Exercise 5 (for grade) ~ Wednesday, October 19, 2022 ~ CPSC 535 Fall 2022

Write one submission for your entire group, and write all group members' names on that submission. Turn in your submission before the end of class. The ❎ symbol marks where you should write answers.

Recall that our recommended problem-solving process is:

1. **Understand** the problem definition. What is the input? What is the output?
2. **Baseline** algorithm for comparison
3. **Goal** setting: improve on the baseline how?
4. **Design** a more sophisticated algorithm
5. **Inspiration** (if necessary) from patterns, bottleneck in the baseline algorithm, other algorithms
6. **Analyze** your solution; goal met? Trade-offs?

Follow this process for each of the following computational problems. For each problem, your submission should include:

a. State are the input variables and what are the output variables
b. Pseudocode for your baseline algorithm, that needs to include the data type and an explanation for any variable other than input and output variables
a. The Θ-notation time complexity of your baseline algorithm, with justification.

and if you manage to create an improved algorithm:

c. Answer the question: how is your improved algorithm different from your baseline; what did you change to make it faster?
d. Pseudocode for your improved algorithm, that needs to include the data type and an explanation for any variable other than input and output variables
a. The Θ-notation time complexity of your improved algorithm, with justification.

Today's problems are:

1. *Spring 2020 Exercise 2 (Modified CLRS Exercise 2.3-7 , also* *https://www.geeksforgeeks.org/given-an-array-a-and-a-number-x-check-for-pair-in-a-with-sum-as-x/)*

Design an algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose product is exactly x.
Your goal: baseline O(n^2) time, improved to either O(n log n) worst-case time or O(n) expected time.

2. *Spring 2021 Exercise 5 (Cracking the Code Interview, ex 17.13 Re-space modified)*

By a bad accident and due to a simple editor which does not have an "Undo" function, all the spaces in a document are gone. Of course you can add them manually, but it would be easier if you write an algorithm to re-insert the spaces between what it considers as correct words. It may not be possible to do it exactly as it was, but the number of unrecognized words is to be minimized. A sentence like "I removed all the spaces by mistakes, my fault." became "Iremovedallthespacesbymistakes,myfault.". Most of the words are in a dictionary but a few are not. Given a dictionary D, i.e. a list of *n* strings, and the document, i.e. a string S of m characters, design an algorithm using tries to unconcatenate S in a way that minimizes the number of words that are not in D.
Your goal:baseline O(n*n*m) time, improved to expected O(n *m) worst-case time.
EXAMPLE:
Input: D={a, back, day, go, garden, gardening, is, my, to, today, yard, you**, in**},
S="todayisanicedaytodogardeninginmybackyard"
Output: today is a nice day to do garden in g in my back yard (1 unrecognized word, g)

3. *(Brute force / naive algorithm, Levitin, page 107, ex. 5 modified)*

Given one binary text with ten 1's, compute the number of comparisons (successful or unsuccessful) made by the brute-force algorithm in searching for each of the following patterns in the text:
a) P = 11110
b) P = 10001
c) P = 10101
Show your work.

Names

Write the names of all group members below.

Timothy Lee

Hetal Patel

Rosa Cho

Exercise 1: Solve and provide answer

Understand

Input/Output

Read and discuss the problem statement, then answer: **Describe the input and output definition in your own words.**

Input: A set or array of "n" integers and an integer "x"

Output: The pair of integers that if product exists, else None is such product does not exist.

Examples

Write out a concrete input value for the problem, and the concrete correct output. Repeat this at least once (so there are at least two input/outputs).

Input: [1,2,5,6,8], x = 8

Output: (1,8)

Input: [1,5,11,100], x = 61

Output: None

Corner Cases

A corner case is a special case of input that is particularly unusual or challenging. Are there corner cases for this problem? If so, describe them.

There could be negative numbers in the set or all the numbers could be the same.

Use Case

Think of a specific use case for an algorithm that solves this problem. What kind of feature could it support in practical real-world software?

A problem like this could help match two separate data from the database together. For example, perhaps you want to match a teacher's data with a course code to the course details that has the same course code.

Similar Problems

Does this problem resemble any other problems you remember? If so, which problem? Does that problem have an efficient algorithm? If so, what is the run time of that algorithm, inΘ-notation?

A similar problem is the 2-sum problem where we check to see if the sum of two numbers are equal to the given number. The algorithm does range from O(n^2) to O(logn) to O(n), depending on implementation.

Baseline

Design Overview

Design a naïve algorithm that solves the problem. This should be a simple algorithm that is easy to describe and understand. First, describe the basic idea of your algorithm in 1-2 sentences.

We will first try the naive approach that will pick a value from the set and check the product with all succeeding values. We continue to do this with every value down the set and return a possible pair or None if we iterate through all pairs and no matching product is found.

Pseudocode

Now, write clear pseudocode for your baseline algorithm.

```
Def find_product(array, x):
    For i in range (len(array)):
        For j in range ((i + 1), len(array))::
            If array[i] * array[j] == x:
                Return (array[i], array[j])
    Return None
```

Efficiency

What is the Θ-notation time complexity of your baseline algorithm? Justify your answer.

This algorithm will run at O(n^2). We have nested loops where we check each value in the array with the remaining succeeding values in the array. We do this for each element in the array until a match is found or if no match is found at all.

Improvement

Now, steps 3-6 involving creating a better algorithm. Only attempt this part if you finished the earlier parts and still have time.

Goal

What aspect of your baseline algorithm do you want to improve? Time efficiency, or something else?

☒ We can try to improve the time complexity of the algorithm to possibly O(n) using a hash table..

Design Overview

Design a better algorithm for the same problem. First, describe the basic idea of your algorithm in 1-2 sentences.

☒ We want to iterate through the elements, divide the target "x" by each element, and store into the hash table. As we iterate, if a matching divided target value is found in the hashtable, we can return those values.

Pseudocode

Now, write clear pseudocode for your improved algorithm.

☒

```
Def find_product(array, x):
    Hash_table = {}
    For i in range(0 , len(array)):
        # here, we want to make sure we aren't dividing by 0 and simply add 0 to the table if it exists.
        If array[i] != 0:
            cur = x / array[i]
            If cur in hash_table:
                Return (cur, array[i])
            Hash_table[array[i]] = True
        Else:
            Hash_table[0] = True
    Return None
```

Efficiency

What is the Θ-notation time complexity of your improved algorithm? Justify your answer.

☒ The time complexity of the algorithm is O(n) since we now only have a single loop. However, we have increased the space complexity to O(n) by using a hashtable.

Analysis

Did you meet your goal? Why or why not?

❎ Yes, the goal was met. We were able to get the most optimal time complexity of O(n) for this algorithm.

Exercise 2: Solve and provide answer

❎ Understand

Input/Output

Read and discuss the problem statement, then answer: **Describe the input and output definition in your own words.**

❎

Input: dictionary D, String S, character m,

Output: String S + character sp = “ “ concatenated

Examples

Write out a concrete input value for the problem, and the concrete correct output. Repeat this at least once (so there are at least two input/outputs).

❎ Example

Input D = “thisisanadvancedalgorithmsclass”

Output D_sp = “this is an advanced algorithms class”

Corner Cases

A corner case is a special case of input that is particularly unusual or challenging. Are there corner cases for this problem? If so, describe them.

❎What would be challenging if we were expected to include tabs/indentations if given a paragraph. Another issue would be determining whether or not to include spaces in words that aren’t common in the English language. Either way, it seems like it’s a matter of understanding the English language and grammar rules similar to compilers.

Use Case

Think of a specific use case for an algorithm that solves this problem. What kind of feature could it support in practical real-world software?

❎ This type of algorithm that we will use can be applied to not just word processors to determine where to insert whitespaces but also compilers that need to determine how to separate different strings/words/dictionaries. For instance, python compilers are especially sensitive to spacing so, given a user-written code, the python compiler determines whether the spacing is correct and, if not, throws a syntactical error.

Similar Problems

Does this problem resemble any other problems you remember? If so, which problem? Does that problem have an efficient algorithm? If so, what is the run time of that algorithm, in -notation?

☒ This reminds me of a time when I had to write an algorithm for CPSC 131 that need to search through strings for a specific word and then replace that word. The runtime of the algorithm was expected to be O(n) given any length string.

Baseline

Design Overview

Design a naïve algorithm that solves the problem. This should be a simple algorithm that is easy to describe and understand. First, describe the basic idea of your algorithm in 1-2 sentences.

☒ Given a dictionary aka list of strings, we must determine where to insert a whitespace based on a set of rules based off of the presence of upper-case words, word length, where the string begins, and where it ends. Once we establish these rules, then we insert the whitespaces until the string ends and then print/return the newly spaced sentence.

Pseudocode

Now, write a clear pseudocode for your baseline algorithm.

```
☒ dictionary D = [“asdf”, “diier”, tiger”, “Djikstra”]  //given input, dictionary D

for int I in D:                        //create a nested for loop
        temp_list = [[]]                  //create temporary container
    for char m in I:
            if m.isupper()
              temp_list.append()     //if there is a character that is an upper-case, add it first to
the temporary container
              temp_list[-1].append(m)       //continue to add characters

         temp.append(‘ ‘.join(‘’.join(i) for I in temp_list))        //after each character is joined, then
add whitespace after each word
         return(print(str(temp_list)))            //return printed list
```

Efficiency

What is the -notation time complexity of your baseline algorithm? Justify your answer.

❎ It should be a O(n) complexity given input characters m within dictionary D.

Improvement

Now, steps 3-6 involving creating a better algorithm. Only attempt this part if you finished the earlier parts and still have time.

Goal

What aspect of your baseline algorithm do you want to improve? Time efficiency, or something else?

❎ I would like to improve space complexity as the given base algorithm can only read it one character at a time.

Design Overview

Design a better algorithm for the same problem. First, describe the basic idea of your algorithm in 1-2 sentences.

❎ Given dictionary d, we will use a tree based algorithm in order to determine where there are any upper-case characters, where a string begins, and where a string ends. Then we can determine where to insert whitespaces.

Pseudocode

Now, write a clear pseudocode for your improved algorithm.

```
❎ dictionary D = [“asdf”, “diier”, tiger”, “Djikstra”]  //given input, dictionary D

for int i in D: //create nodes
    node[i] = D[i]

node sp = ‘ ‘          //create special node that is just a whitespace

for in j, fill, node in D:
      if char m in D = D.upperCase() //if uppercase character found
      root = node(m)                    //set string which uppercase character belongs to as root node

while (root){        //this loop goes through each node and inserts node_sp = whitespace after
each node, it traverses the tree from left to right

root -> left->right = insert.node_sp
```

}

Efficiency

What is the -notation time complexity of your improved algorithm? Justify your answer.

❎ The time complexity of the newly improved algorithm should be O(n) because the traversal of each node should be once given characters m.

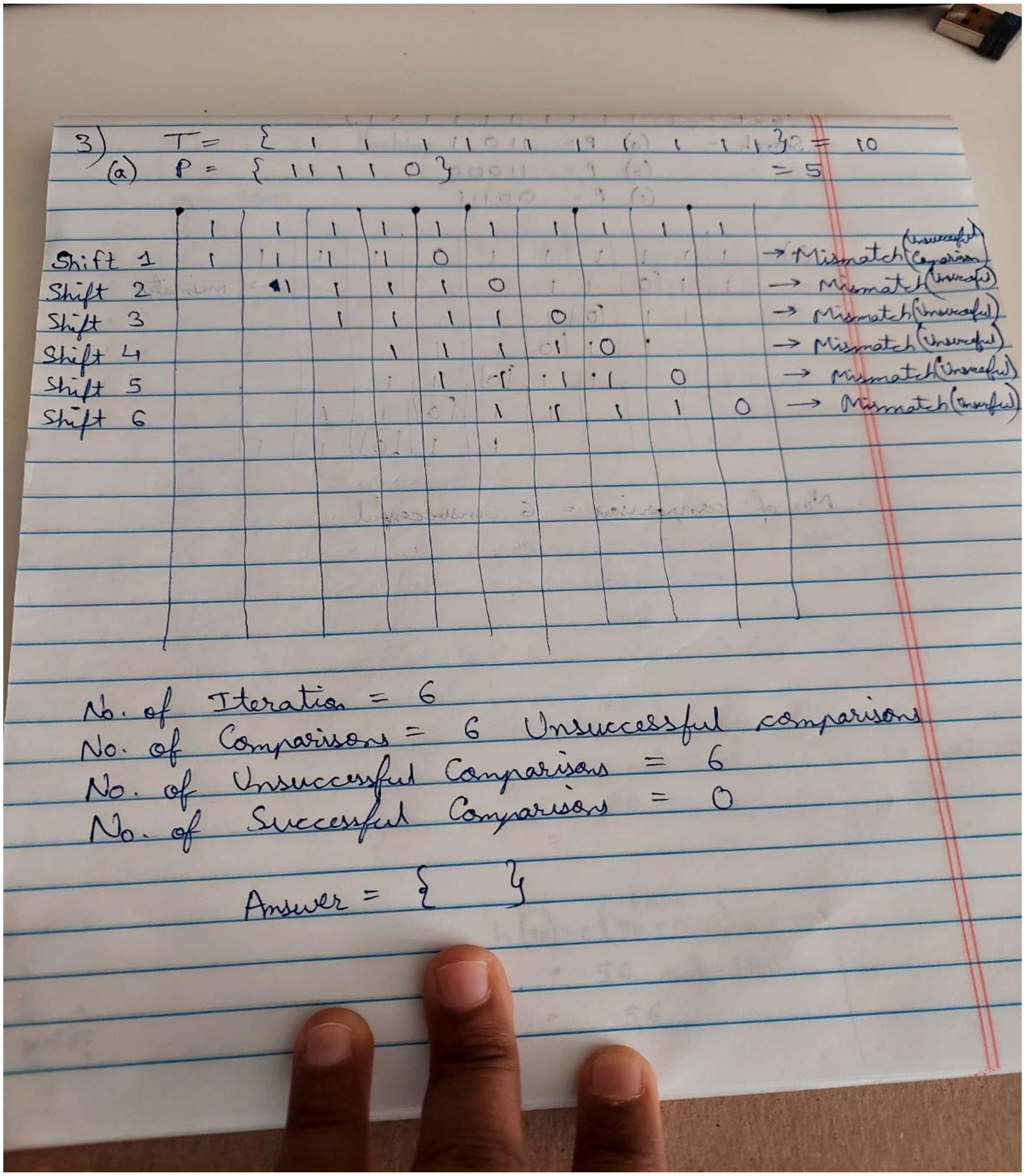
Analysis

Did you meet your goal? Why or why not?

❎ Yes, because the goal was to go through a string and insert whitespaces between each word so the newly improved algorithm was able to do so but much more efficiently.

Exercise 3: Solve and provide answer

X



3) T = { 1 1 1 1 1 1 1 1 1 1 } = 10

(a) P = { 1 1 1 1 0 } = 5

	1	1	1	1	1	1	1	1	1	1	
Shift 1	1	1	1	1	0						→ Mismatch (Unsuccessful Comparison)
Shift 2		1	1	1	1	0					→ Mismatch (Unsuccessful)
Shift 3			1	1	1	1	0				→ Mismatch (Unsuccessful)
Shift 4				1	1	1	1	0			→ Mismatch (Unsuccessful)
Shift 5					1	1	1	1	0		→ Mismatch (Unsuccessful)
Shift 6						1	1	1	1	0	→ Mismatch (Unsuccessful)

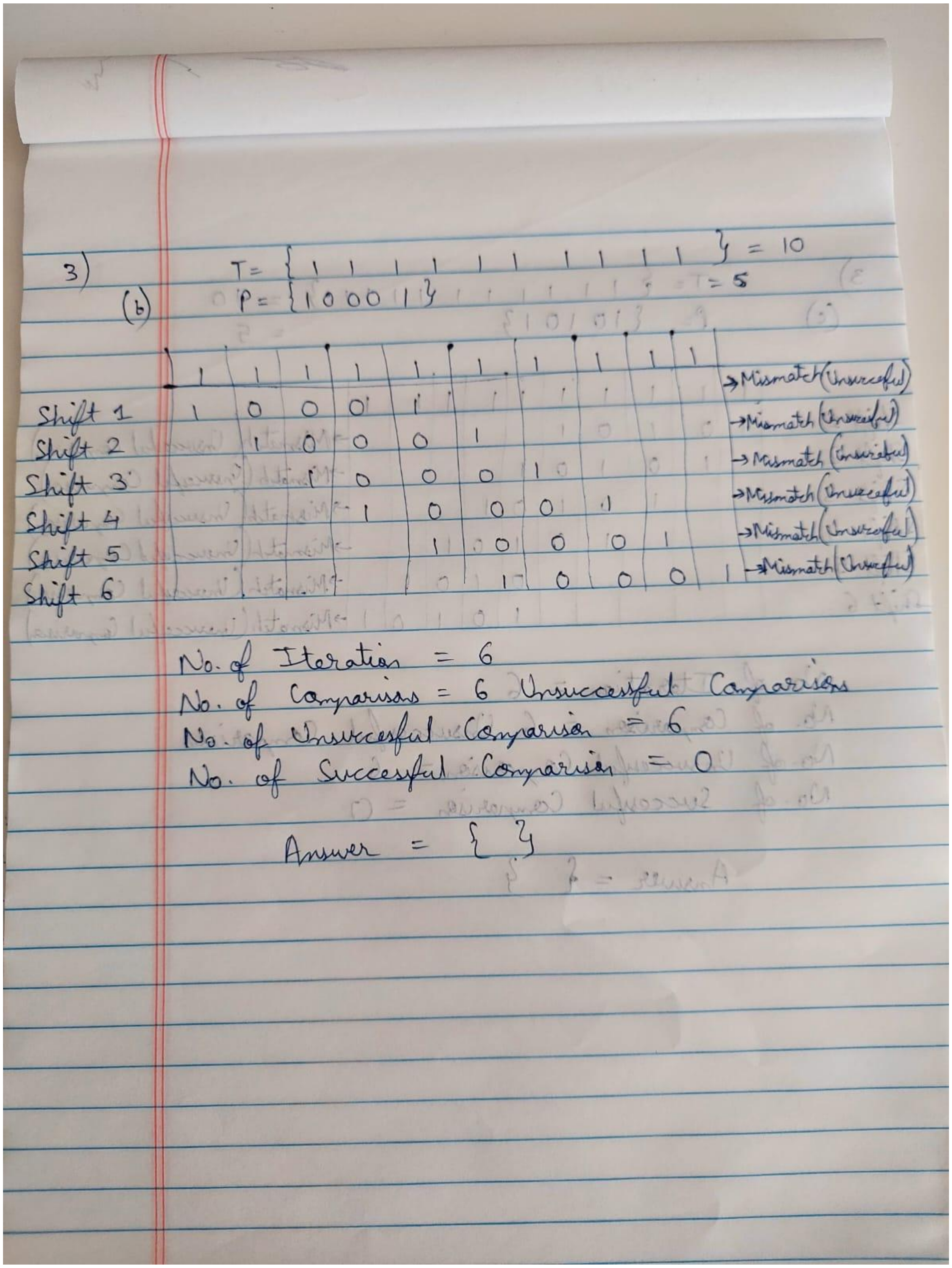
No. of Iteration = 6

No. of Comparisons = 6 Unsuccessful comparisons

No. of Unsuccessful Comparisons = 6

No. of Successful Comparisons = 0

Answer = { }



3)

(b) T= { 1 1 1 1 1 1 1 1 1 1 } = 10

P= {1 0 0 0 1} = 5

	1	1	1	1	1	1	1	1	1	1	
Shift 1	1	0	0	0	1						→ Mismatch (Unsuccesful)
Shift 2		1	0	0	0	1					→ Mismatch (Unsuccesful)
Shift 3			1	0	0	0	1				→ Mismatch (Unsuccesful)
Shift 4				1	0	0	0	1			→ Mismatch (Unsuccesful)
Shift 5					1	0	0	0	1		→ Mismatch (Unsuccesful)
Shift 6						1	0	0	0	1	→ Mismatch (Unsuccesful)

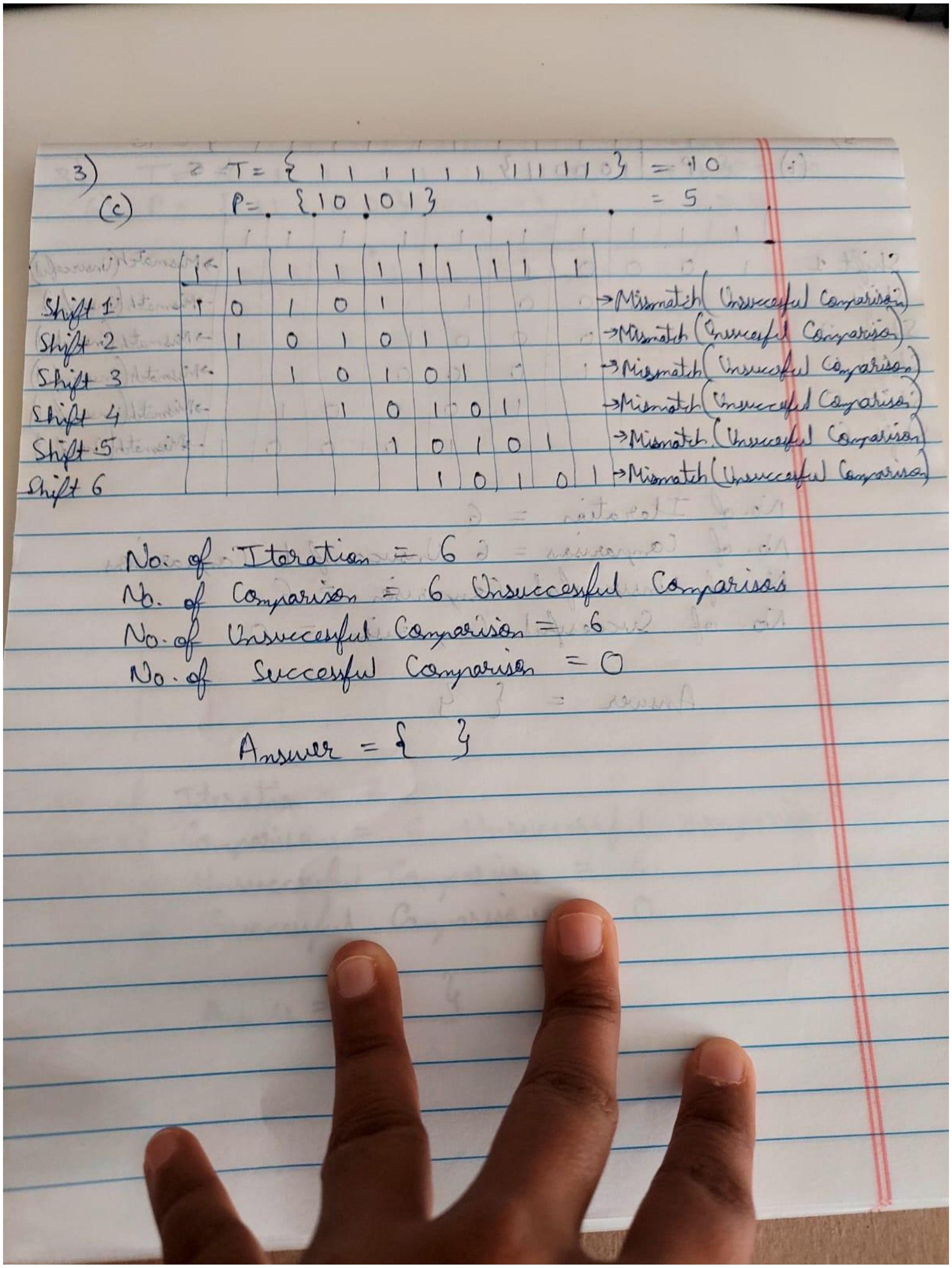
No. of Iteration = 6

No. of Comparisons = 6 Unsuccessful Comparisons

No. of Unsuccesful Comparison = 6

No. of Succesful Comparison = 0

Answer = { }



3) (c) T = {1 1 1 1 1 1 1 1 1 1} = 10

P = {1 0 1 0 1} = 5

	1	1	1	1	1	1	1	1	1	1	
Shift 1	1	0	1	0	1						→ Mismatch (Unsuccessful Comparison)
Shift 2		1	0	1	0	1					→ Mismatch (Unsuccessful Comparison)
Shift 3			1	0	1	0	1				→ Mismatch (Unsuccessful Comparison)
Shift 4				1	0	1	0	1			→ Mismatch (Unsuccessful Comparison)
Shift 5					1	0	1	0	1		→ Mismatch (Unsuccessful Comparison)
Shift 6						1	0	1	0	1	→ Mismatch (Unsuccessful Comparison)

No. of Iteration = 6

No. of Comparison = 6 Unsuccessful Comparisons

No. of Unsuccessful Comparison = 6

No. of Successful Comparison = 0

Answer = { }