1. Find the start page and end page
   We will calculate each point's indegree and outdegree. In the end, if its outdegree - indegree == 1, it's start point, if it's -1, it's end point.
   TC is O(n)

```
from collections import defaultdict
def findFirstAndLastPage(arr):
 memo_indegree = defaultdict(int)
 memo_outdegree = defaultdict(int)
 start_points, end_points = [], []
 points = set()
 for start, end in arr:
   memo_indegree[end] += 1
   memo_outdegree[start] += 1
   points.add(start)
   points.add(end)

 for point in points:
   if memo_indegree[point] - memo_outdegree[point] == 1:
     end_points.append(point)
   elif memo_indegree[point] - memo_outdegree[point] == -1:
     start_points.append(point)
```

2. 208. Implement Trie (Prefix Tree)
   We will use dict to implement Trie, every node has children and word mark isEndOfWord. When we insert the word, in the end, we will set isEndOfWord to True, TC is O(n)

```
from collections import defaultdict
class TrieNode:
    def __init__(self):
        self.children = defaultdict(TrieNode)
        self.isEndOfWord = False

class Trie:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.root = TrieNode()
    def insert(self, word: str) -> None:
        """
        Inserts a word into the trie.
        """
        cur = self.root
        for w in word:
            cur = cur.chilren[w]
        cur.isEndOfWord = True

    def search(self, word: str) -> bool:
        """
        Returns if the word is in the trie.
        """
        cur = self.root
        for w in word:
            if w not in cur.children:
                return False
            cur = cur.children[w]
        return cur.isEndOfWord

    def startsWith(self, prefix: str) -> bool:
        """
        Returns if there is any word in the trie that starts with the given prefix.
        """
        cur = self.root
        for w in prefix:
            if w not in cur.children:
                return False
            cur = cur.children[w]
        return True
```

3. Match Users randomly
   We will get two random index and check they are not from the same team. Then we will replace these two elements with last two elements in the array. Each time we will reduce our endIndex by 2. TC is O(n)

```
from random import *
def getRandomUsers(users, endIndex):
 if len(set(map(lambda a: a["team"], users[:endIndex + 1]))) < 2:
   return False
 while True:
   index1, index2 = sample(range(endIndex + 1), 2)
   if users[index1]['team'] != users[index2]['team']:
     users[index1], users[endIndex] = users[endIndex], users[index1]
     users[index2], users[endIndex - 1] = users[endIndex - 1],
users[index2]
     return(users[endIndex]['id'], users[endIndex - 1]['id'])

def matchUsers(users):
 length = len(users)
 endIndex = length - 1
 result = []
 for _ in range(length // 2):
   pair_users = getRandomUsers(users, endIndex)
   if pair_users:
     result.append(pair_users)
   else:
     break
   endIndex -= 2
 return result
```

4. Valid Square
   We will check whether four sides are equal and diagonal**2 == side ** 2 * 2, then it's square. TC is O(1)

```
class Solution:
  def validSquare(self, p1: List[int], p2: List[int], p3: List[int], p4: List[int]) -> bool:
    d1 = self.getDis(p1, p2)
    d2 = self.getDis(p1, p3)
    d3 = self.getDis(p1, p4)
    if d1 == 0 or d2 == 0 or d3 == 0:
      return False

    if (d1 == d2 and d1 * 2 == d3 and self.getDis(p4, p2) == self.getDis(p3, p4) == d1)
or (d1 == d3 and d1 * 2 == d2 and self.getDis(p3, p4) == self.getDis(p3, p2) == d1) or
(d3 == d2 and d2 * 2 == d1 and self.getDis(p2, p4) == self.getDis(p2, p3) == d2):
      return True
    return False

  def getDis(self, a, b):
    return (a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2
```

5. Best Time to Buy and Sell Stock
   We will always remember the min element and compare with the current element and get maximum difference. TC is O(n)
   ```
   class Solution:
       def maxProfit(self, prices: List[int]) -> int:
           min_num = float('inf')
           result = 0
           for price in prices:
               if min_num > price:
                   min_num = price
               else:
                   result = max(result, price - min_num)
           return result
   ```