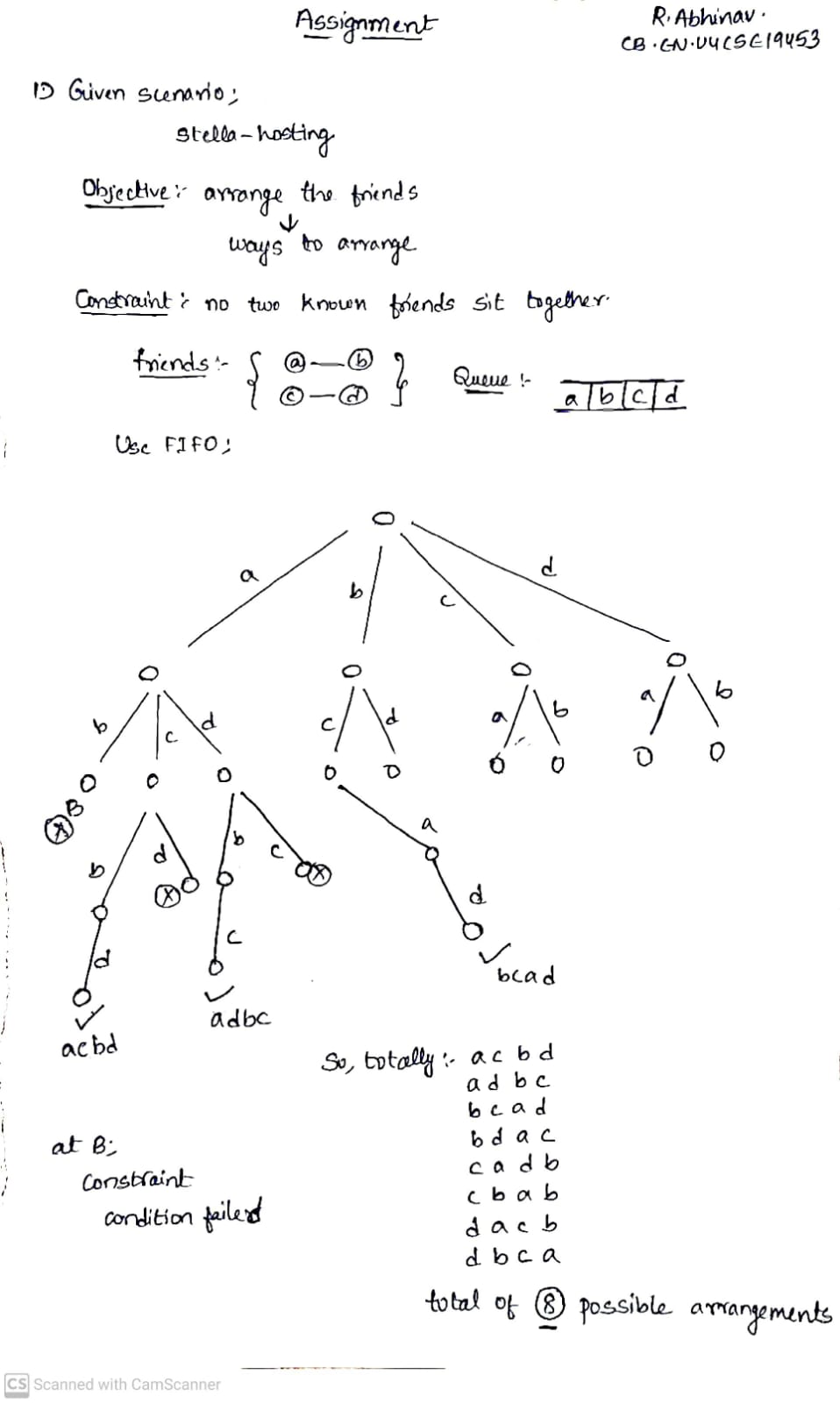
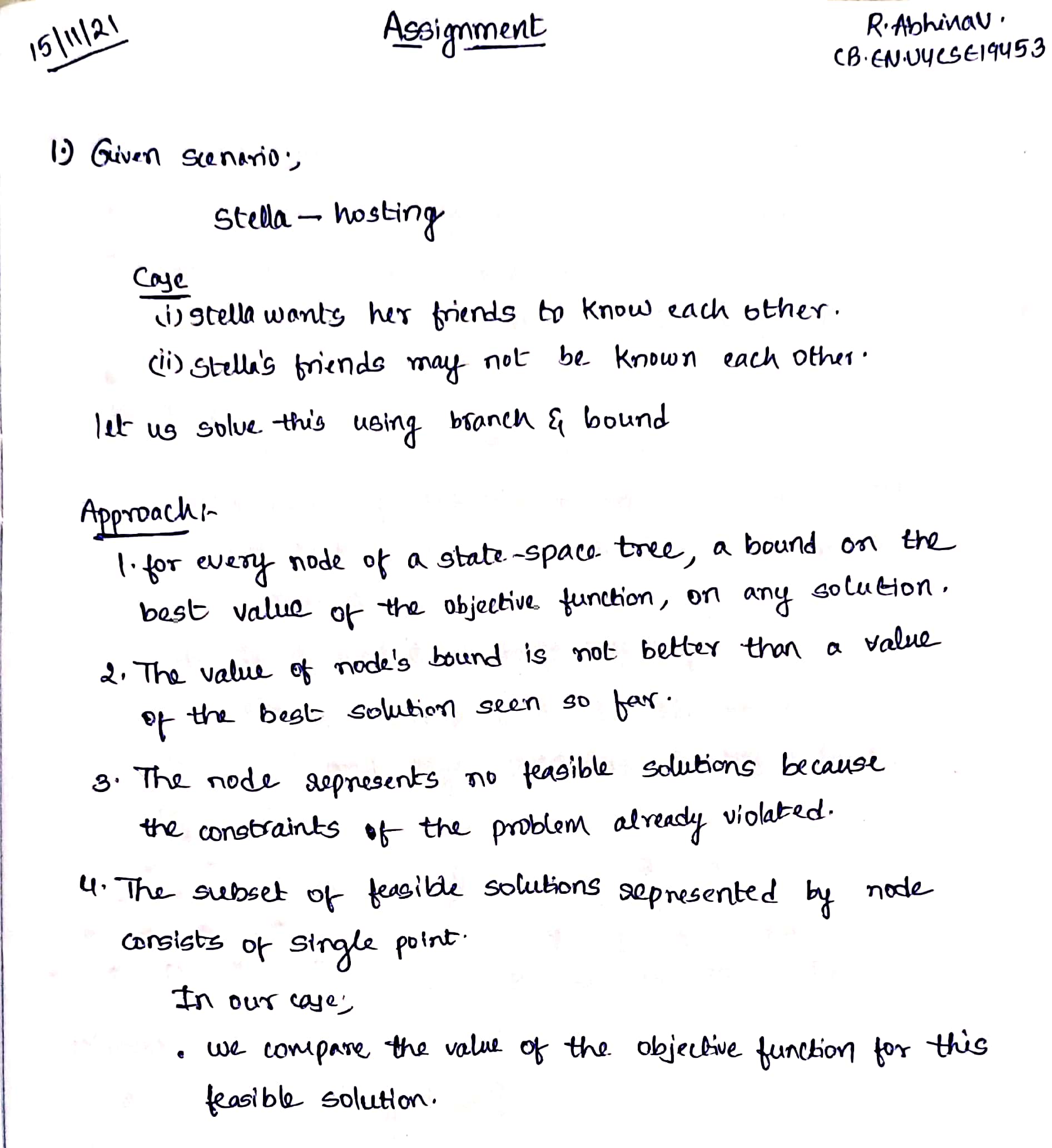
**19CSE302 – Design and Analysis of algorithms**

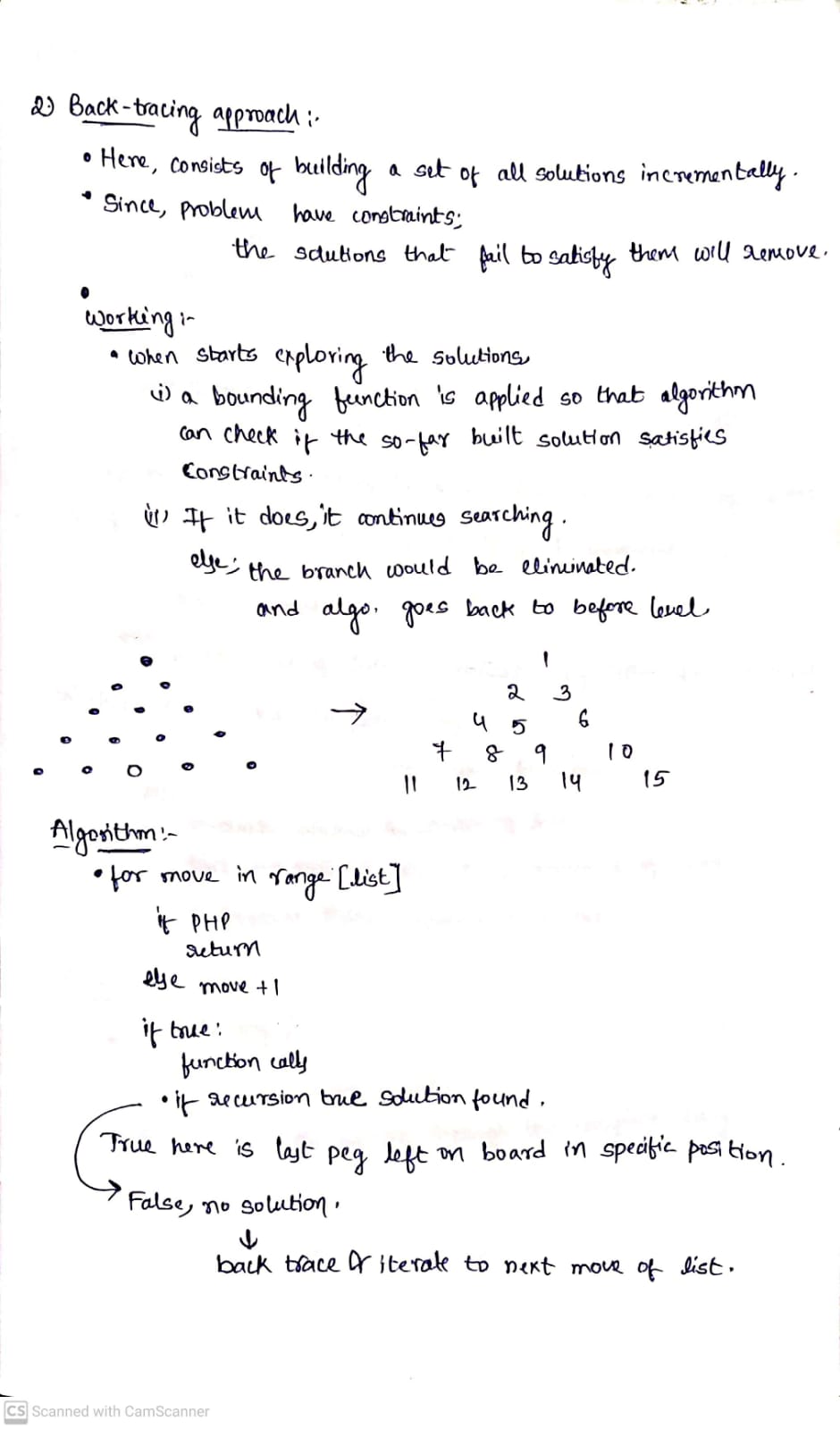
**Assignment – 10.11.2021**

1. Stella is hosting a Party. Stella invited few guests for the dinner and the party is arranged in the garden with few dining tables. Suppose that the host wants her friends to know each other. Use Branch and Bound method to solve the problem, also implement the solution.





1. **Puzzle Pegs :**

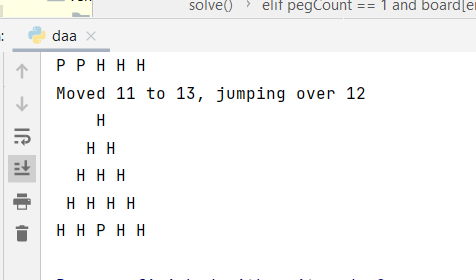




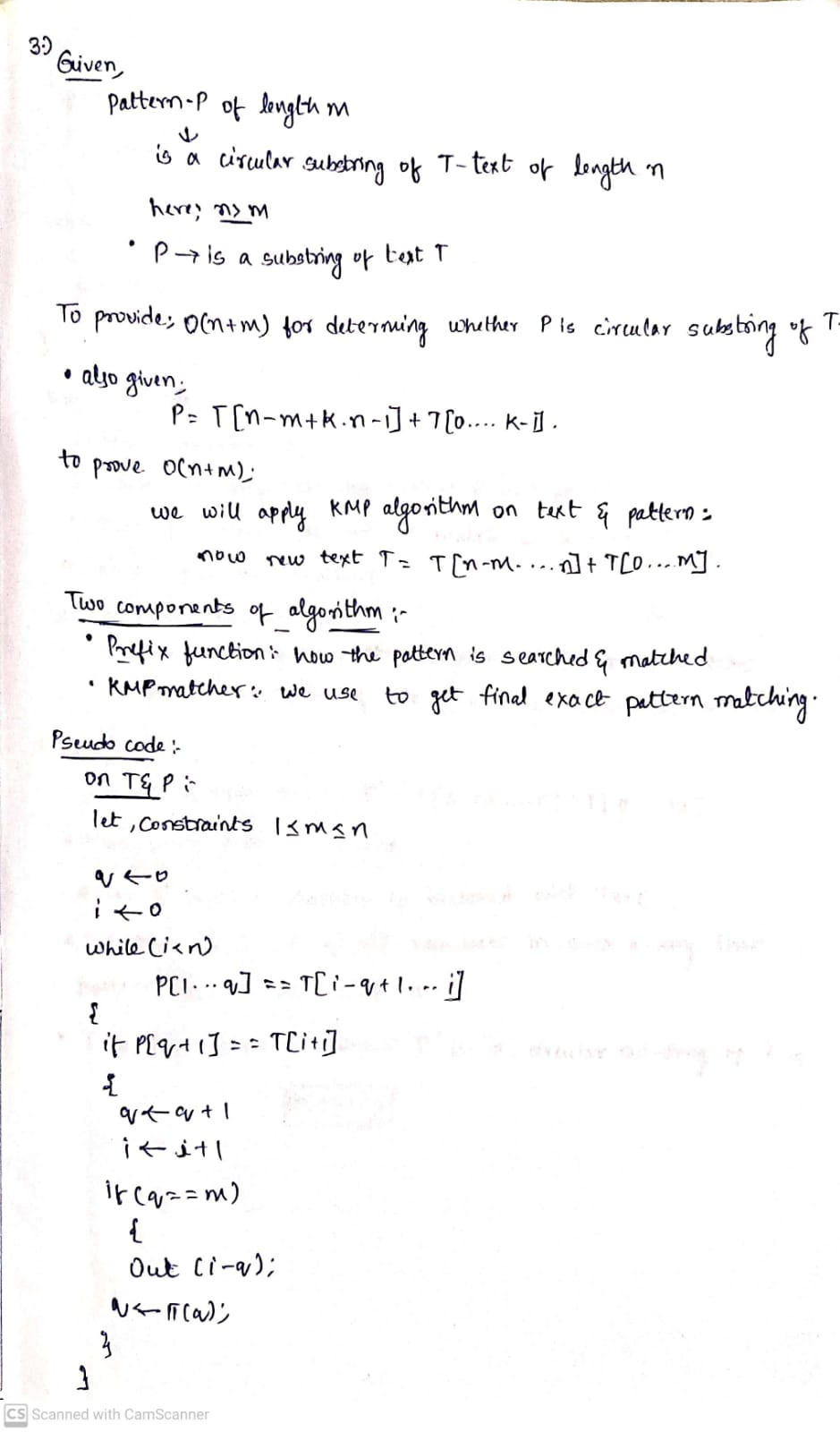
**Code:**

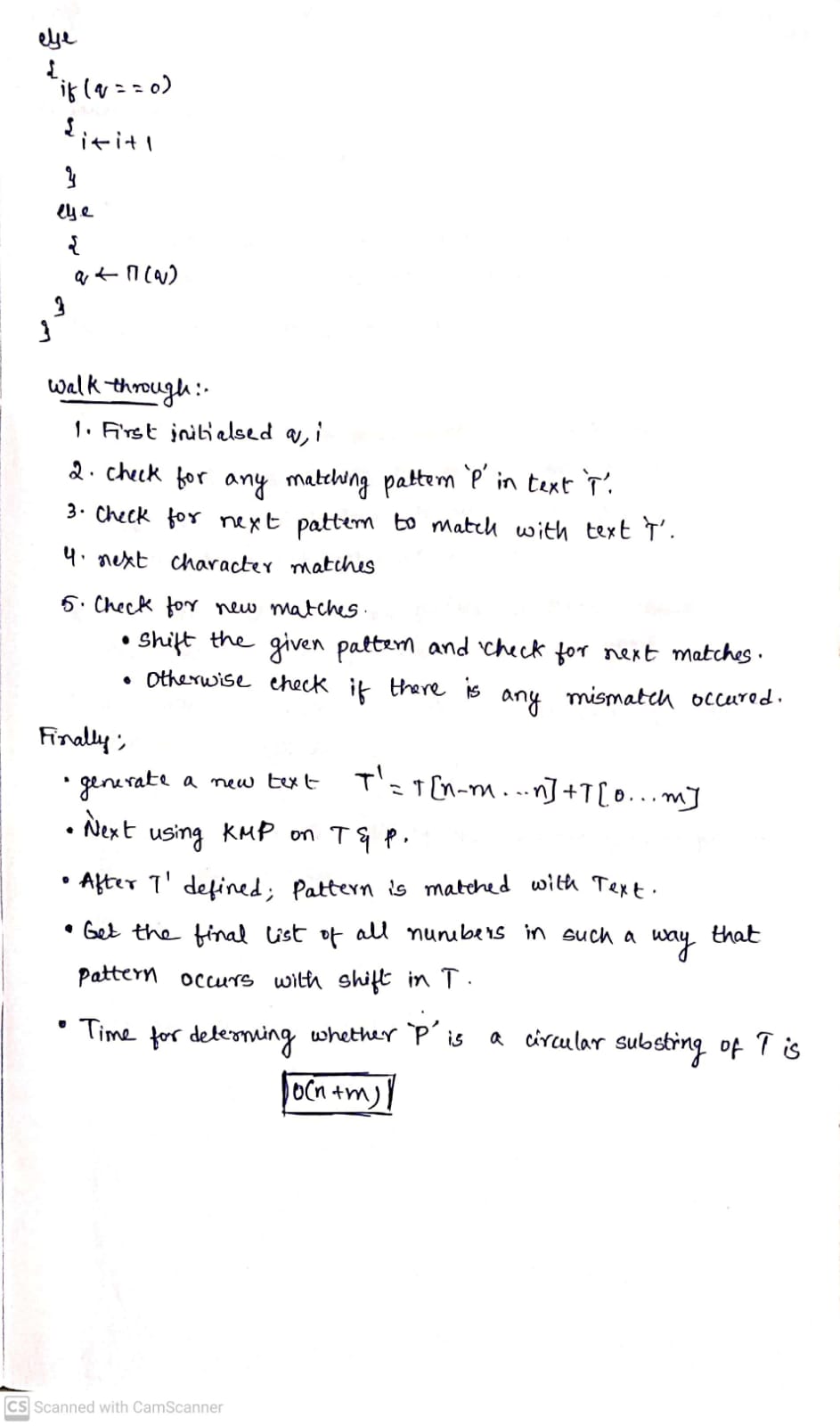
def func(nw: list, end: int):  
 for step in MOVES:  
  
 if nw[step[0]] == PEG and nw[step[1]] == PEG and nw[step[2]] == HOLE:  
  
 nw[step[0]] = HOLE  
 nw[step[1]] = HOLE  
 nw[step[2]] = PEG  
  
 clone = copy.deepcopy(nw)  
 nws.append(clone)  
  
 if func(nw, end):  
 jumps.append(  
 'stepd ' + str(step[0]) + ' to ' + str(step[2]) + ', jumping over ' + str(step[1]))  
 return True  
  
 if clone in nws:  
 nws.restep(clone)  
  
 nw[step[0]] = PEG  
 nw[step[1]] = PEG  
 nw[step[2]] = HOLE  
  
 pegCount = count(nw, PEG)  
 if pegCount == 1 and end == -1:  
 return True  
  
 elif pegCount == 1 and nw[end] == PEG:  
 return True  
  
 else:  
 return False  
  
  
if \_name\_ == "\_main\_":  
 main()

**Output:**



1. **S**ay that a pattern P of length m is a circular substring of a text T of length n if there is an index 0 ≤ i < m, such that P = T[n − m + i..n − 1] + T[0..i − 1], that is, if P is a substring of T or P is equal to the concatenation of a suffix of T and a prefix of T. Give an O(n +m)-time algorithm for determining whether P is a circular substring of T.

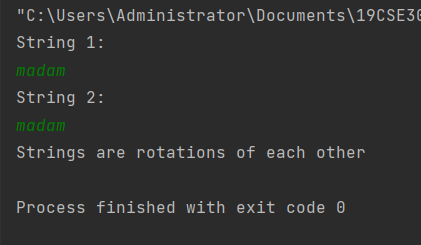




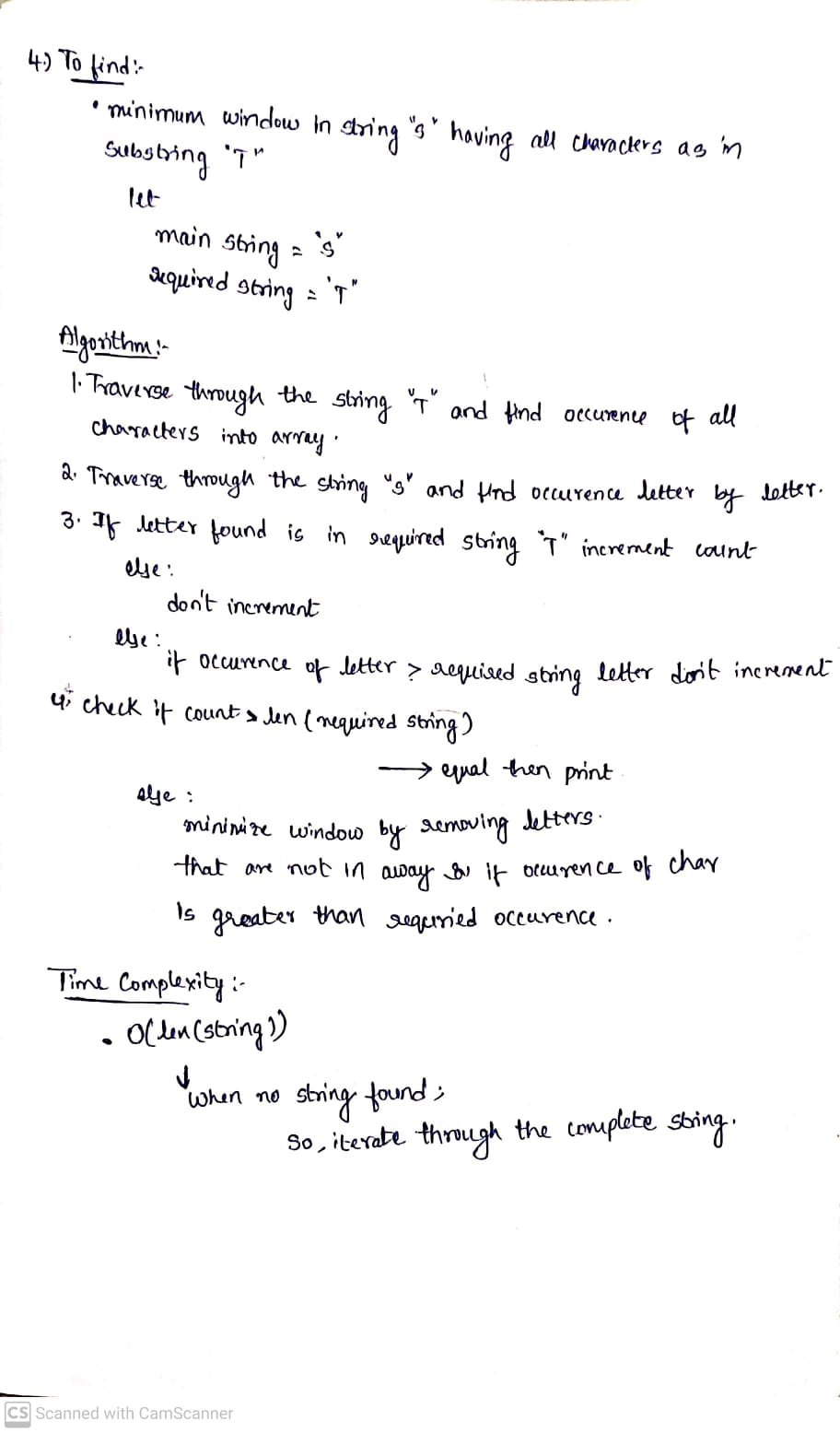
**Code:**

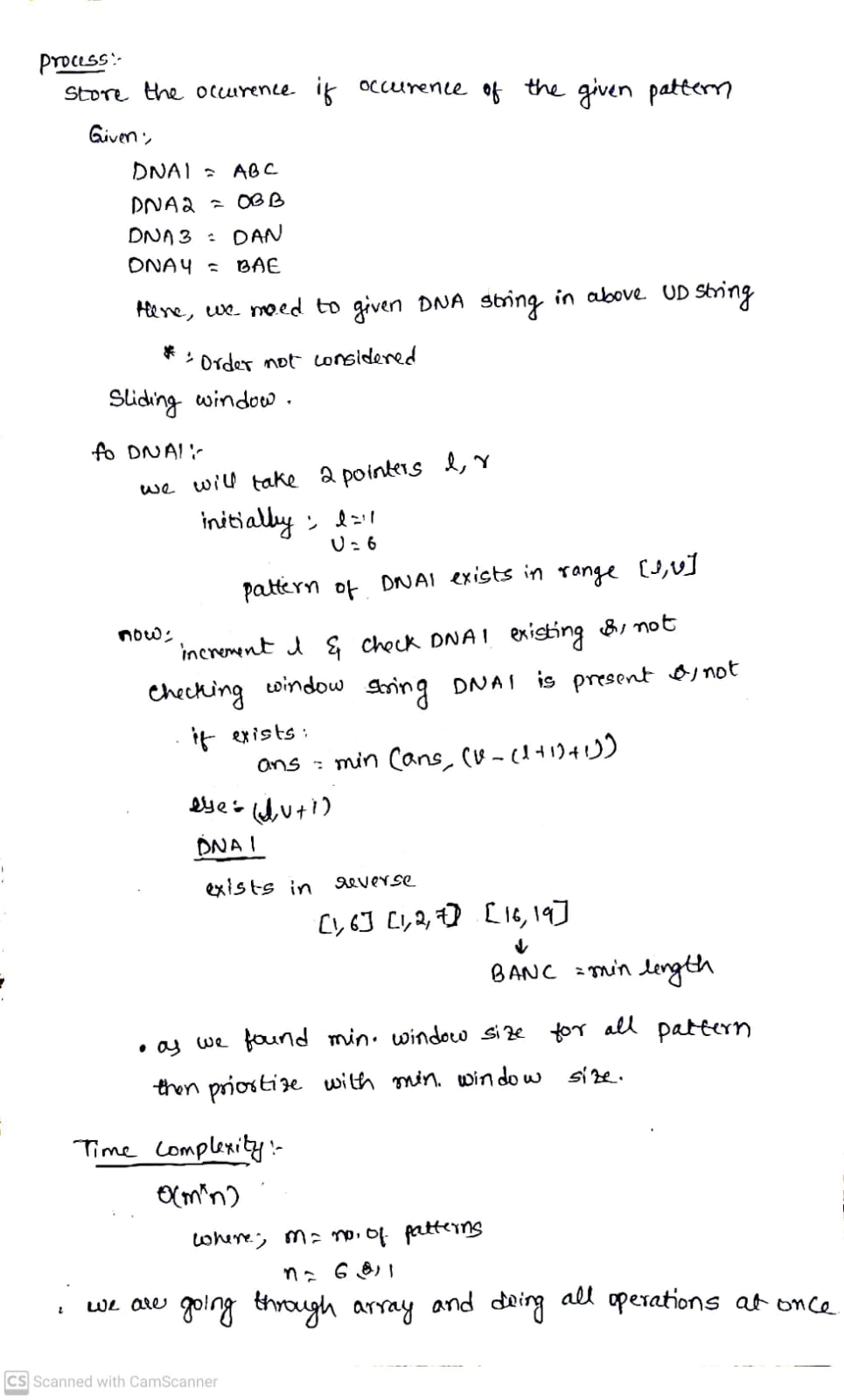
def func(p, q):  
 if (len(p) != len(q)):  
 return False  
 temp = p + p  
 e = len(q)  
 n = len(temp)  
 lps = [0] \* e  
 l = 0  
 k = 1  
  
 while (k < e):  
 if q[k] == q[l]:  
 l += 1  
 lps[k] = 1  
 k += 1  
 else:  
 if l != 0:  
 l = lps[l - 1]  
 else:  
 lps[k] = 0  
 k += 1  
 i = 0  
 j = 0  
  
 while (i < n):  
 if q[j] == temp[i]:  
 i += 1  
 j += 1  
 if j == e:  
 return True  
 elif i < n and q[j] != temp[i]:  
 if j != 0:  
 j = lps[j - 1]  
 else:  
 i += 1  
 return False  
  
i1 = input("String 1:\n")  
i2 = input("String 2:\n")  
if func(i1, i2):  
 print("Strings are rotations of each other")  
else:  
 print("Strings are not rotations of each other")

**Output:**

****

1. A crime has been committed in the city and the forensics person is able to identify some unknown DNA sample ‘UD’ on the crime scene. On the other hand, police has identified ‘N=5’ suspects who have the highest probability to be the murderer. The probability of a suspect to be a murderer is obtained by matching his DNA with unknown DNA ‘UD’. Arrange the suspects in the order of their decreasing probability to be a murderer.

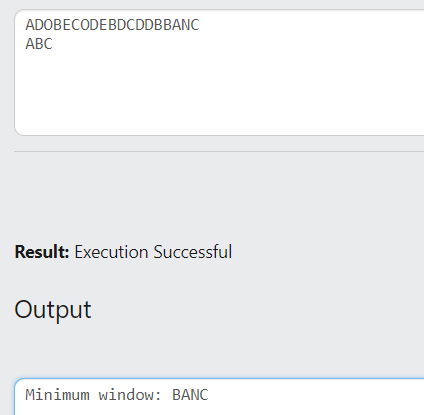


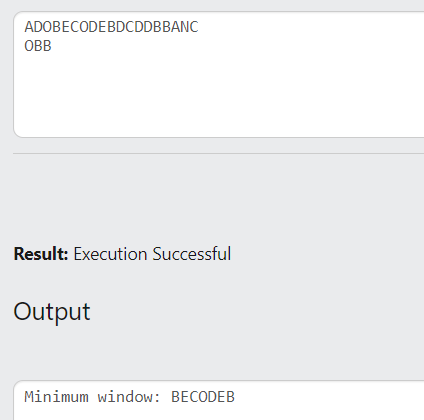


**Code:**

def SlidingWindow(main\_text, sub\_text):  
  
 main\_len = len(main\_text)  
 if main\_len < len(sub\_text):  
 return -1  
 main\_occ = [0] \* 256  
   
 start = 0  
 result = main\_len + 1  
 count = 0  
  
 for i in sub\_text:  
 main\_occ[ord(i)] += 1  
 if main\_occ[ord(i)] == 1:  
 count += 1  
 j = 0  
 i = 0  
  
 while(j < main\_len):  
 main\_occ[ord(main\_text[j])] -= 1  
 if main\_occ[ord(main\_text[j])] == 0:  
 count -= 1  
  
 while count == 0:  
 if result > j - i + 1:  
 result = j - i + 1  
 start = i  
 main\_occ[ord(main\_text[i])] += 1  
 if main\_occ[ord(main\_text[i])] > 0:  
 count += 1  
 i += 1  
 j += 1  
 if result > main\_len:  
 return "-1"  
 return main\_text[start:start+result]  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main\_text = input()  
 sub\_text = input()  
 result = SlidingWindow(main\_text, sub\_text)  
 print("Minimum window:", result)

**Output:**

****

****