CSE221 ALGORITHMS

Topic: Recursive Tree Construction, Backtracking, Searching, Binary Search

Prepared by:

Saad Bin Sohan

BRAC University

Email: sohan.academics@gmail.com

GitHub: https://github.com/saad-bin-sohan

Topic: Recursive Tree Construction

two examples we will see

fibonacci (class)

${}^{n}C_{r}$ (assignment)

[redacted]

$$fib(n) = \begin{cases} 0 & ; \text{ if } n = 0 \\ 1 & ; \text{ if } n = 1 \\ fib(n-1) + fib(n-2) & \text{elsewise} \end{cases}$$

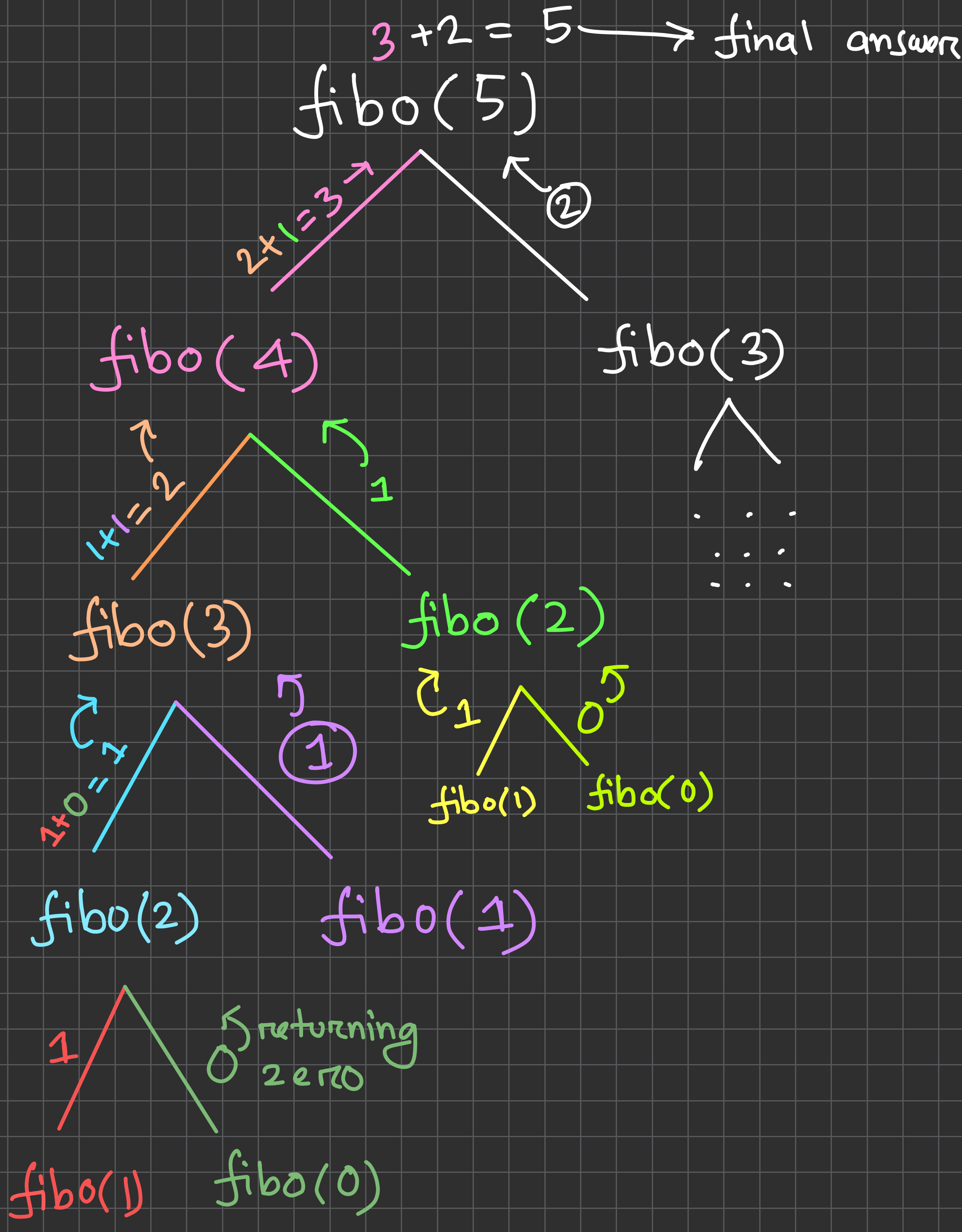
```
def fibo(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return fibo(n-1)+fibo(n-2)
```

base case (no function call, just return value)

recursive case (the function calling itself)

→ 221 exam e কোনো code আসবে না। question type: simulation,
algorithm detect

[redacted]


3 +2 = 5 → final answer
fibo(5)
2+1=3
2
fibo(4)
fibo(3)
1+1=2
1
fibo(3)
fibo(2)
1+0=1
1
1
0
fibo(1)
fibo(0)
fibo(2)
fibo(1)
1
0
returning zero
fibo(1)
fibo(0)

Topic:

Back-tracking

- Checks for all possible combinations
- Subset sum
- Generate all subsets

$\{1, 2, 3\}$

$\Rightarrow \{1,2\}, \{\}, \{1,2,3\}$

set এ $\{1,2\}$ is same as $\{2,1\}$

☆ set এ n সংখ্যক element থাকলে

2^n সংখ্যক subset পাওয়া possible.

Que Generate all subsets

for $\{1, 5, 2\}$

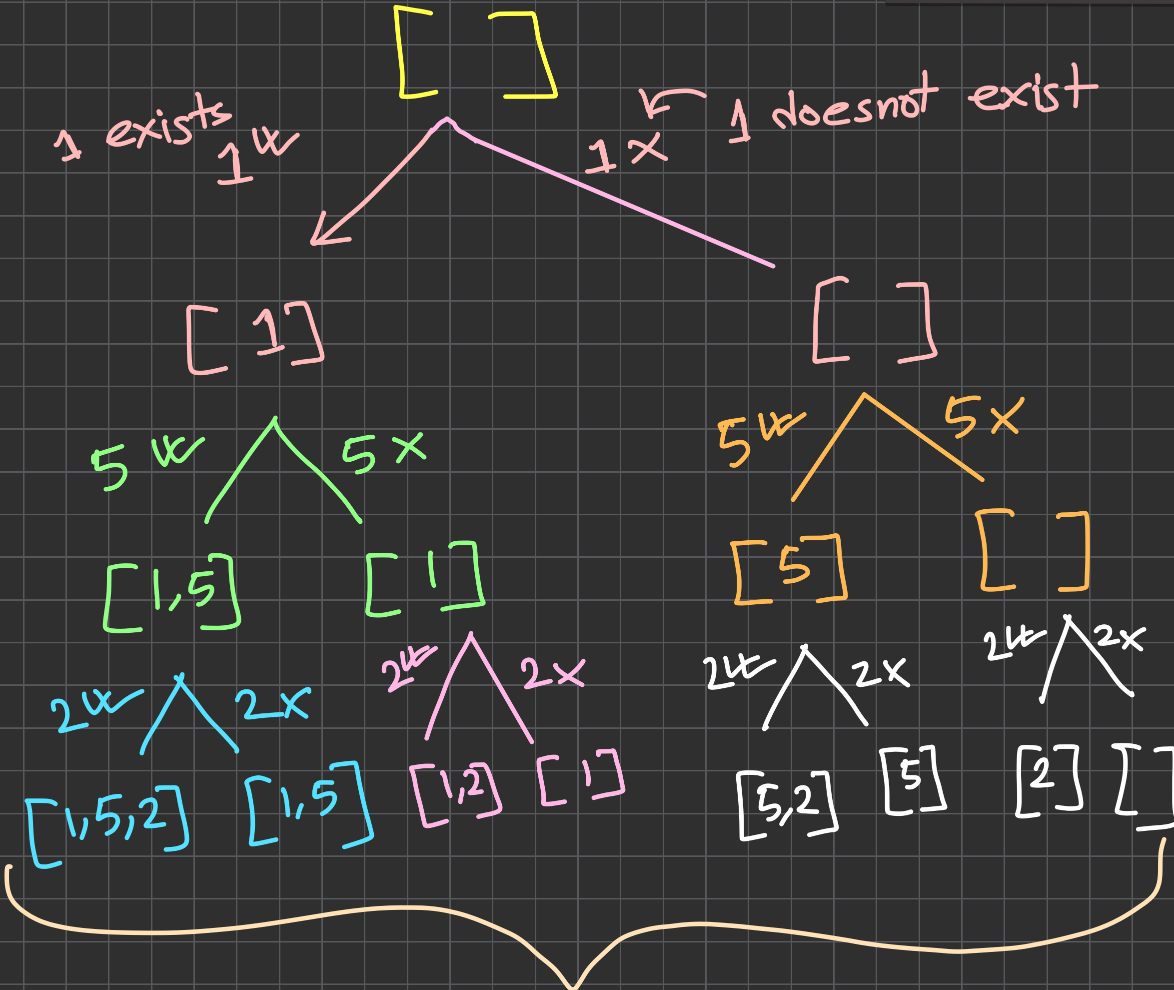
soln:

[ ] → start with an empty array.

then take each element from the set one by one and keep appending them in the previous array

in both cases that: the got number being appended and not being appended.

[redacted]



all the possible subsets

que for "subset sum" equation

Que $A = [1, 5, 2]$

target amount $= 3$. can we achieve the target amount by adding up any combination of the array?

<u>soln</u>: subset generate করতে করতে at a point $[1,5]$ এর জন্য sum $1+5 = 6 > 3$ হলো. so এর পরে $[1,5]$ এ আর কোনো element append করার

প্রয়োজন নেই। bcz it will only increase the sum and therefore no more forward going with [1,5]

[redacted]

Searching

Linear

Binary

Ternary

Linear search:

arr

0	1	2	3	4	5
5	8	2	10	7	5

key = 10

```
for i in range(len(arr)):
    if key == arr[i]:
        return i
return "False"
```

time complexity for linear search: $O(n)$

Linear search can be applied to both sorted and unsorted arrays.

Binary Search

Pre condition: array MUST be sorted

0	1	2	3	4	5	6	7	8
3	5	7	13	18	20	25	30	33

we will use three variables:

i) l : denoting left sub-array

ii) r : denoting right sub-array

iii) mid : if the target key is at the middle of the array

0	1	2	3	4	5	6	7	8
3	5	7	13	18	20	25	30	33

$l=0$ $r=8$

$$mid = \frac{l+r}{2} = \frac{0+8}{2} = 4$$

[redacted]

<u>scenario goes like this:</u>

we have a sorted array with some numbers in it.

the number that we want to find out whether this number is in the array or not, we will call it "key".

There can happen either one of these three situations (if the number exists in the array at all):

i) the number is at the exact middle point of the array

ii) the number exists in any position left to the mid index of the array

iii) the number exists in any position right to the mid index of the array

if the target number is smaller than the number at the mid-index of the array, then we consider only the left side of the array (by updating the value of r. And so the new r becomes the index right before mid index.

$r = mid - 1$

something like this,

0	1	2	3	4	5	6	7	8
3	5	7	13	18	20	25	30	33

$l=0$ $\qquad$ $r=8$

$$mid = \frac{l+r}{2} = \frac{0+8}{2} = 4$$

$r = mid - 1$

if $key < arr[mid]$

let
$key = 5$

else if the target number is bigger than the number at the mid-index of the array, then we consider only the right side of the array (by updating the value of l. And so the new l becomes the index right after mid index.

$l = mid+1$

[redacted]

something like this,

0	1	2	3	4	5	6	7	8
3	5	7	13	18	20	25	30	33

$l=0$ $\quad$ l $\quad$ $r=8$

$$mid = \frac{l+r}{2} = \frac{0+8}{2} = 4$$

if key > arr[mid] $\quad$ key = 30

so in binary search,

```
while l<=r:

    mid = (l+r)//2

    if key == arr[mid]:
        return mid

    elif key < arr[mid]:
        r = mid-1

    elif key > arr[mid]:
        l = mid+1
```

step	number of elements to search
0	$n = n/2^0$
1	$n/2 = n/2^1$
2	$(n/2)/2 = n/2^2$
3	$n/8 = n/2^3$
⋮	⋮
k	$(n/2^k)$

by the end of our search we will have only one element to search

so,

$$n/2^k = 1$$

$$n = 2^k$$

$$\log_2 n = \log_2 2^k$$

$$\Rightarrow \log_2 n = k$$

$$\Rightarrow \boxed{k = \log_2 n}$$

so time complexity for binary search $= O(\log_2 n)$

for example, if len(arr) == 1024

linear search এ ⟶ 1024 টা search

binary search এ ⟶ $\log_2 1024$

$= \log_2 2^{10}$

$= 10$ টা search

করতে হবে maximum