Leetcode: Data Structure and Algorithms

Rafiq Islam

Table of contents

## Binary Search

|  |
| --- |
| **Problem 1: Leetcode 69: Sqrt(x)**  Given a non-negative integer, , return the square root of rounded down to the nearest integer. The returned integer should be non-negative as well.  You may not use any built-in exponent function. For example, x\*\*0.5 in python.  Example:  Input: x=4 Output: 2   Input: x=8 Output: 2  Explanation: Square root of 4 is 2 and square root of 8 is 2.8284. But we need to round down to any fraction. Therefore, the square root of 8 is also 2.  **Solution:**  The square root of any number is less than or equal to . The brute force solution to this would be . Because, say , then  for to 8:    In contrast, if we explore binary search then the time complexity reduces to . Say the square root is which is the middle value in the range of 1 to . Then if , we search for the root in the left half. Otherwise, if then we search the right side. However, when , then is a possible candidate for the square root.  *Algorithm:*   1. set left value , right value 2. Compute the middle value 3. If then search the left side: set 4. If then search the right side: set   def square\_root(x):  l, r = 0, x   sq = 0  while l<=r:  m = l + (r-l)//2   if m\*\*2 > x:  r= m-1  elif m\*\*2 < x:  l = m+1   sq = m  else:  return m   return sq    print(square\_root(6))  2 |

## Array

|  |
| --- |
| **Problem 1: Intersection of two sets**  Say, we are given two sets and . We want to find the intersection of the elements in these sets.  def intersection\_of\_two\_sets(A,B):  set\_a = set(A)  return [b for b in B if b in set\_a]  A = [2,3,5,6,8] B = [4,6,8] print(intersection\_of\_two\_sets(A,B))  [6, 8] |

|  |
| --- |
| **Problem 2: Histogram from a given array and bin number**  Say, we are given an array and number of bins. We want to  def generate\_histogram(A, num\_bins):  min\_value = min(A)  max\_value = max(A)  bin\_width = (max\_value-min\_value)/num\_bins |

## String

|  |
| --- |
| **Problem 1: Leetcode 242: Valid Anagram**  Given two strings s and t, return true if t is an *anagram* of s, and false *otherwise*  Example: Input: s="anagram", t="nagaram" Output: true  Since the word anagram has 3 a’s, 1 n, 1 g, 1 r, and 1 m and nagaram has exactly the same number of the same alphabets, therefore they are anagram of each other.  Example: Input: s="rat", t="cat" Output: false  Since the word rat has 1 r, 1 a, and 1 t but cat has 2 elements same as rat but one element different. Therefore the answer is false.  Note that, they both have the same length.  **Constraints:**   * .length, .length * s and t consists of lowercase English letters   **Solution:** We can use hasmap to solve this problem. Basically, we will create two hasmaps for two words and match the keys and values of the hasmaps. If they are equal then it’s an anagram, otherwise not.  def isAnagram(s,t):  if len(s) != len(t):  return False     hash\_s, hash\_t = {}, {}  for i in range(len(s)):  hash\_s[s[i]] = 1 + hash\_s.get(s[i],0) # get function collects the key and values.  hash\_t[t[i]] = 1 + hash\_t.get(t[i],0) # if there's no key, 0 is the default value    for c in hash\_s:  if hash\_s[c] != hash\_t.get(c,0): # Here, get function ensures there is no   return False # key error  return True  s = "anagram" t = "nagaram"  print(isAnagram(s,t))  u = "rat"  print(isAnagram(s,u))  True False  **Time and Space Complexity:** Time complexity where and are the length of s and t and memory complexity is the same  Optimization: Can we solve the problem in ? If we assume sort doesn’t require extra space, then  def isAnagram(s,t):  return sorted(s)== sorted(t)  s= "anagram" t= "nagaram"  print(isAnagram(s,t))  True |

## Reference

All my solutions here are based on the solutions found from [NeetCode](https://www.youtube.com/c/neetcode).