CHENNAI MATHEMATICAL INSTITUTE
https://www.cmi.ac.in

Mid Semester Examination, Jan–Apr 2024

80/100

Name: Himanshu	Roll Number: MDS202327
Subject: ADT	Date: 27/02/2024
Course & Year: MScDS, IYear	Total No. of Pages: 4

Begin here

(1) a)

```
def secondBest(n,A):
    if n==1:
        return None
    else:
        x = A[0]
        y = None
    for i in range(1,n):
        if A[i]>x:
            y = x
            x = A[i]
    return y
```

Input: (3, [3,2,1])

The input is valid as $n = 3 \geq 1$, $n \leq 5$

Output: if loop fails (exits)

enters else: sets $x = A[0] = 3$, $y = \text{None}$

enters for loop ✓

$i = 1, 2$

$i = 1 \Rightarrow A[1] = 2 \not> x = 3$ (exits if)

$i = 2 \Rightarrow A[2] = 1 \not> x = 3$ (exits if)

returns $y = \text{None}$

But correct output should be $y = 2$

10/10

The reason why pseudo code returns wrong output is when a non-increasing array of integers is passed the if statement fails each time and doesn't update values of x, y assigned in else. and returns y = None

b)
```
def SecondBest(n, A):
    if n == 1:
        return None
    Largest = A[0]
    SecLargest = None
    for i in range(1, n):
        if A[i] < Largest:
            SecondLargest = A[i]
    for i in range(1, n):
        if A[i] > Largest:
            SecLargest = Largest
            Largest = A[i]
    return SecLargest
```

Input: $(3, [3,2,1])$. The input is valid as $n = 3 \geq 1$, $n \leq 5$

Output: if fails ⇒ Largest = 3, SecLargest = None

enters for loop i = 1, 2:

A[1] = 2 < 3 ⇒ SecLargest = 2

A[2] = 1 < 3 ⇒ SecLargest = 1

enters for loop i = 1, 2:

A[1] = 2 !> 3 : exits

A[2] = 1 !> 3 : exits

returns SecLargest = 1

10/10 ✓

But correct output should be secondgest = 2

The reason why pseudo code returns wrong output is

when non increasing array of integers is passed

the first loop assigns secondgest to smallest integer in array

and second loop : if condition fail each time and

doesn't update secondgest

and returns the smallest integer in the array

3. Say the masks people got for eg. 1, 4, 4, 1, 4, ...

take K=1 ⇒ holds true

for K=2 ⇒ subsets are of form {1,4}, {4,4}, {1,1}, {1}, {4}, ..

according to claim :

removing arbitrarily "1" element from subset

gives subset with same masks

takes S={1,4} for e.g.

remove 1 ⇒ {4} = S'
x=

m' holds the claim

remove 4 ⇒ {1} = S'' → m''
y=

holds the claim

10/10

$(S'' \setminus \{x\}) = (S'' \setminus \{y\})$ ✓

but $m' = 4 \neq m'' = 1$ ✓

Hence the proof is not correct ✓

(2) a) def maxPairProduct(n, A)

```
1.  max = -1                                          1
2.  maxindex = None                                   1
3.  for i in range(n):                                n
        if A[i] > max:                                n
4.          max = A[i]                                n
5.          maxindex = i                              n
6.  altmax = -1                                       1
7.  for i in range(n):                                n
8.      if A[i] == max and i != maxindex:             (1+1)n
9.          altmax = A[i]                             n
10. if altmax != -1:                                  1
11.     return max * altmax ✓                         1
12. else:
13.     secondmax = secondBest(A, max)                4n+3
14.     return secondmax * max ✓                      1
```

15/15

```
def secondBest(A, max):
1.  n = len(A)                                        1
2.  sec = -1                                          1
3.  for i in range(n):                                n
4.      if A[i] < max and A[i] > sec:                 2n
5.          sec = A[i]                                n
6.  return sec ✓                                      1
```

maxPairProduct() 2.2.

b) from line 3 to 6

the for loop calculates the maximum integer in the array and its index in the array (returns first occurance)

from line 8 to 10.

the for loop calculates alternate maximum

i.e other occurance of max in the array at different position than maxindex

and updates altmax with max (other occurance)

if max occurs once: leaves it to be -1

from line 11 to 12

if altmax is updated (meaning other occurance of max is there)

returns product of max, altmax (essentially max)

from line 13 to 15

otherwise (max occurs once only)

Calculate secondmax using secondBest() functions

and returns secondmax * max

15/15

secondBest() Note: max in loop is calculated above and passed in function

from line 3 to 6

checks if the element < max, > sec (set to -1 initially

and updates if hold

returns the second largest integer in array ✓

Since it is given len(A) ≥ 2 and before secondBest is called we have established that max occurs only once

hence our algorithm is correct

c) First analysis of secondBest()

$T(n) = 1+1+n+2n+n+1$

$= 4n+3$

$= O(n)$

Analysis of maxPairProduct()

$T(n) = 1+1+n+n+n+n+1+n+2n+n+1+1+4n+3+1$

$= 12n+9$

$= O(n)$

10/10

Algorithm runs in O(n) time and since we are making any copy of the array, just assigning value to variables (constant space is assumed)

hence constant extra space is used. ✓

3) def isNTP(s):
    l = 0
    n = len(s)
    r = n-1
    if s[l] != s[r]:
        return False
    else:
        isNTP(s[l+1, r])

4

3) a) def isNTP(s):

```
1.    if len(s)<2:
2.        return False
3.    else:
4.        return isPalndrome(s, 0, len(s)-1)
```

def isPalndrome(s, l, r):

```
1.    if l>=r:
2.        return True
3.    else:
4.        if s[l] != s[r]:
5.            return False
6.        else:
7.            return isPalndrome(s, l+1, r-1)
```

10/10

4

3 b) def isNTPSequence(w):

```
1.    if len(w)<2:
2.        return False
3.    else:
4.        return NTPHelper(w)
```

✓

def isNTPSequence(w): (Assuming input w has length ≥ 2 as given in definition box)

```
1.    n=len(w)
2.    for i in range(2,n+1):
3.        l=0
4.        r=i-1
5.        if isPalndrome(w[0:i],l,r)==True:
6.            new=w[i:]
7.            if len(new)==0:
8.                return True
9.            elif len(new)==1:
10.               return False
11.           else:
12.               isNTPSequence(new)
13.   return False
```

return this value? Or throw it away?

0/20

If you are not returning the value returned by the function call inside the red box: your algorithm will fail for an input like w = aaaa.

If you are returning this value: your algorithm will fail for an input like: w = abacabaddd

Note: ~~isNTPHelper()~~ isNTPSequence(): w of length n

in Line 5: w[0:i] goes from 0 to i-1

i.e [w[0], w[1], ..., w[i-1]]

in Line 6: w[i:] goes from i to last index

i.e [w[i], w[i+1], ..., w[n-1]]

(3) c) Assuming isNTP(), works correctly

Line 1 to 2: isPalindrome()

~~if len(w) < 2 (not a non trivial palindrome by definition)~~

~~return False~~ Your algorithm is wrong. See the counter-example.

Line 2 to 4

for loop from 2 to n (2, 3, ..., n)

checks incrementally if w[0:i] is a palindrome

str of length ≥ 2

Since i starts from 2

0/10

if not palindrome

increments i and check for w[0:i+1] ... so on

if loop exits: meaning str as a whole is not a palindrome

(not a non trivial palindrome)

return False

else for some j w[0:j] is a palindrome

Line 6 cuts the palindrome prefix part out

and line 7-10
checks if in_{cutout} = 0 i.e. completely made up of NTP (Hence returns true)

else if in_{cutout} ≠ 1 : cannot be a NTP by definition (Hence returns false)

else:
recursively solves for remaining cutout string

since we are incrementing i by 1 and not leaving any case the algorithm is correct

~~and since it runs in one for loop the~~

d) almost $\log_2 n$ ~~times~~ times (if it is made up of all NTP of length 2) ✗ 0/10

e) $\log_2 n$ is the upperbound for the calls ✗ 0/10