localizedMessage.toString())
    }
}

@Test
fun create_array() {
    var array = ""
    problem1.push( stackNum: 0,  value: 3)
    problem1.push( stackNum: 1,  value: 3)
    array += problem1.peek( stackNum: 0)
    array += problem1.peek( stackNum: 1)
    assertEquals( expected: "33", array)
}
```

```kotlin
class Problem2Test {
    private val problem2 = Problem2()
    private var array = arrayOf("random", "string", "int", "test", "char")
    private var array2 = arrayOf("random", "string", "int", "test", "char")
    private val emptyArray : Array<String> = emptyArray()
    private val intArray = arrayOf("10", "12", "16")
    private val charArray = arrayOf("a", "b", "c")

    @Test
    fun search_string() {
        var result = problem2.search(array, str: "int")
        assertEquals( expected: 2, result)
    }

    @Test
    fun search_bad_string() {
        var result = problem2.search(array, str: "bad")
        assertEquals( expected: -1, result)
    }

    @Test
    fun empty_array() {
        var result = problem2.search(emptyArray, str: "bad")
        assertEquals( expected: -1, result)
    }
    @Test
    fun search_null_string() {
        var result = problem2.search(array, str: null)
        assertEquals( expected: -1, result)
    }
    @Test
    fun search_empty_string() {
        var result = problem2.search(array, str: "")
        assertEquals( expected: -1, result)
    }
```

```
@Test
fun search_empty_variable() {
    var emptyString = ""
    var result = problem2.search(array, emptyString)
    assertEquals( expected: -1, result)
}
@Test
fun search_null_variable() {
    var nullVariable = null
    var result = problem2.search(array, nullVariable)
    assertEquals( expected: -1, result)
}

@Test
fun search_Int_array() {
    var result = problem2.search(intArray,  str: "12")
    assertEquals( expected: 1, result)
}

@Test
fun search_Int_array2() {
    var result = problem2.search(intArray,  str: "10")
    assertEquals( expected: 0, result)
}

@Test
fun search_char_array() {
    var result = problem2.search(charArray,  str: "c")
    assertEquals( expected: 2, result)
}
```