CSE221 ALGORITHMS

Assignment 1

Prepared by:
Saad Bin Sohan
BRAC University

Email: sohan.academics@gmail.com
GitHub: https://github.com/saad-bin-sohan

Assignment-01 (50 marks)

Iterative Time Complexity + Searching + Recursive tree Construction

1. Proof That C(n,r) = C(n-1,r-1) + C(n-1,r) [Here C means the combination] [You have to give mathematical proof only] **[5 mark]**

2. Write a recursive python code to find out the value of ${}^{n}C_{r}$ **[5 mark]**

3. Using your recursive python code in 2, Construct a recursive tree for the value [n=5 , r=2] **[5 mark]**

4. Write a pseudocode/python code for ternary search. [Should be similar to binary search] **[5 mark]**

5. Find out the time complexity of ternary search. [The way i have shown time complexity of binary search] **[5 mark]**

6. Find out the time complexity of following Code Snippet. **[10 mark]**

```
int i,j,k,m,multi,a,b,c
for( i = n; i >= 1; i = i / 7 ){
      for( j = 1; j <= n; j = j + 3 ) {
            for( k=1; k<=40 ; k=k+1){
                  multi=a*b
             }
            for( m=n ; m>=1 ; m=m-5 ){
                  multi=multi*c
            }
       }
 }
```

7. Find out the time complexity of following Code Snippet. [**5 mark**]

```
1. for i in range (1,n)
2.     j= 1
3.     while j*j < i
4.         j= j+1
```

8. Asymptotic Time complexity [**2 mark**]

In the primary scholarship exam in Bangladesh, four lakh (n=4,00,000) students take part but only the top **50** students are given an award.
Write the asymptotic time complexity to give the awards. Assume that each award is given in a constant time.

9. Asymptotic Time complexity [**3 mark**]

Find the time-complexity of the following task in terms of number of students.

You are given a student attendance sheet. Each student has a unique integer ID. You have to count the number of students having an even number as ID. The list is sorted but the IDs are not necessarily consecutive. So you check each ID one by one.

10. Searching: [**4+1 mark**]

You are given an array containing N distinct integers in a wave-like sequence. Meaning, the numbers in the beginning are in ascending order, and after a specific position, they are in descending order. For example: [1, 3, 4, 5, 9, 6, 2, -1]
You have to find the maximum number of this sequence. Can you devise an efficient algorithm such that the time complexity will be less than $O(N)$?

a) **Present** your solution idea as a pseudocode/ python code/ flowchart/ step-by-step instructions/ logical explanation in one-two paragraphs.
b) **Write** the time complexity of your algorithm.

Answer to the question - 1

$C(n-1, r-1)$, $C(n-1, r)$ can be rewritten as $\binom{n-1}{r-1}$, $\binom{n-1}{r}$

so,

$$\binom{n-1}{r-1} + \binom{n-1}{r} = \frac{(n-1)!}{(r-1)!\,[n-1-(r-1)]!} + \frac{(n-1)!}{r!\,(n-1-r)!}$$

$$= \frac{(n-1)!}{(r-1)!\,(n-r)!} + \frac{(n-1)!}{r!\,(n-r-1)!}$$

$$= \frac{(n-1)!\,r}{(r-1)!\,(n-r)!\,r} + \frac{(n-1)!\,(n-r)}{r!\,(n-r-1)!\,(n-r)}$$

$$= \frac{(n-1)!\,r}{r!\,(n-r)!} + \frac{(n-1)!\,(n-r)}{r!\,(n-r)!}$$

$$= \frac{(n-1)!\ (r+n-r)}{r!\ (n-r)!}$$

$$= \frac{(n-1)!\, n}{r!\ (n-r)!}$$

$$= \frac{n!}{r!\ (n-r)}$$

$$= \binom{n}{r}$$

$$= {}^{n}C_{r} \quad \text{[Proved]}$$

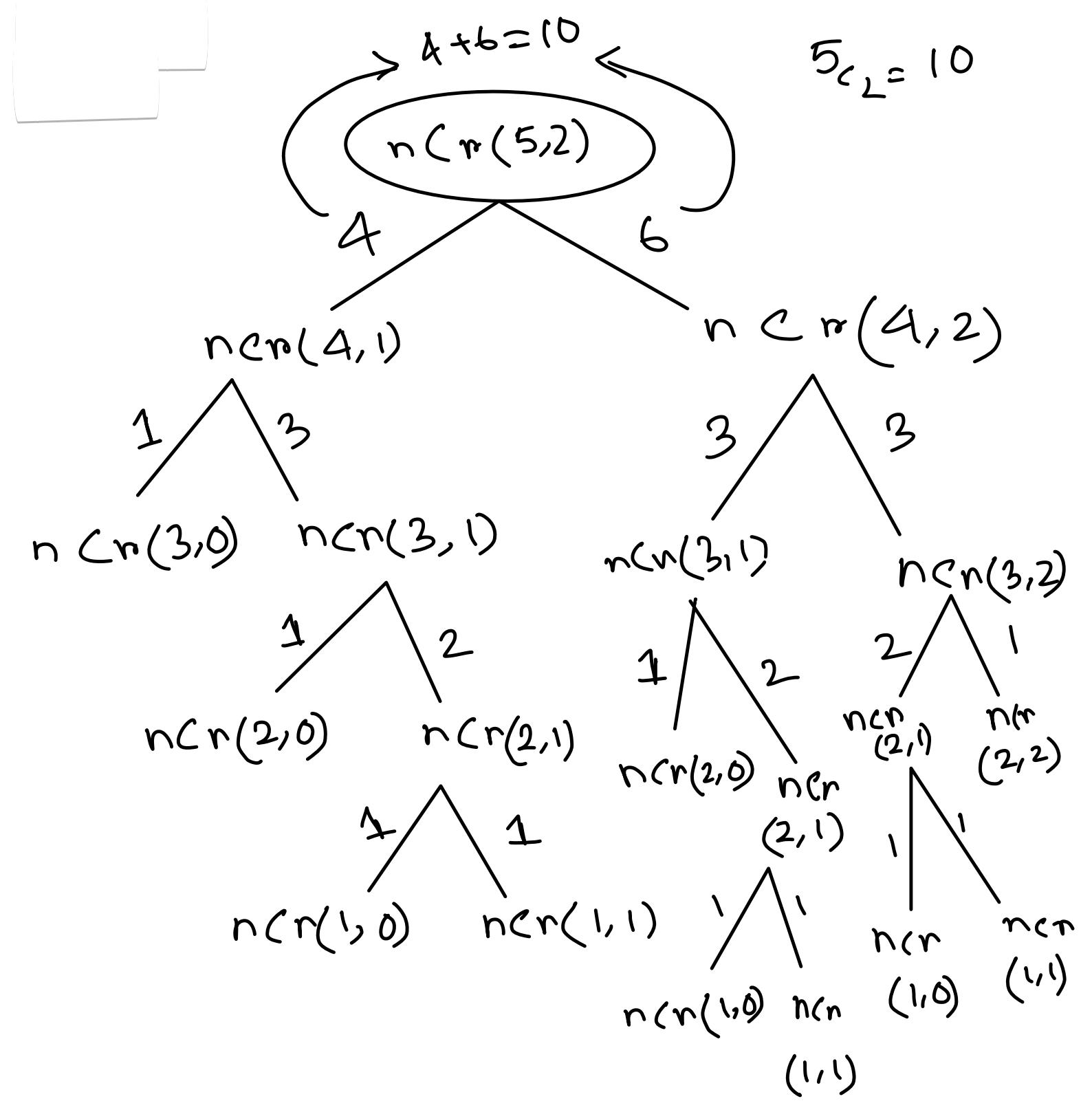
$$= C(n, r)$$

Answer to question- 2

```
def nCr(n, r):
    if r == 0 or n == r:
        return 1
    return nCr(n-1, r-1) + nCr(n-1, r)
```

Answer to question- 3

Let $n = 5$ and $r = 2$, we get the following tree,


4+6=10
5C2=10
nCr(5,2)
4
6
nCr(4,1)
nCr(4,2)
1
3
3
3
nCr(3,0)
nCr(3,1)
nCr(3,1)
nCr(3,2)
1
2
1
2
2
1
nCr(2,0)
nCr(2,1)
nCr(2,0)
nCr(2,1)
nCr(2,1)
nCr(2,2)
1
1
1
1
1
1
nCr(1,0)
nCr(1,1)
nCr(1,0)
nCr(1,1)
nCr(1,0)
nCr(1,1)

Answer to question- 4

```
def ternary_search(arr, left, right, key):
    if left<=right:
        mid1 = left + (right-left)//3
        mid2 = right - (right+left)//3
        if arr[mid1] == key:
            return mid1
        if arr[mid2] == key
            return mid2

        if key < arr[mid1]:
            return ternary_search(arr, left, mid1-1,
                                  key)
```

```
elif key > arr[mid2]:
    return ternary_search(arr, mid2+1,
                          right, key)

else:
    return ternary_search(arr,
                          mid1+1, mid2-1, key)

return "Number not found"
```

Answer to question- 5

Considering the ternary search code written in 4, we can write the following,

step	number of elements to search
0	$n = n/3^0$
1	$n/3 = n/3^1$
2	$n/9 = n/3^2$
3	$n/27 = n/3^3$
.	
.	
.	
k	$n/3^k$

So, $n / 3^k = 1$

$\Rightarrow \quad n = 3^k$

$$\log_3 n = \log_3 3^k$$

$$\log_3 n = k$$

$\Rightarrow$ $\boxed{k = \log_3 n}$

so time complexity for ternary search $= O(\log_3 n)$

Answer to question-6

```
int i,j,k,m, multi, a,b,c
for (i=n; i>=1; i= i/7) {
    for (j=1; j<=n; j=j+3){
        for (k=1; k<=40, k=k+1){
            multi= a*b
        }
        for (m=n; m>=1; m=m-5){
            multi=multi*c
        }
    }
}
```

$\rightarrow O(\log_7 n)$

$\rightarrow O(n)$

$\rightarrow O(40) = O(1)$

$\rightarrow O(n)$

ultimately the time complexity is,

$$\Rightarrow O(\log_7 n) \times O(n) \times \{O(1) + O(n)\}$$

$$\Rightarrow O(\log_7 n) \times O(n) \times O(n)$$

$$\Rightarrow O(n \times n \times \log_7 n)$$

$$\Rightarrow O(n^2 \log_7 n)$$ Ans

Answer to question-7

for the given code,

step	n	j_{max}
0	1	0
1	2	1
2	3	1
3	4	1
4	5	2
5	6	2
6	7	2
7	8	2
8	9	2
9	10	3
⋮	⋮	⋮
15	16	3
⋮	⋮	⋮

Step	n	j_{max}
⋮	⋮	⋮
k	k+1	$\sqrt{k}$

so, for the inner while loop, in the worst case scenario, it requires $\sqrt{i}$ times iteration, where i ranges from 1 to n-1. so it becomes $O(\sqrt{n})$. And the outer loops time complexity $= O(n)$

finally time complexity,

$= O(n) \times O(\sqrt{n})$

$= O(n\sqrt{n})$

Ans

Answer to question-8

Since there are 400000 students and we have to award top 50 only, we have to find 50 students with highest scores and therefore we need a sorting algorithm to sort them based on their scores. And asymptotic time complexity has three parts:

(i) upper bound

(ii) lower bound

(iii) tight bound

Lets consider we are using "Count Sort Algorithm"

where,

Case	time complexity
Best case	$O(n)$
Average case	$\Theta(n)$
Worst case	$\Omega(n)$

so we find the top 50 students by sorting them.

And then we can run a "for loop" to give only the first 50 students in the array where

runtime would be,

$$O(50) \Rightarrow O(50 \times 1) \Rightarrow O(1)$$

finally time complexity:

Time complexity (sorting) + Time complexity (awarding)

$O(n)$ + $O(1)$

$$\Rightarrow O(n) + O(1)$$

$$\Rightarrow O(n)$$

so the asymptotic time complexity is

(i) Upper bound: $O(n)$

in worst case scenario, the students are not in any order, so we have to iterate through the entire array to sort them.

(ii) lower bound: $\Omega(n)$

even in base case scenario, we need to iterate through entire students info, so its order of n.

(iii) tight bound: $\theta(n)$

since the time complexity is always n, the time complexity is always n.

And since n has been mentioned as $n = 400000$

we can say => $O(n) = O(400000)$

$= O(400000 \times 1)$

$= O(1)$

Answer to question-9

lets consider a code for this question,

```
count=0
array = [...ids...]

for i in range(len(array)):
    if array[i] %2==0
        count+=1
```

here since all ids have to be checked one after another it will run as many time as the total

student count. so if the length of array containing the students' ids is n, so the time complexity is = $O(n)$

(ans)

Answer to question- 10

10 (a)

We cannot make the task a constant time complexity operation. So the only option is to make a logarithmic functional approach.

We can modify the binary search to accomplish the task where time complexity will be $O(\log_2 n)$ instead of $O(n)$.

So here's the code:

```
def customised_binary_search(arrr):
    mid = len(arrr)//2
    arrr_1 = arrr[mid:]
    arrr_2 = arrr[:mid]

    if len(arrr)==1:
        return arrr[0]

    if arrr_1[-1]>arrr_2[0]:
        return customised_binary_search(arrr_1)

    else:
        return customised_binary_search(arrr_2)
```

```
arr = [1, 3, 4, 5, 9, 6, 2, -1]

function_call = customised_binary_search(arr)

print(function_call)
```

10 (b)

In the given code, since each iteration reduces the number of elements to be searched by half its time complexity is same as binary search.

time complexity = $O(\log_2 n)$