2347. Best Poker Hand

Hash Table

Counting

type starting from the best ranking hand to the worst.

Problem Description

Easy

arrays: one for the ranks of the cards (ranks) and one for the suits (suits). The task is to determine the best poker hand from the following options, ranked from best to worst: "Flush", "Three of a Kind", "Pair", and "High Card".

The problem involves simulating the evaluation of a poker hand with given ranks and suits of cards. You are provided with two

A "Flush" is when all five cards have the same suit. A "Three of a Kind" is when three of the cards have the same rank. A "Pair" is when two of the cards have the same rank. If none of these hands are possible, you have a "High Card", which is your hand's highest ranking card.

You need to identify which of these hands you can form with your cards and return a string that represents the best possible hand.

The intuition behind approaching this solution is to categorize the poker hand hierarchies and check for the presence of each

Intuition

The best hand we can have is a "Flush". Since a "Flush" requires all cards to be of the same suit, we can simply check if the

- suits array contains the same suit for every card. If this condition is met, we can return "Flush" right away without checking for other possibilities because "Flush" is the highest-ranked hand we're considering in this problem. Next, we can look for "Three of a Kind". Counting the occurrence of each rank in the ranks array helps us to identify if there
- are any three cards of the same rank. If we find that any rank appears at least three times, we have "Three of a Kind". If "Three of a Kind" doesn't exist, we move on to check for a "Pair". Similarly to the previous step, if we find that any rank
- appears exactly twice, we have a "Pair". Lastly, if none of the above hands are formed, by default we have a "High Card". There's no need to identify which card it is,
- **Solution Approach**

Set and Frequency Counting: • Flush Check: To identify a "Flush," we're looking for the uniqueness of suits. If all suits are the same, the set of suits would have a length of

1. However, instead of converting suits into a set, which has a time complexity of O(n), an optimized approach checking all(a == b for a, b in pairwise(suits)) is used. This uses the pairwise function from Python's itertools module to check if every adjacent pair of

since "High Card" simply refers to the situation where none of the other hands are possible.

The implementation of the solution uses a few fundamental algorithms, data structures, and patterns:

elements is the same. If they are all the same, it's a flush.

- Frequency Counting: To identify "Three of a Kind" and "Pair", we can use counting to track the frequency of each rank. This is done with Counter from the collections module. The Counter object, cnt, maps each rank to the number of times it appears in the ranks list. **Conditional Logic:** • The checks are done in a particular order, from the best hand to the worst. This is done using a series of if statements.
- First, the "Flush" is checked. If the "Flush" condition is met, the function immediately returns 'Flush', since we do not need to check for other hand types.

If the condition for "Flush" is not met, the frequency count (cnt) is used to check if there's any rank that appears at least 3 times for "Three

of a Kind".

cards.

class Solution:

cnt = Counter(ranks)

- If there is no "Three of a Kind", the function then checks for "Pair" by looking for any rank that appears exactly twice in the cnt. o If neither of those hands are possible, the function returns 'High Card' by default as it's the lowest hand that can be made with any set of
- This approach uses efficient data structures to minimize the time complexity and leverages the power of the Python standard library for counting and pairwise comparison to simplify the logic. The solution encapsulates each of these checks within a single class method bestHand, which takes two arguments: ranks and
 - # Check for Flush if all(a == b for a, b in pairwise(suits)): return 'Flush'

Check for Three of a Kind if any(v >= 3 for v in cnt.values()): return 'Three of a Kind' # Check for Pair

```
if any(v == 2 for v in cnt.values()):
             return 'Pair'
        # If none of the above, return High Card
        return 'High Card'
  This code provides an efficient solution to the problem by methodically checking for each type of poker hand in decreasing order
  of rank and is an example of how understanding the domain (poker hands ranking) aids in crafting a clear and concise algorithm.
Example Walkthrough
  Let's consider an example where you have the following hand of cards:
 • Ranks: [10, 7, 10, 4, 3]
 • Suits: ['H', 'D', 'H', 'S', 'H']
```

suits, corresponding to the ranks and suits of the cards in the poker hand.

def bestHand(self, ranks: List[int], suits: List[str]) -> str:

Flush Check: We first check if there's a "Flush". We compare each suit with the next one by pairwise comparison: ∘ 'H' == 'D'? No. Since not all suits are the same, it's not a "Flush". We move on to the next check.

Create a frequency count of the ranks using Counter: Counter([10, 7, 10, 4, 3]) results in {10: 2, 7: 1, 4: 1, 3: 1}

Python

Java

class Solution {

Since no rank has a frequency of 3 or more, we do not have a "Three of a Kind". Since there's no "Three of a Kind", we move on to check for a "Pair":

We do not proceed any further since we have already identified a "Pair", which takes precedence over "High Card".

Final Hand Determination:

■ We find that the rank 10 appears twice (10: 2).

This confirms that we have a "Pair" for this hand.

Let's walk through the solution approach step by step:

Frequency Counting for "Three of a Kind" and "Pair":

Check for "Three of a Kind" by looking for any rank that appears three times:

cards. Solution Implementation

class Solution: def bestHand(self, ranks: List[int], suits: List[str]) -> str: # Check if all suits are the same by comparing each pair of adjacent suits. # If so, return 'Flush' since all cards have the same suit. if all(suit1 == suit2 for suit1, suit2 in pairwise(suits)):

Thus, according to our solution approach, the function bestHand would return 'Pair' as the best hand possible with the given

Check if there's any rank with at least three occurrences. # If so, return 'Three of a Kind'. if any(count >= 3 for count in rank_counter.values()): return 'Three of a Kind'

from typing import List

from collections import Counter

return 'Flush'

If so, return 'Pair'.

return 'Pair'

return 'High Card'

rank_counter = Counter(ranks)

Use a Counter to count the occurrences of each rank.

Check for any rank with exactly two occurrences.

string bestHand(vector<int>& ranks, vector<char>& suits) {

// Initialize an array to count occurrences of each rank.

for (int i = 1; $i < 5 \&\& isFlush; ++i) {$

isFlush = suits[i] == suits[i - 1];

// Check if all the suits are the same, which would mean a Flush.

bool hasPair = false; // Flag to check if there is at least one pair.

// If a rank appears twice, we mark that we have found a pair.

// Return "Pair" if a pair was found, otherwise return "High Card".

// If a rank appears three times, it is Three of a Kind.

// Count the occurrences of each rank and check for Three of a Kind or Pair.

if any(count == 2 for count in rank_counter.values()):

If none of the above conditions are met, return 'High Card'.

// Method to determine the best hand from the given ranks and suits of cards

from itertools import pairwise

```
public String bestHand(int[] ranks, char[] suits) {
   // Initially assume we have a flush (all suits are the same)
   boolean isFlush = true;
   // Check all card suits; if any are different, set isFlush to false
   for (int i = 1; i < 5 && isFlush; ++i) {
       if (suits[i] != suits[i - 1]) {
           isFlush = false;
   // If all card suits are the same, we have a flush
   if (isFlush) {
        return "Flush";
   // Counter for the occurrences of each rank
   int[] rankCount = new int[14];
   // Flag to indicate if at least one pair has been found
   boolean hasPair = false;
   // Iterate over all the ranks to count occurrences and identify pairs or three of a kind
    for (int rank : ranks) {
        rankCount[rank]++;
       // If a rank count reaches 3, we have a three of a kind
       if (rankCount[rank] == 3) {
           return "Three of a Kind";
       // If a rank count is exactly 2, note that we have a pair
       if (rankCount[rank] == 2) {
           hasPair = true;
   // Return the best hand based on whether we've found a pair or have only high card
   return hasPair ? "Pair" : "High Card";
```

}; **TypeScript**

C++

public:

class Solution {

bool isFlush = true;

return "Flush";

int rankCounts[14] = {0};

for (int& rank : ranks) {

rankCounts[rank]++;

if (rankCounts[rank] == 3) {

return "Three of a Kind";

return hasPair ? "Pair" : "High Card";

hasPair = hasPair || rankCounts[rank] == 2;

def bestHand(self, ranks: List[int], suits: List[str]) -> str:

Use a Counter to count the occurrences of each rank.

if any(count == 2 for count in rank_counter.values()):

If so, return 'Flush' since all cards have the same suit.

if all(suit1 == suit2 for suit1, suit2 in pairwise(suits)):

Check if there's any rank with at least three occurrences.

If none of the above conditions are met, return 'High Card'.

Check if all suits are the same by comparing each pair of adjacent suits.

if (isFlush) {

```
function bestHand(ranks: number[], suits: string[]): string {
      // Check for a Flush: all suits are the same
      if (suits.every(suit => suit === suits[0])) {
          return 'Flush';
      // Initialize a counter array to hold the frequency of each rank
      const rankCounts = new Array(14).fill(0);
      let hasPair = false; // Flag to check if a Pair has been found
      // Loop through the ranks to count occurrences of each rank
      for (const rank of ranks) {
          rankCounts[rank]++;
          // If a rank count reaches 3, we have a 'Three of a Kind'
          if (rankCounts[rank] === 3) {
              return 'Three of a Kind';
          // Check if we have at least one pair
          hasPair = hasPair || rankCounts[rank] === 2;
      // If a pair was found, return 'Pair'
      if (hasPair) {
          return 'Pair';
      // If none of the above hands are found, return 'High Card'
      return 'High Card';
from typing import List
from collections import Counter
from itertools import pairwise
```

if any(count >= 3 for count in rank_counter.values()): return 'Three of a Kind' # Check for any rank with exactly two occurrences. # If so, return 'Pair'.

return 'Flush'

return 'Pair'

Time and Space Complexity

return 'High Card'

number of suits.

case time complexity of O(m).

rank_counter = Counter(ranks)

If so, return 'Three of a Kind'.

provided. Here is an analysis of its complexity:

class Solution:

Time Complexity: all(a == b for a, b in pairwise(suits)): This checks if all elements in suits are the same. Assuming pairwise is an iterable that provides tuples of successive pairs from suits, this operation has a time complexity of O(n), where n is the

Counter(ranks): Counting the frequency of each rank has a time complexity of O(m), where m is the number of ranks.

any(v >= 3 for v in cnt.values()): Iterating over the values of the counter to check for a 'Three of a Kind' has a worst-

The given Python function bestHand determines the best hand possible in a card game based on the suits and ranks of the cards

Since the number of cards in a hand is typically small and fixed (for example, 5 in many games), both n and m can be considered constants, and the time complexity can be simplified to 0(1).

any(v == 2 for $v in cnt_values())$: Similarly, this check for a 'Pair' has a time complexity of O(m).

Space Complexity:

Counter(ranks): The counter here creates a dictionary with a unique entry for each rank. The space complexity is O(m) since

Temporal space needed to store the pairs in pairwise(suits): Since only two elements from suits are considered at a time, the extra space is 0(1). No additional data structures with significant space requirements are used.

it stores as many entries as there are unique ranks.

Like the time complexity, as the hand size is fixed, m can be considered a constant, simplifying the space complexity to 0(1). Overall, both the time and space complexities for this code can effectively be considered constant, 0(1), under the assumption of

a fixed-size hand in a card game.