# horizontal lineData 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

**Notes and Requirements**

* Your submission must be your effort. You can not copy other students' code.
* This worksheet is graded on performance; Implementations must be correct.
* You are encouraged to visit our office hours to ask coding questions.
* Only the latest (most recent) submission is graded.
* Late submissions are not considered for grading.
* You can not use any third-party libraries.

**Some assignments on this worksheet are manually graded.**

## 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* that 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 ∈ 𝐴* and *b ∈ 𝐵*, and *ab = m*. Return your result as a Python list of tuples. If there are two combinations *𝑎 ∈ 𝐴* that all match to one (or duplicates of) *𝑏 ∈ 𝐵*, then return one pair *(a,b)* (please see example 3 below). Your resulting list has to be sorted ascending for every *𝑎 ∈ 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**

* 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

**Requirements**

* 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**

* Time complexity **O(log(m+n))**, where *m,n* are the length of the arrays *num1, num2*.