Data Structures

Homework Assignment 2 - Analysis of Algorithms

Problem 1 - Nearest Temperature - 25 Points
Problem 2 - Lists Product - 25 Points
Problem 3 - Longest Substring Length - 25 Points
Problem 4 - Median of Two Sorted Arrays - 25 Points

25% of Gradescope Autograder test cases are hidden for this assignment.

Fall 2024 - Revision 1 - *September*

Problem 1 - Nearest Temperature - 25 Points

This problem is designed to enhance your understanding of the binary search algorithm and its application in finding the closest neighbor (element) in a sorted array. Write the program *find_nearest_temperature, which* finds the temperature closest to a given target temperature in a sorted array of temperatures. In case of a tie, choose the smaller temperature.

Example

```
def main():
  sorted_temps = [-20, -15, -5, 3, 8, 12, 30, 45, 50, 60]
  target_temp = 7

  nearest = find_nearest_temperature(sorted_temps, target_temp)
  print(nearest)  # should print 8
```

Requirements

- Implement the solution using the binary search algorithm.
- The input list `sorted_temps` is guaranteed to be sorted in ascending order.
- You are not allowed to use any Python built-in search functions.
- Handle edge cases where the target temperature is less than the smallest temperature, greater than the largest temperature in the array, or no temperature exists.

Time Complexity

- **O(log n)**, where *n* is the number of temperatures in the list.

Space Complexity

- **O(1)**, as the space used does not depend on the input size.

Problem 2 - Lists Product - 25 Points

Given two lists, *A* and *B*, of length *n*, storing integers > 0 in ascending order. Implement the function `product_checker(A, B, m)` to find the product *m* of two elements so that *(a,b): a* $\in A$ and *b* $\in B$, and *ab* = *m*. Return your result as a Python list of tuples. If there are two combinations $a \in A$ that all match to one (or duplicates of) $b \in B$, then return one pair *(a,b)* (please see example 3 below). Your resulting list has to be sorted ascending for every $a \in A$.

Example 1

```
A = [2, 4, 5, 6, 8, 10, 12]
B = [1, 2, 4, 9, 10, 20]
res = product_checker(A, B, 40)
print(res)  # Should print [(2, 20), (4, 10), (10, 4)]
```

Example 2

```
A = [1, 4, 5, 6, 20]
B = [1, 2, 4, 10]
res = product_checker(A, B, 100)
print(res)  # Should print: []
```

Example 3

```
A = [1, 2, 2, 3, 5]
B = [1, 5, 50, 50, 100] # here 50 is duplicated
res = product_checker(A, B, 100)
print(res)  # Should print: [(1, 100), (2, 50)]
```

Requirements

- You can not use any third-party tools or libraries for solving this task.

Time Complexity

- **O(nlogn)**, where *n* is the number of elements in *A* and *B*.

Space Complexity

- **O(n)**, where *n* is the space claimed through your algorithm.

Problem 3 - Longest Substring Length - 25 Points

Given a string *s*, find the length of the longest substring without repeating characters.
For simplicity, the string *s* only consists of lowercase English letters *(a-z)*.

Example 1

```
s = “abcabcbb”
res = lengthOfLongestSubstring(s)
print(res)  # Length of the substring “abc”, should print 3
```

Example 2

```
s = “bbbbb”
res = lengthOfLongestSubstring(s)
print(res)  # Length of the substring “b”, should print 1
```

Example 3

```
s = “pwwkew”
res = lengthOfLongestSubstring(s)
print(res)  # Length of the substring “wke”, should print 3
            # Note that substring is not subsequence
```

Requirements

- You can not use any third-party tools or libraries for solving this task.

Time Complexity

- **O(n)**, where *n* is the length of the string *s*.

Problem 4 - Median of Two Sorted Arrays - 25 Points

Given two sorted arrays without duplication *nums1* and *nums2* of size *m* and *n*, respectively. Find the median of the two sorted arrays.

Example 1

```
nums1 = [1, 3, 5, 7]
nums2 = [2]
result = findMedianSortedArrays(nums1, nums2)
print(result)  # should print 3
```

Example 2

```
nums1 = [1, 2]
nums2 = [4, 5]
result = findMedianSortedArrays(nums1, nums2)
print(result)  # Median is (2 + 4) / 2 = 3
               # Should print 3
```

Requirements

- You can not use any third-party tools or libraries for solving this task.

Time Complexity

- **O(min(m,n))**, where *m,n* are the length of the arrays *num1, num2*.