100

Venkata Naga Sri Sai Pranavi Kolipaka

Other

PDF generated at: 18 Jul 2025 21:42:44 UTC

View this report on HackerRank

Score

100% • 80 / 80

scored in TIP102: Unit 7 Version A (Standard) - Summer 2025 in 53 min 6 sec on 18 Jul 2025 13:47:51 PDT

Candidate Information

Email	kolipakavnssaipranavi@gmail.com
Test	TIP102: Unit 7 Version A (Standard) - Summer 2025
Candidate Packet	View
Taken on	18 Jul 2025 13:47:51 PDT
Time taken	53 min 6 sec/ 90 min
Personal Member ID	129054
Email Address with CodePath	kolipakavnssaipranavi@gmail.com
Github username with CodePath	Pranavi2002
Invited by	CodePath

Suspicious Activity detected

Code similarity

Code similarity • 2 questions

Skill Distribution

There is no associated skills data that can be shown for this assessment

Tags Distribution

There is no associated tags data that can be shown for this assessment

Questions

Coding Questions • 60 / 60

Status	No.	Question	Time Taken	Skill	Score	Code Quality
✓	1	Compute Power Function Coding	3 min 51 sec	-	20/20	-

✓	2	Bad Product Coding	25 min 51 sec	-	20/20	-
✓	3	First and Last Position of Element in Sorted Array Coding	14 min 55 sec	-	20/20	-

Multiple Choice + Debugging • 20 / 20

Status	No.	Question	Time Taken	Skill	Score	Code Quality
✓	4	What is the time complexity of mystery_function()? Multiple Choice	1 min	-	5/5	-
✓	5	What is the output of the following code snippet? Multiple Choice	1 min 57 sec	-	5/5	-
✓	6	What is the output of the following code snippet? Multiple Choice	2 min 51 sec	-	5/5	-
✓	7	Debug this code Coding	1 min 20 sec	-	5/5	-

1. Compute Power Function

✓ Correct

Coding

Question description

Given two integers, x and n, where n is non-negative, write a function `power()` that recursively computes and returns x^n

Example 1:
Input: x = 2, n = 3
Output: 8
Explanation: 2^3 = 2 * 2 * 2 = 8

Example 2:
Input: x = 5, n = 0
Output: 1
Explanation: 5^0 = 1

Example 3:
Input: x = 3, n = 2
Output: 9
Explanation: 3^2 = 3 * 3 = 9

Candidate's Solution

Language used: Python 3

```
#!/bin/python3

import math
import os
import random
import re
import sys
import ast


#
# Complete the 'power' function below.
#
# The function is expected to return an INTEGER.
# The function accepts following parameters:
```

```
#  1. INTEGER x
#  2. INTEGER n
#
# Psuedo-code:
# similar to finding the exponent of a number
# if n is 0, then return 1 since any number power 0 is always 1
# else, multiply the number x as many times as n with itself and return the final number

def power(x, n):
    # Write your code here
    if n == 0:
        return 1
    else:
        return x * power(x, n-1)


if __name__ == '__main__':
    outfile = open(os.environ['OUTPUT_PATH'], 'w')
    input_data = sys.stdin.read().strip().splitlines()

    results = []

    for line in input_data:
        x, n = eval(line)
        result = power(x, n)
        results.append(result)

    for res in results:
        outfile.write(str(res) + '\n')
    outfile.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Base case: Zero exponent	Easy	Hidden	Success	0	0.0322 sec	10.8 KB
Base case: Exponent of 1	Easy	Hidden	Success	0	0.0278 sec	10.9 KB

Small base and exponent	Easy	Hidden	Success	0	0.0345 sec	10.9 KB
Large base and small exponent	Easy	Hidden	Success	0	0.0268 sec	10.9 KB
Small base and large exponent	Easy	Hidden	Success	0	0.0363 sec	11 KB
Negative base with even exponent	Easy	Hidden	Success	0	0.0284 sec	10.8 KB
Negative base with odd exponent	Easy	Hidden	Success	0	0.0268 sec	10.9 KB
Negative base with zero exponent	Easy	Hidden	Success	0	0.031 sec	10.6 KB
Base is zero	Easy	Hidden	Success	0	0.0281 sec	10.9 KB
Base is zero and exponent is zero (0^0)	Easy	Hidden	Success	0	0.0288 sec	10.9 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0308 sec	10.9 KB

No comments.

2. Bad Product

Correct

Coding

Question description

You are given a list of characters `letters` that is sorted in **non-decreasing order**, and a character `target`. There are **at least two different** characters in `letters`.

Return *the smallest character in* `letters` *that is lexicographically greater (occurs later in the alphabet) than* `target`. If such a character does not exist, return the first character in `letters`.

Your solution must have `O(log n)` time complexity.

Example Input: letters = ["c","f","j"], target = "a"
Expected Output: "c"
Explanation: The smallest character that is lexicographically greater than 'a' in letters is 'c'.

Example Input: letters = ["x","x","y","y"], target = "z"
Expected Output: "x"
Explanation: There are no characters in letters that is lexicographically greater than 'z' so we return letters[0].

Candidate's Solution

Language used: **Python 3**

```
#!/bin/python3

import math
import os
import random
import re
import sys
import ast


#
# Complete the 'next_greatest_letter' function below.
#
```

```
14 # The function is expected to return a STRING.
15 # The function accepts following parameters:
16 #  1. STRING_ARRAY letters
17 #  2. STRING target
18 #
19 # Pseudo-code:
20 # similar to binary search
21 # take two variables low and high initially first and last value in the list
22 # loop until high becomes greater than low
23 # take mid variable which is the middle value
24 # if letters[mid] > target, then search in the left side of the mid, make
   high = mid - 1
25 # if letters[mid] <= target, then search in the right side of the mid, make
   low = mid + 1
26 # low is now the index of the smallest letter > target
27 # if no such letter, return letters[0]
28
29 def next_greatest_letter(letters, target):
30     # Write your code here
31     low = 0
32     high = len(letters) - 1
33     while low <= high:
34         mid = (low + high) // 2
35         if letters[mid] > target:
36             high = mid - 1
37         else:
38             low = mid + 1
39
40     return letters[low % len(letters)]
41
42 if __name__ == '__main__':
43     outfile = open(os.environ['OUTPUT_PATH'], 'w')
44     input_data = sys.stdin.read().strip().splitlines()
45
46     results = []
47
48     for line in input_data:
49         letters, target = eval(line)
50         result = next_greatest_letter(letters, target)
51         results.append(result)
52
53     for res in results:
54         outfile.write(str(res) + '\n')
55     outfile.close()
56
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Basic Case	Easy	Hidden	Success	0	0.0305 sec	10.9 KB
No characters in letters that is lexicographically greater than target	Easy	Hidden	Success	0	0.0272 sec	11 KB
Target Matches a Letter in the List	Easy	Hidden	Success	0	0.0304 sec	10.9 KB
Target is Greater than All Letters	Easy	Hidden	Success	0	0.0289 sec	10.9 KB
All Letters are the Same	Easy	Hidden	Success	0	0.0295 sec	10.9 KB
Single Letter in the List	Easy	Hidden	Success	0	0.0273 sec	10.9 KB
Target is Equal to the Last Letter	Easy	Hidden	Success	0	0.0319 sec	10.8 KB
Target Falls Between Two Letters	Easy	Hidden	Success	0	0.0293 sec	10.9 KB
Large List with Repeated Characters	Easy	Hidden	Success	0	0.0287 sec	10.9 KB

Target is Smaller than All Letters in the List	Easy	Hidden	Success	0	0.0349 sec	11 KB
Wrap Around Case	Easy	Hidden	Success	0	0.027 sec	10.9 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0286 sec	10.9 KB

No comments.

3. First and Last Position of Element in Sorted Array

Correct

Coding

Question description

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given target value.

If the target is not found in the array, return `[-1, -1]`.

You must write an algorithm with O(log n) runtime complexity.

Example 1:
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]

Example 2:
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]

Example 3:

Input: nums = [], target = 0
Output: [-1,-1]

Candidate's Solution

Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys
import ast


#
# Complete the 'search_range' function below.
#
# The function is expected to return an INTEGER.
# The function accepts following parameters:
#  1. INTEGER_ARRAY nums
#  2. INTEGER target
#

def search_range(nums, target):
    # Write your code here
    def first_pos(is_first):
        low = 0
        high = len(nums) - 1
        pos = -1
        while low <= high:
            mid = (low + high) // 2
            if nums[mid] == target:
                pos = mid
                if is_first:
                    high = mid - 1
                else:
                    low = mid + 1
            elif nums[mid] < target:
                low = mid + 1
            else:
                high = mid - 1
        return pos
```

```
    first = first_pos(True)
    last = first_pos(False)
    return [first, last]


if __name__ == '__main__':
    outfile = open(os.environ['OUTPUT_PATH'], 'w')
    input_data = sys.stdin.read().strip().splitlines()

    results = []

    for line in input_data:
        nums, target = eval(line)
        result = search_range(nums, target)
        results.append(result)

    for res in results:
        outfile.write(str(res) + '\n')
    outfile.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Basic Case	Easy	Hidden	Success	0	0.03 sec	10.9 KB
Basic Case	Easy	Hidden	Success	0	0.0279 sec	10.9 KB
Single Element Matching Target	Easy	Hidden	Success	0	0.0324 sec	10.9 KB
All Elements Match Target	Easy	Hidden	Success	0	0.0311 sec	10.9 KB
Target at the Beginning	Easy	Hidden	Success	0	0.0248 sec	10.9 KB

Target at the End	Easy	Hidden	Success	0	0.0303 sec	10.8 KB
Empty Array	Easy	Hidden	Success	0	0.03 sec	10.9 KB
No Target Found	Easy	Hidden	Success	0	0.0258 sec	10.9 KB
Multiple Occurrences of Target	Easy	Hidden	Success	0	0.0304 sec	10.9 KB
Target Less than All Elements	Easy	Hidden	Success	0	0.0283 sec	10.9 KB
Target Greater than All Elements	Easy	Hidden	Success	0	0.0271 sec	10.9 KB
Pass/Fail Case	Easy	Hidden	Success	20	0.0287 sec	10.9 KB

No comments.

4. What is the time complexity of mystery_function()?

Correct

Multiple Choice

Question description

What is the time complexity of `mystery_function()`?

```
def mystery_function(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ O(1)
- ○ O(log n)
- ● O(n) ✓
- ○ O(n^2)

ⓘ No comments.

5. What is the output of the following code snippet?

✓ Correct

Multiple Choice

Question description

```
def mystery_function(arr):
    if not arr:
        return 0
    return arr[0] + mystery_function(arr[1:])

print(mystery_function([1, 2, 3, 4, 5]))
print(mystery_function([10, 20, 30]))
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ 10 60<!-- notionvc: 2c7c0493-507f-437f-8f8c-2a77373027dd -->
- ○ 15 70<!-- notionvc: 3cb76204-901b-49aa-831c-8b65b6688e69 -->
- ● ** 15 60<!-- notionvc: 7601423a-3ea5-4a30-88f5-0c3cc346cd88 --> ** ✓
- ○ 15 50<!-- notionvc: 491d88d7-5cc3-42dd-8958-d940a17fc43f -->

No comments.

6. What is the output of the following code snippet?

Correct

Multiple Choice

Question description

```
from collections import deque

def recursive_helper(queue):
    if not queue:
        return 0
    current = queue.popleft()
    return current + recursive_helper(queue)


def sum_with_queue(arr):
    queue = deque(arr)
    return recursive_helper(queue)

print(sum_with_queue([1, 2, 3, 4, 5]))
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ 10
- ○ 12
- ○ 16
- ● 15 ✓

No comments.

7. Debug this code

Correct

Coding

Question description

The code provided below incorrectly implements the `binary_search()` function. Implemented correctly, `binary_search()` accepts a unique list of integers `nums` and an integer `target` and returns the index of `target` in the list. If `target` is not a value in the list, the function should return -1.

Identify any bug(s) within the given implementation and correct the code so that it successfully passes the provided test cases.

Candidate's Solution

Language used: Python 3

```python
#!/bin/python3

import math
import os
import random
import re
import sys
import ast


def binary_search(nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1
if __name__ == '__main__':
    input_data = sys.stdin.read().strip()
    input_list = ast.literal_eval(input_data)
```

```
27
28      nums = input_list[0]
29      target = input_list[1]
30
31      result = binary_search(nums, target)
32      print(result)
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Pass/Fail Case	Easy	Hidden	Success	5	0.0291 sec	11 KB

No comments.