DAAPY 11

Saturday, April 1, 2023 9:09 PM

1. March 26
2. April 2 -- 11 class
3. April 9 12 – 13
4. April 16
5. April 23 FINAL EXAM 14

EXAM

Coding → 100 → Full test → 70%

Code

NON CODING

Very Simple

3-4

HAND

85

MAX

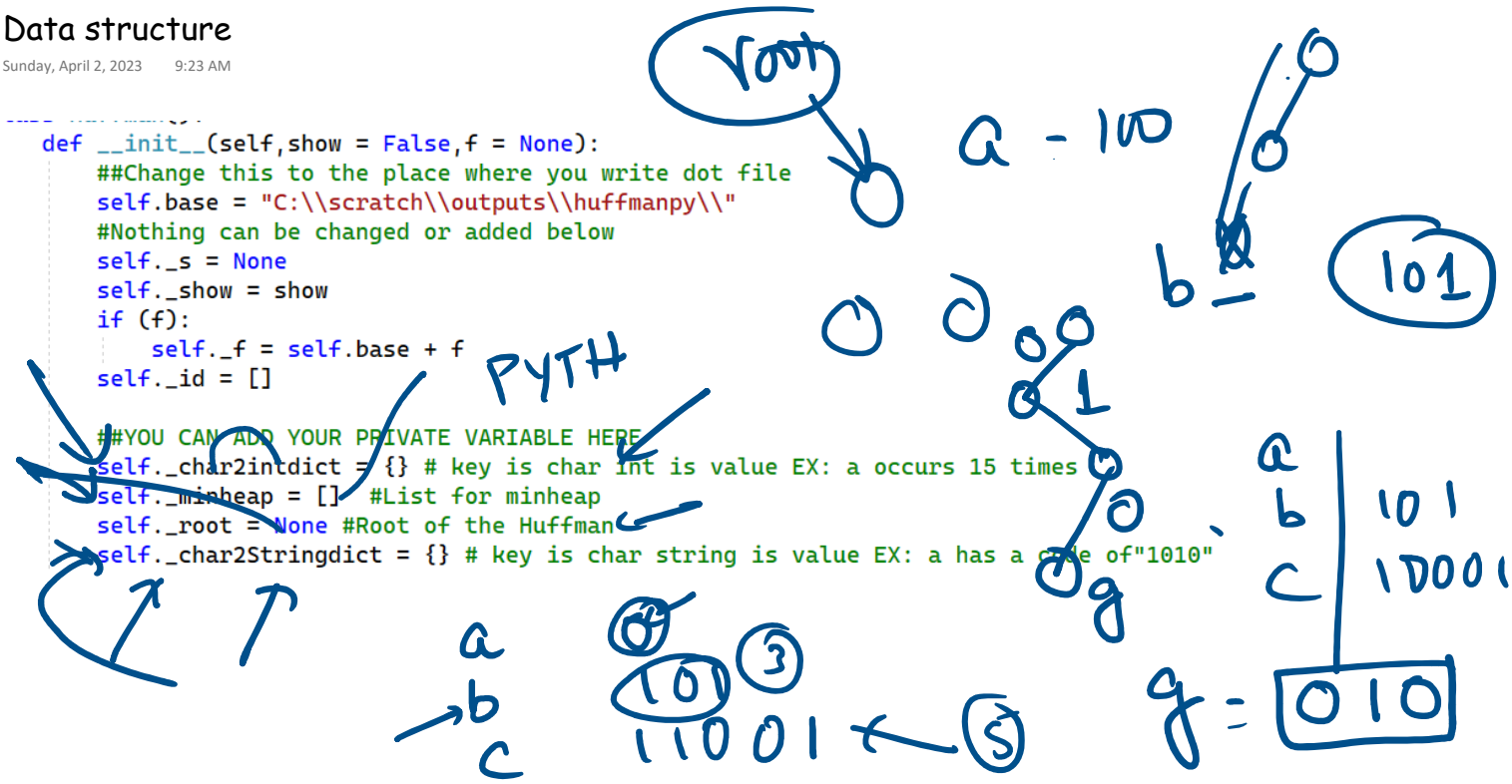
3 hr

1-1½

Data structure

Sunday, April 2, 2023 9:23 AM

```
class Huffman():
    def __init__(self,show = False,f = None):
        ##Change this to the place where you write dot file
        self.base = "C:\\scratch\\outputs\\huffmanpy\\"
        #Nothing can be changed or added below
        self._s = None
        self._show = show
        if (f):
            self._f = self.base + f
        self._id = []

        ##YOU CAN ADD YOUR PRIVATE VARIABLE HERE
        self._char2intdict = {} # key is char int is value EX: a occurs 15 times
        self._minheap = []  #List for minheap
        self._root = None #Root of the Huffman
        self._char2Stringdict = {} # key is char string is value EX: a has a code of"1010"
```



Node ds

Sunday, April 2, 2023 9:33 AM

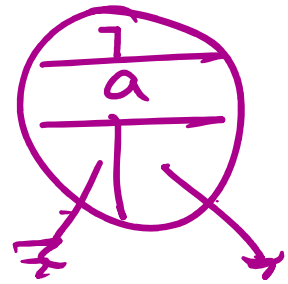
```python
class Node():
    def __init__(self, freq, data, l = None, r = None):
        # Nothing can be added in class Node
        # Must use Node class.Hacker rank fails without this Node
        self.freq= freq #Note public
        self.data=data #Note public
        self.left = l #Note public
        self.right = r #Note public
    ## YOU CAN ADD ANY NUMBER OF PRIVATE FUNCTIONS ONLY
    def is_leaf(self):
        if (self.left == None and self.right == None):
            return True
        return False

    def __lt__(self,rhs:'Node')->'bool':
        if (self.freq < rhs.freq):
            return True
        return False
```

Node(

n = node(7, a)

Min or Max

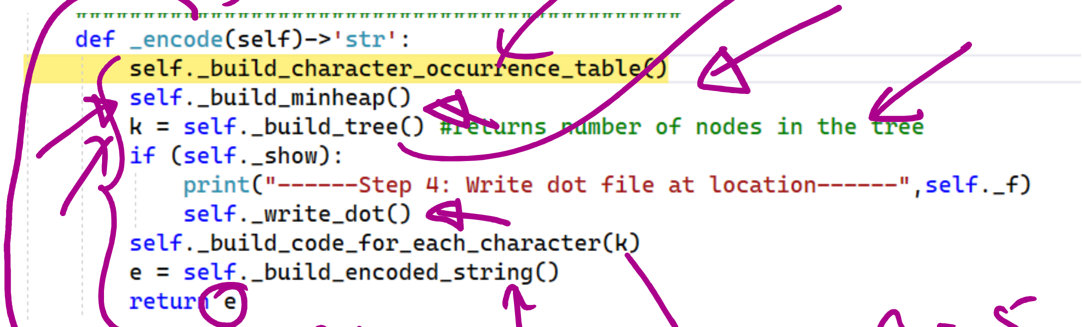
Minheap



1

Sunday, April 2, 2023 9:05 AM

s



```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
def _encode(self)->'str':
    self._build_character_occurrence_table()
    self._build_minheap()
    k = self._build_tree() #returns number of nodes in the tree
    if (self._show):
        print("------Step 4: Write dot file at location------",self._f)
        self._write_dot()
    self._build_code_for_each_character(k)
    e = self._build_encoded_string()
    return e
```

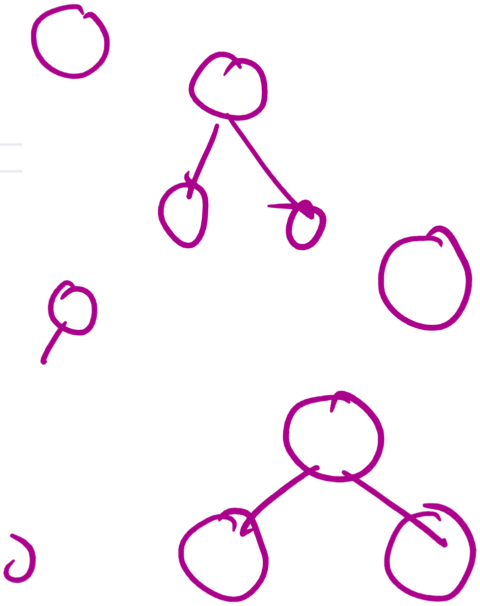
$\Theta(n)$

self._test1("aaabbgggggghhhhaaagggggaaaaa_+@#",show,"3.dot");

a = 5

b = 1

g = 10



2

Sunday, April 2, 2023 9:24 AM

```
##############################################
#  Time:  THETA(n)
#  Space: THETA(n)
##############################################
def _build_character_occurrence_table(self):
    for c in self._s:
        v = 0
        if (c in self._char2intdict):
            v = self._char2intdict[c] ## THETA(1)
        self._char2intdict[c] = v + 1
    if (self._show):
        print("------Step 1: Character occurrence ------")
        self._print_character_occurrence_table()
```

Code

abc

Θ(n)

Θ(1)

Θ(1)

Funda

a to z

Θ(n

```
        ##TOO MANY PRIVATE VARIABLE HERE
        self._char2intdict = {} # key is char int is value Ex: a occurs 15 times
```

26

Θ(1)

abcabcf

a = 2
b = 2
c = 2
f = 1

Char

a = [26]

Normal

abcabc9510

3

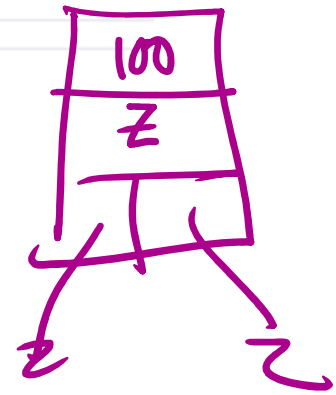
Sunday, April 2, 2023 9:36 AM

```
##########################################
#  Time:  THETA(n)
#  Space: THETA(n)
##########################################
def _build_minheap(self):
    if (self._show):
        print("------Step 2: Tree Leaves is build in this order------")
    i = 1
    for key, value in self._char2intdict.items():
        n = Node(value,key)
        heapq.heappush(self._minheap,n)
        if (self._show):
            print("Leaf Node",i,":",key,value)
            i = i + 1
```

a 11
b 5
z 100

$\Theta(n)$

log n



0

1
2
3
4

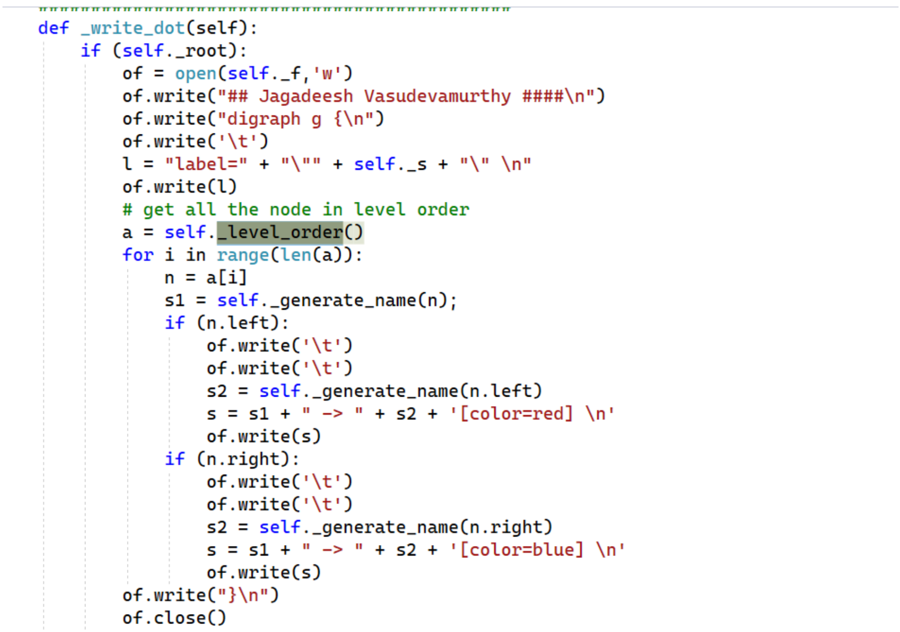
4

Sunday, April 2, 2023 9:45 AM

```
##############################################
#  Time:  THETA(n)
#  Space: THETA(n)
##############################################
def _level_order(self)->'Python List':
    a = []
    q = [] #queue of nodes
    q.append(self._root)
    while (len(q) != 0):
        n = q.pop(0) #queue. THis is NOT O(1)
        a.append(n)
        nl = n.left
        nr = n.right
        if (nl):
            q.append(nl)
        if (nr):
            q.append(nr)
    return a
```

5

Sunday, April 2, 2023 9:46 AM

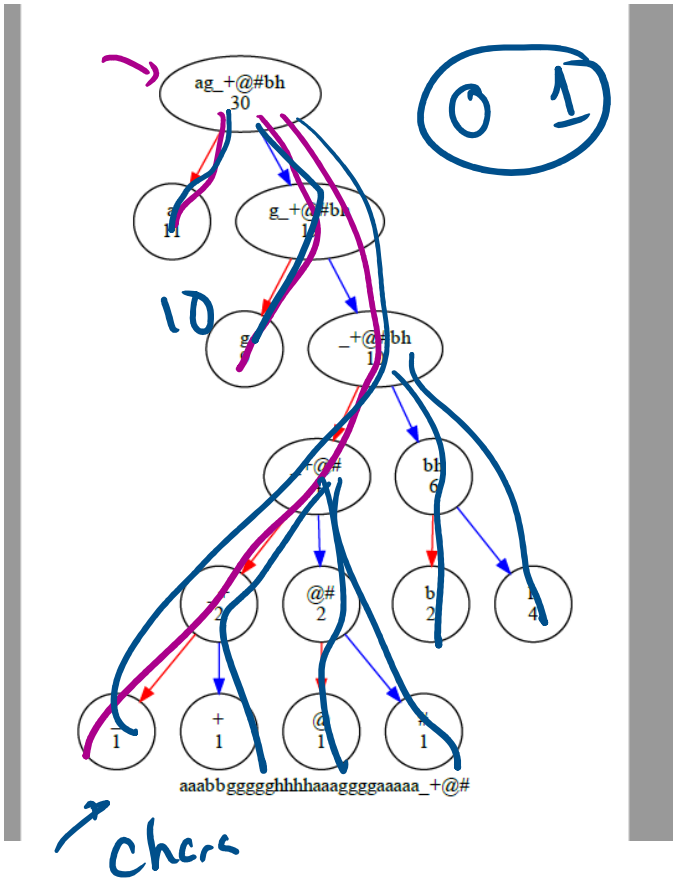


```python
def _write_dot(self):
    if (self._root):
        of = open(self._f,'w')
        of.write("## Jagadeesh Vasudevamurthy ####\n")
        of.write("digraph g {\n")
        of.write('\t')
        l = "label=" + "\"" + self._s + "\" \n"
        of.write(l)
        # get all the node in level order
        a = self._level_order()
        for i in range(len(a)):
            n = a[i]
            s1 = self._generate_name(n);
            if (n.left):
                of.write('\t')
                of.write('\t')
                s2 = self._generate_name(n.left)
                s = s1 + " -> " + s2 + '[color=red] \n'
                of.write(s)
            if (n.right):
                of.write('\t')
                of.write('\t')
                s2 = self._generate_name(n.right)
                s = s1 + " -> " + s2 + '[color=blue] \n'
                of.write(s)
        of.write("}\n")
        of.close()
```

```
## Jagadeesh Vasudevamurthy ####
digraph g {
        label="aaabbggggghhhhaaaggggaaaaa_+@#"
                "ag_+@#bh\n30" -> "a\n11"[color=red]
                "ag_+@#bh\n30" -> "g_+@#bh\n19"[color=blue]
                "g_+@#bh\n19" -> "g\n9"[color=red]
                "g_+@#bh\n19" -> "_+@#bh\n10"[color=blue]
                "_+@#bh\n10" -> "_+@#\n4"[color=red]
                "_+@#bh\n10" -> "bh\n6"[color=blue]
                "_+@#\n4" -> "_+\n2"[color=red]
                "_+@#\n4" -> "@#\n2"[color=blue]
                "bh\n6" -> "b\n2"[color=red]
                "bh\n6" -> "h\n4"[color=blue]
                "_+\n2" -> "_\n1"[color=red]
                "_+\n2" -> "+\n1"[color=blue]
                "@#\n2" -> "@\n1"[color=red]
                "@#\n2" -> "#\n1"[color=blue]
}
```

6

Sunday, April 2, 2023 9:48 AM




```
## Jagadeesh Vasudevamurthy ####
digraph g {
        label="aaabbgggggghhhhaaaggggaaaaa_+@#"
                "ag_+@#bh\n30" -> "a\n11"[color=red]
                "ag_+@#bh\n30" -> "g_+@#bh\n19"[color=blue]
                "g_+@#bh\n19" -> "g\n9"[color=red]
                "g_+@#bh\n19" -> "_+@#bh\n10"[color=blue]
                "_+@#bh\n10" -> "_+@#\n4"[color=red]
                "_+@#bh\n10" -> "bh\n6"[color=blue]
                "_+@#\n4" -> "_+\n2"[color=red]
                "_+@#\n4" -> "@#\n2"[color=blue]
                "bh\n6" -> "b\n2"[color=red]
                "bh\n6" -> "h\n4"[color=blue]
                "_+\n2" -> "_\n1"[color=red]
                "_+\n2" -> "+\n1"[color=blue]
                "@#\n2" -> "@\n1"[color=red]
                "@#\n2" -> "#\n1"[color=blue]
}
```

7

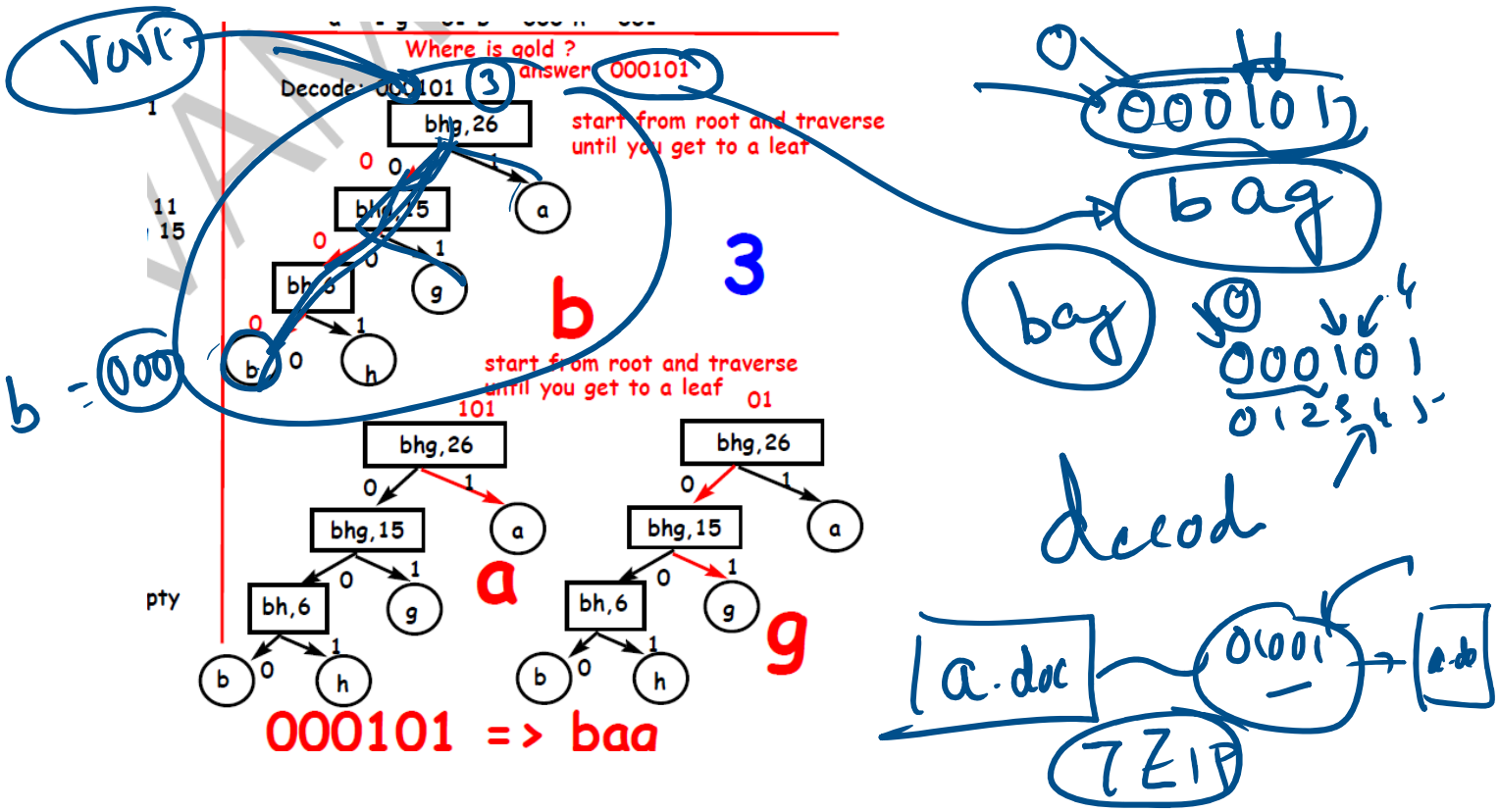
Sunday, April 2, 2023 9:53 AM

V, L, R

```
##############################################
#  Time: THETA(h)
#  Space: THETA(h)
# h height of thetree = O(log n)
##############################################
def _build_code_for_each_character(self, k:'int'):
    level = 0 ;
    code = []
    # the code per node can be never bigger than k, which is the number of node
    for i in range(k):
        code.append(0)
    self._pre_order_traversal_VLR(self._root,level,code)
    if (self._show):
        print("------Step 5: code for each  character is as follows------")
        for key, value in self._char2Stringdict.items():
            print(key,"Code is",value)
```

8

Sunday, April 2, 2023 9:59 AM



9

Sunday, April 2, 2023 10:03 AM

```python
def _decode(self, e:'str')->'str':
    d = ""
    i = 0
    l = len(e)
    while (True):
        [di,i] = self._decode_from_position(e,i)
        d = d + di
        if (i >= l):
            break ;
    if (self._show):
        print("encoded string:",e)
        print("decoded string:",d)
    return d
```

```python
##############################################
#  Time: THETA(length of string)
#  Space: THETA(1)
##############################################
def _decode_from_position(self,e:'str',i:'int')->"string,pos":
    n = self._root
    d = ""
    while (True):
        if (n.is_leaf()):
            return [n.data,i]
        c = e[i]
        i = i + 1
        if (c == "0"):
            n = n.left
        elif (c == "1"):
            n = n.right
        else:
            assert(False)
```

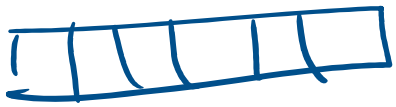
10

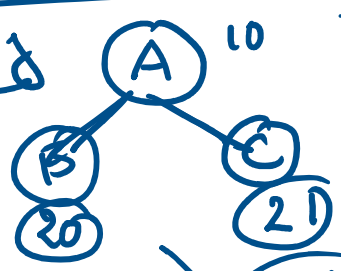
Sunday, April 2, 2023 10:12 AM

Non linear DS

Heap

Complete Binary





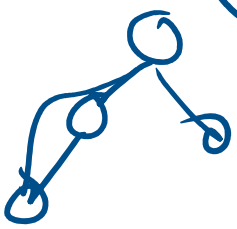
$\log n$

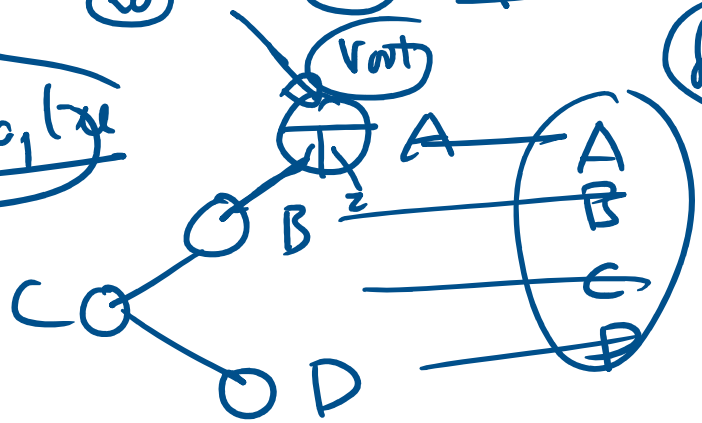
20/12

O(1)

O(log n)

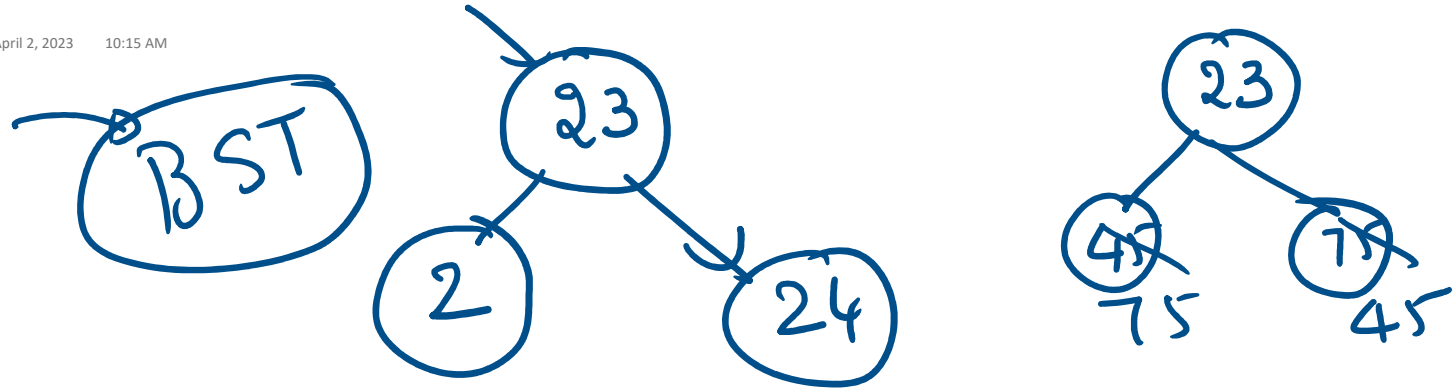
Binary tree





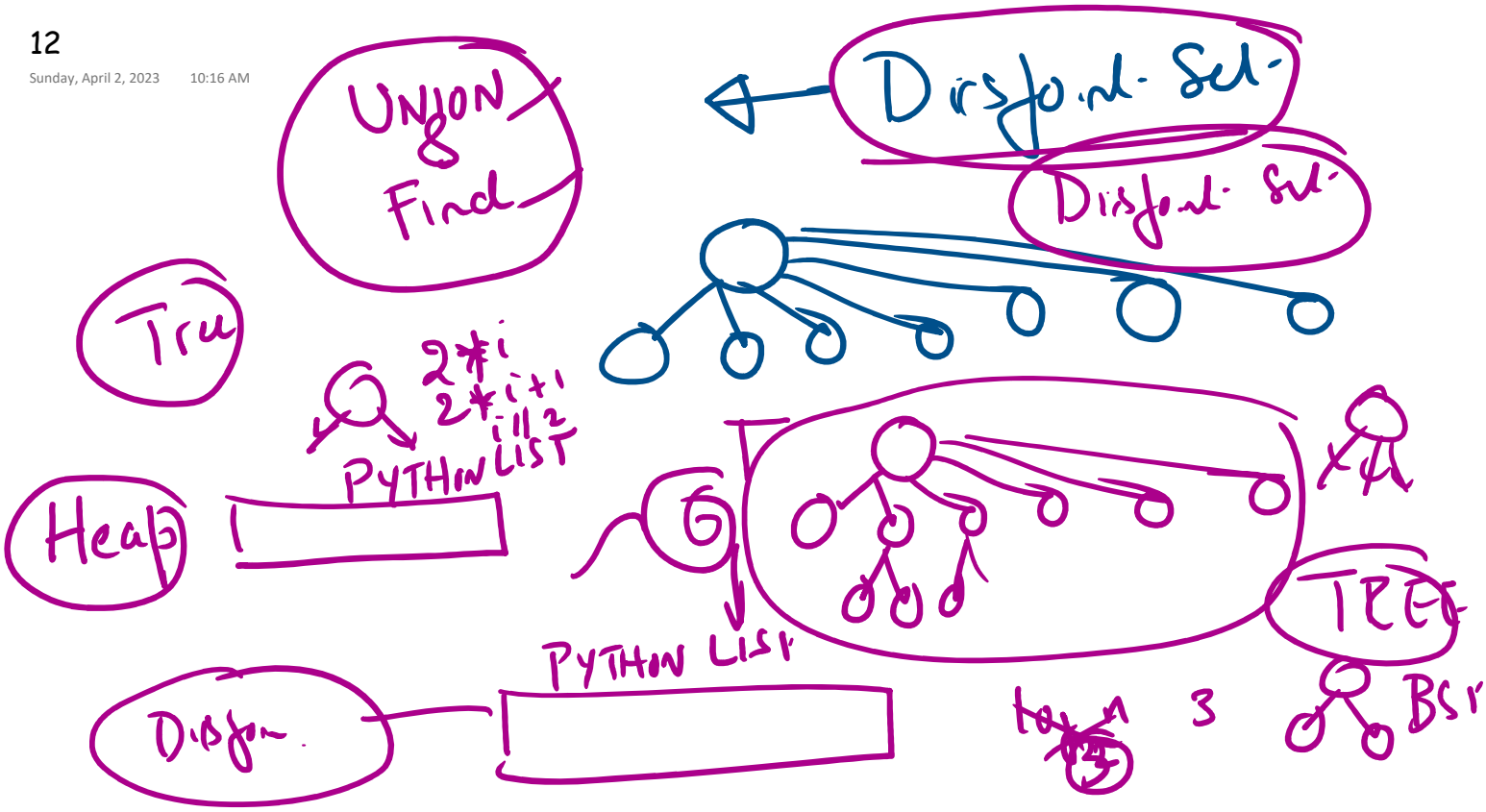
11

Sunday, April 2, 2023 10:15 AM



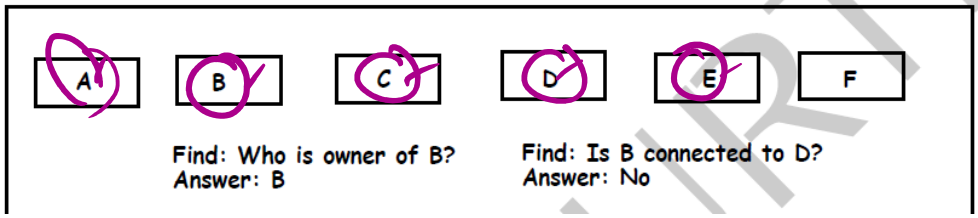
12

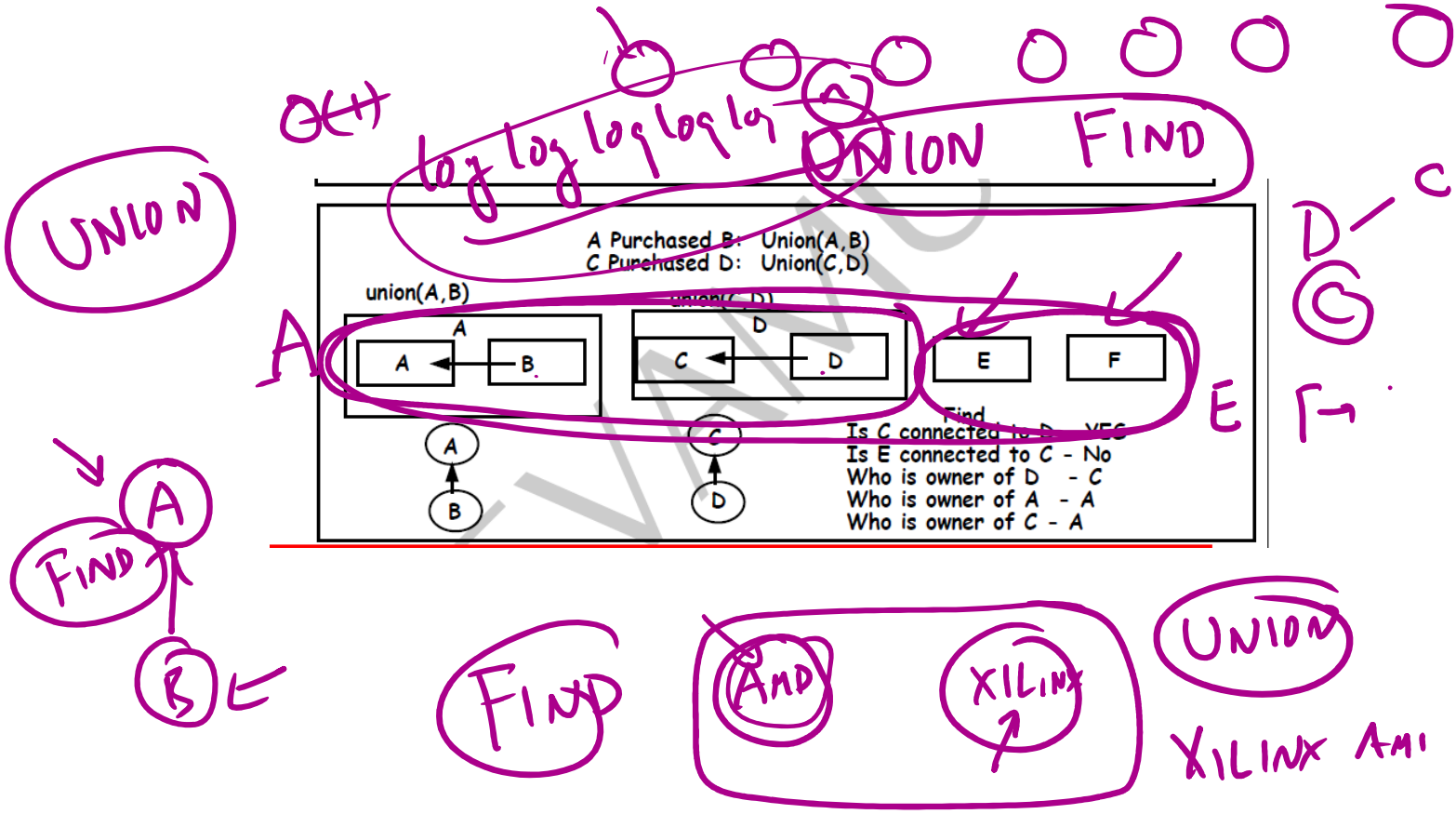
Sunday, April 2, 2023 10:16 AM



13

Sunday, April 2, 2023 10:20 AM





12

Sunday, April 2, 2023 10:25 AM

A is Cousin B

F

Linkedin

1990

UNION FIND

1/2 mill

UNION

Eng

doctor

School

UNION FIND

Disjoint set

log log log

n

n = 6

Count

Dist

13

Sunday, April 2, 2023 10:30 AM

Linked

FAC

PYTHONLIST

UCB

World

log log log log log n

Partiti

We need to do millions of Union and Find operations. The data structure that support union and find is called union-find data structure

Query: Find(B,A) returns yes
Find(C,D) returns yes
Find(B,C) returns yes
Find(D,E) returns no
Find(F,F) returns yes
Find(E,A) returns no

Disjoint sets
1. No item is more than one set
2. Collection of disjoint set is called partition

E

Find

Union

HA

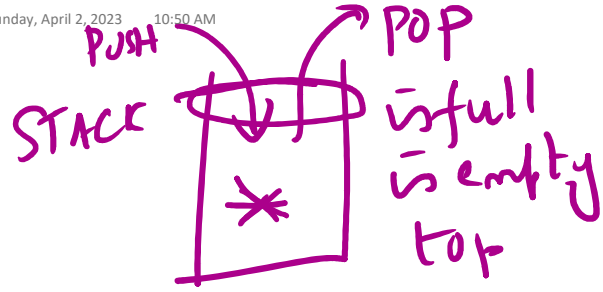
Ackerm

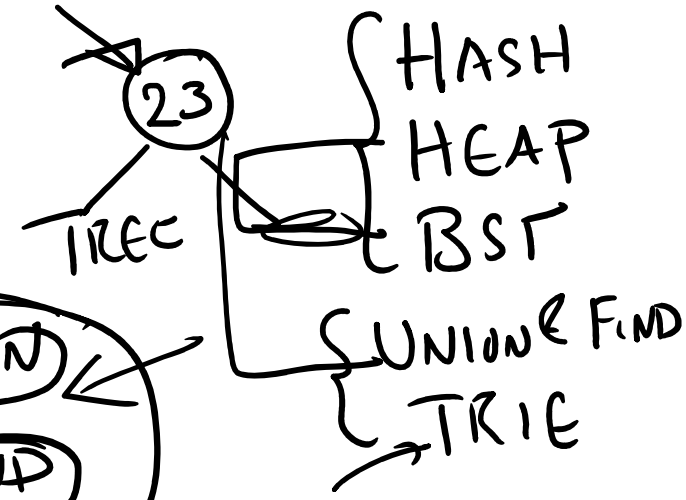
A B C

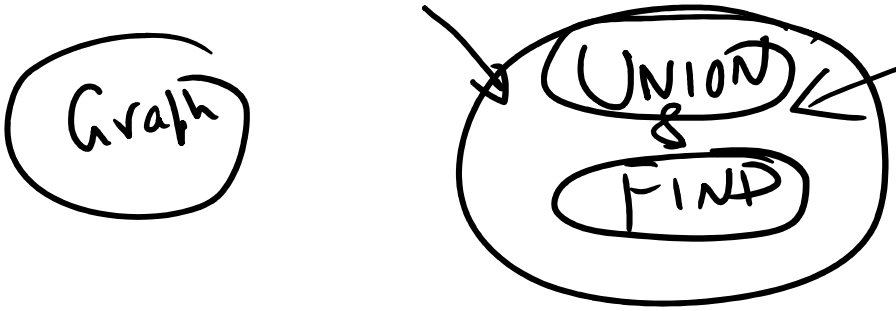
10:50

14

Sunday, April 2, 2023 10:50 AM

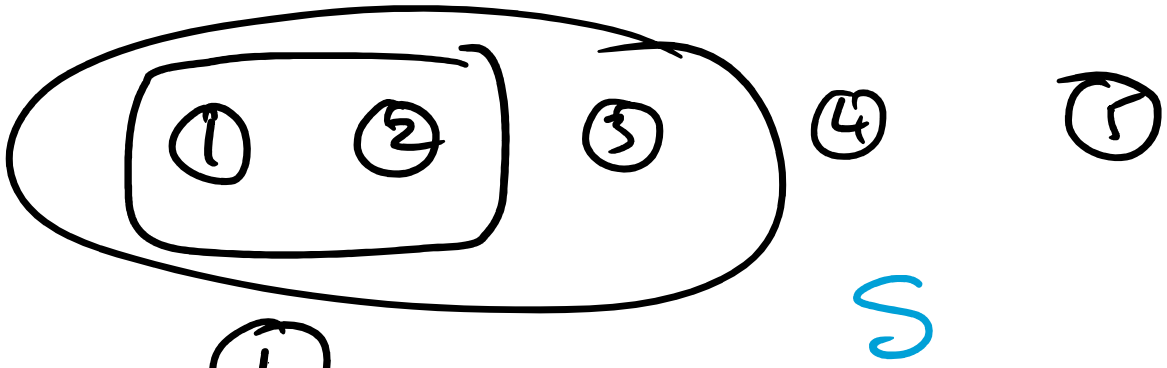




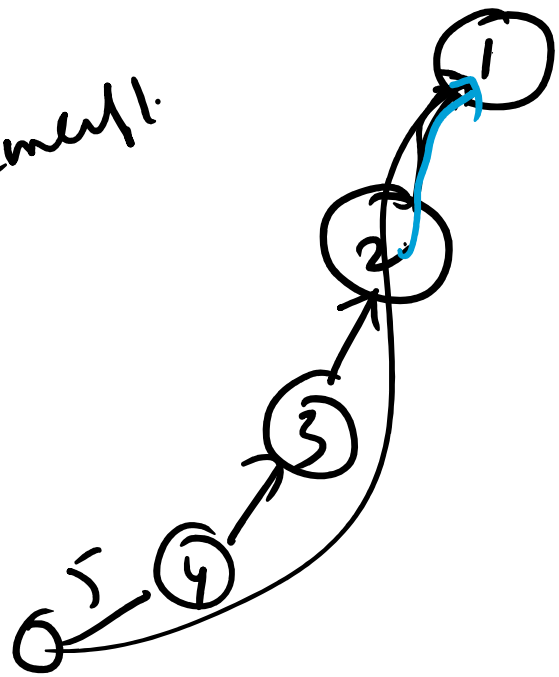


15

Sunday, April 2, 2023 10:54 AM

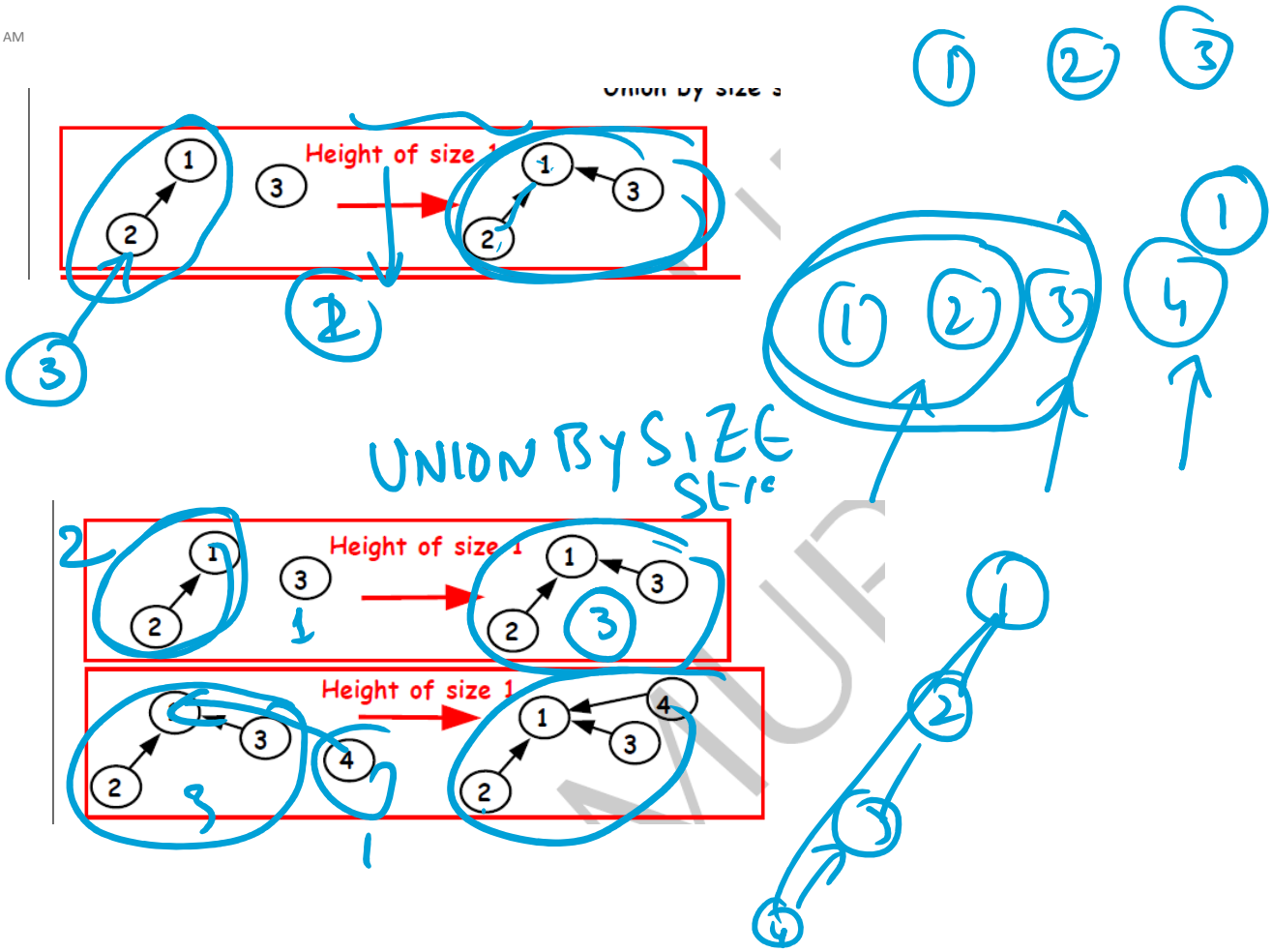


Cmell:



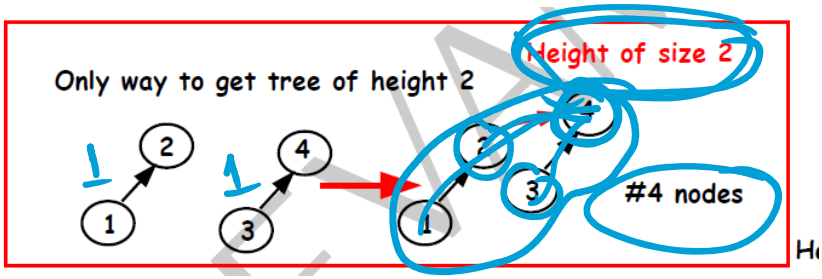
16

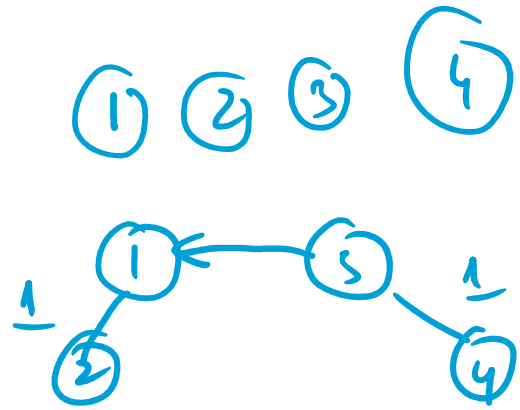
Sunday, April 2, 2023 10:56 AM



17

Sunday, April 2, 2023 10:59 AM

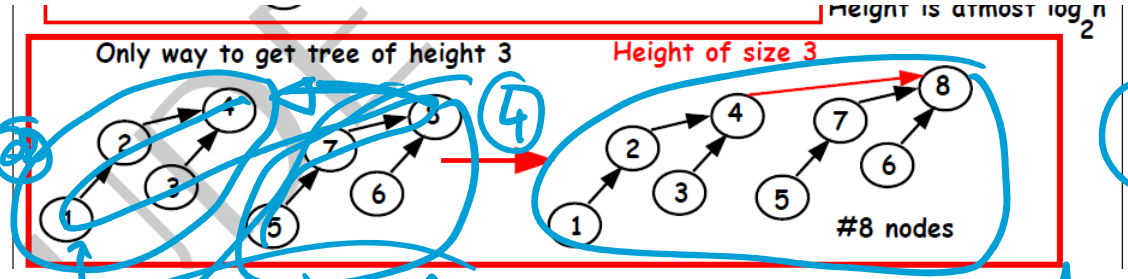




$V(2, 4)$

18

Sunday, April 2, 2023 11:00 AM



8 - 3

4

4

n

$\log_2 n$

1 mill

20

1 Bill

30

log log log log n

0 0 0 0 0 8

185

log n

19

Sunday, April 2, 2023 11:04 AM

n = 1 Billion

8

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

0 1 2 9

OUT FILE

a[8] a[3] 8 a = [ ]

a[400] 1

-1	-1	-1	-1	-1	-1	-1

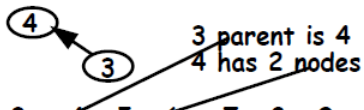
8 8 8 1 3 1

20

Sunday, April 2, 2023 11:07 AM

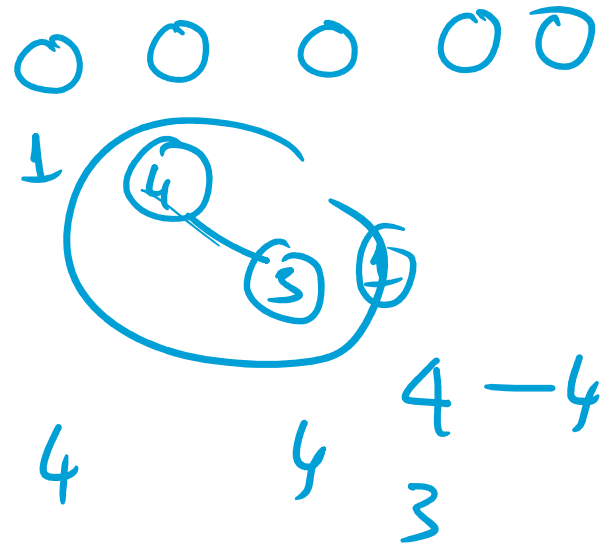
0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

union(4,3)



3 parent is 4
4 has 2 nodes

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	4	-2	-1	-1	-1	-1	-1



UNION WITHOUT FIND

UNION BY SIZE

1 1

30

Sunday, April 2, 2023 11:09 AM

UNION

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	4	-2	-1	-1	-1	-1	-1

HOP

union(4,3)

3 parent is 4
4 has 2 nodes

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	4	-2	-1	-1	-1	-1	-1

-2

a HOP 4 -2

UNION(4, 3)

1 1

3 owner 4

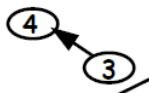
3 —

31

Sunday, April 2, 2023 11:12 AM

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

union(4,3)



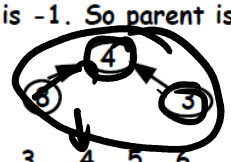
3 parent is 4
4 has 2 nodes

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	4	-2	-1	-1	-1	-1	-1

union(3,8)

3 parent is 4
4 parent is -2. Negative. So 3 parent is 4

8 parent is -1. So parent is 8



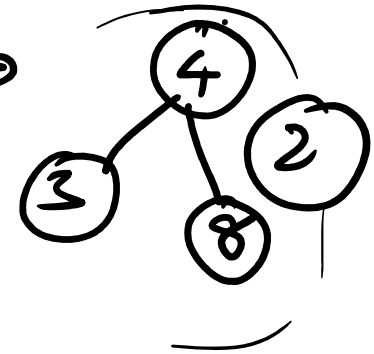
4 has 3 nodes
8 parent is 4
3 parent is 4

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	4	-3	-1	-1	-1	4	-1

-3 4

3 is Son a 4

3 → 4 →



8

8

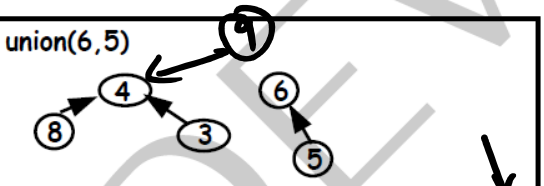
4 UNION 8

8 → 4

32

Sunday, April 2, 2023 11:15 AM

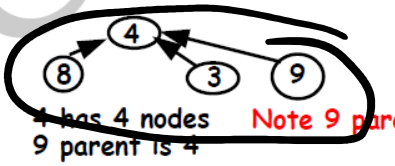
union(6,5)



0	1	2	3	4	5	6	7	8	9
-1	-1	-1	4	-3	6	-2	-1	4	-1

union(9,4) UNION BY SIZE STRATEGY

9 parent is -1, negative. So parent is 9
4 parent is -3, negative. So parent is 4



4 has 4 nodes Note 9 parent is 4
9 parent is 4

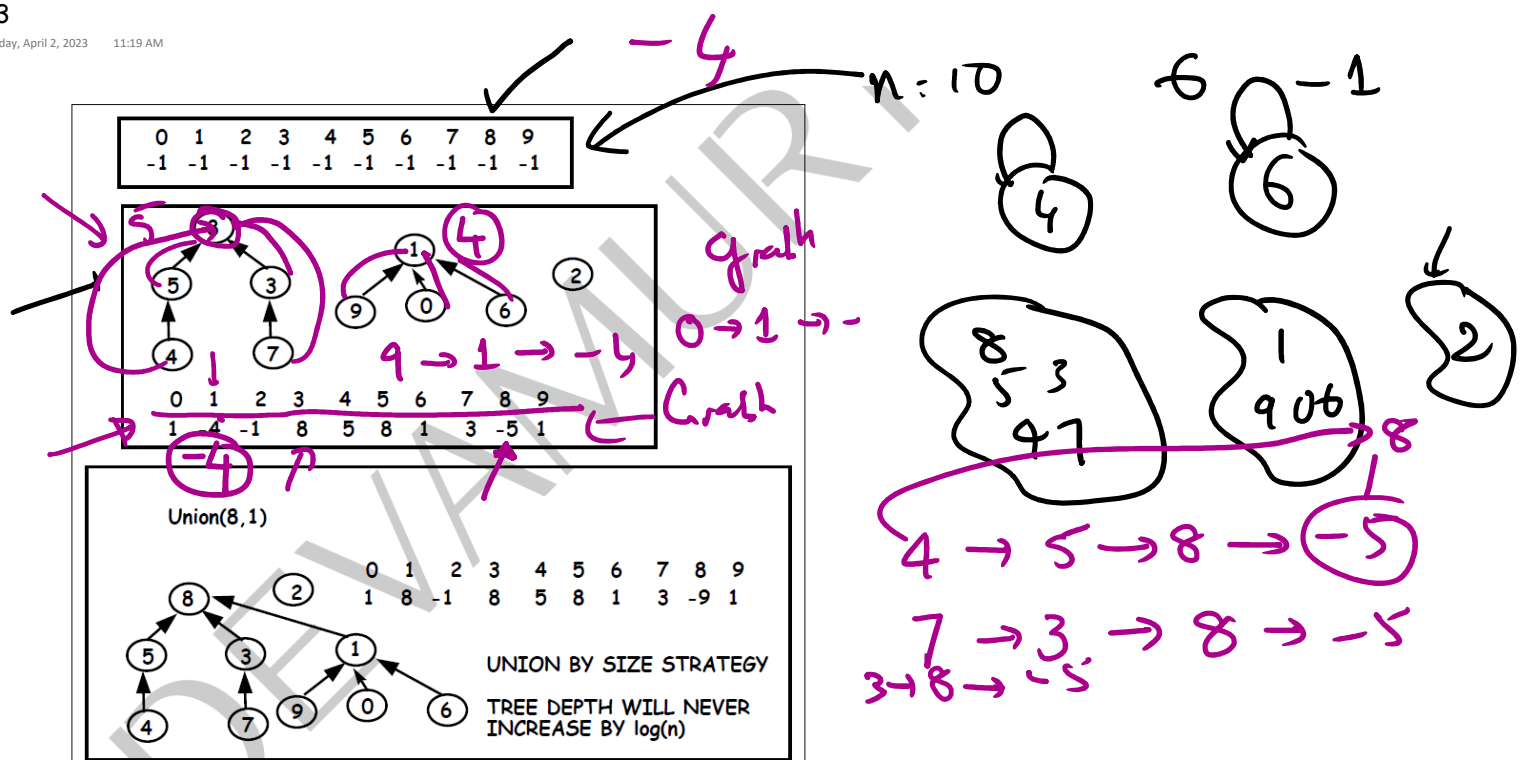
0	1	2	3	4	5	6	7	8	9
-1	-1	-1	4	-4	6	-2	-1	4	4

UNION 9, 4

FIND 9

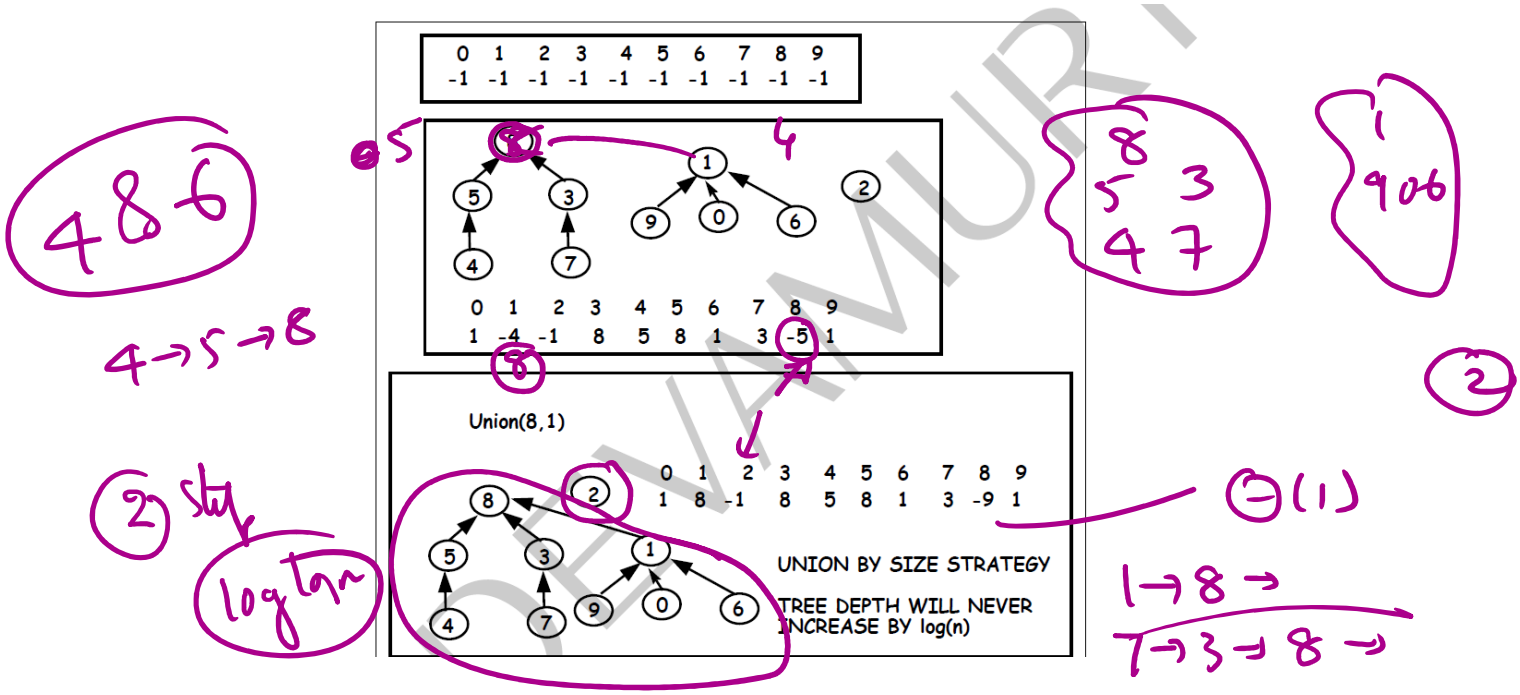
33

Sunday, April 2, 2023 11:19 AM



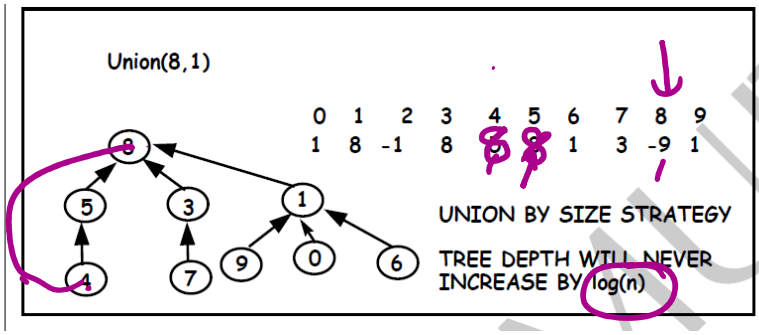
34

Sunday, April 2, 2023 11:23 AM



35

Sunday, April 2, 2023 11:27 AM



8
5 3 1
4 7 9
0 0

4

4 → 5 → 8 → -

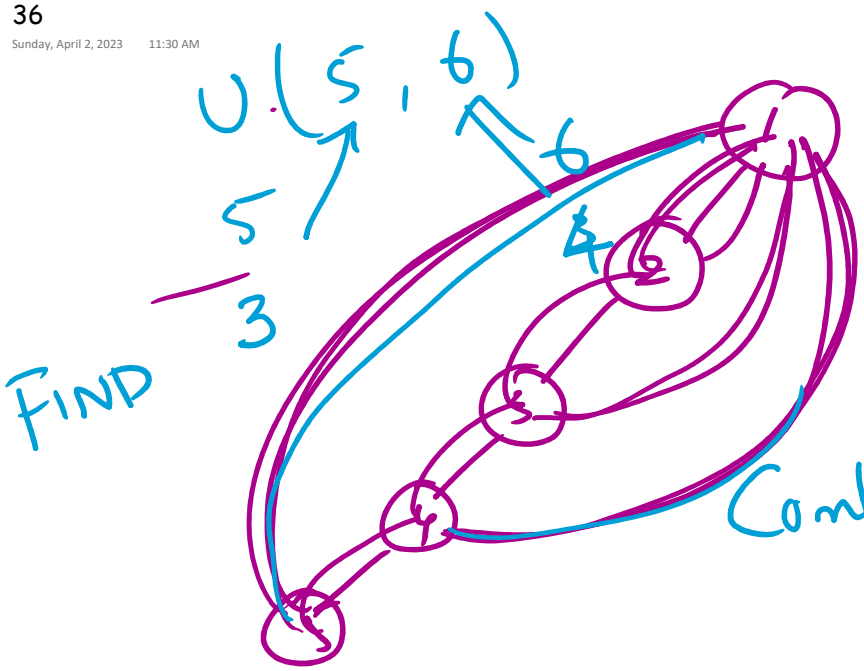
Find(4)

UNION WITHOUT FIND

log n

36

Sunday, April 2, 2023 11:30 AM



log n

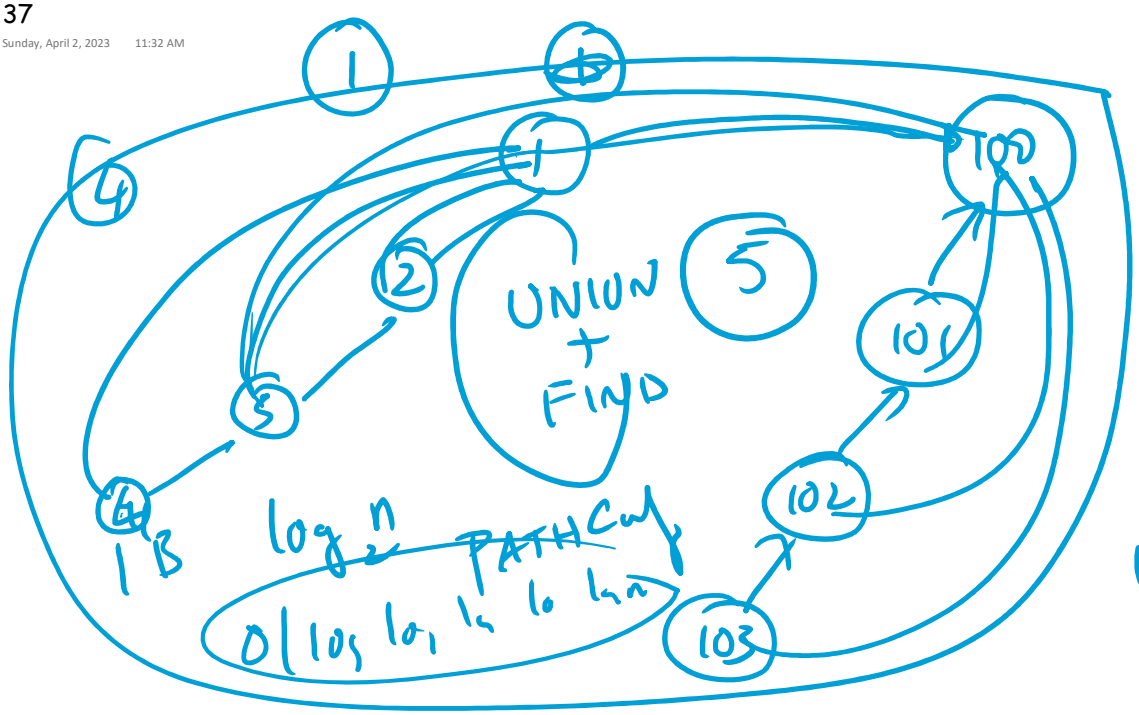
UNION

FIND

Compression ON Demand

37

Sunday, April 2, 2023 11:32 AM



1 HOP

UNION (4, 103)

F(4)

F(103) 100

UNION(3, 101)

100 101

38

Sunday, April 2, 2023 11:40 AM

38

```
#from Util import *
#######################################################
class SUSF():
    def __init__(self,n:'int'):
        #Nothing can be changed here
        self._n = n
        self._id = []
        self._numUnion = 0
        self._numFind = 0
        self._maxHeight = 0
        #he true identity of parent is in a node that has negative weight
        for i in range(n):
            self._id.append(-1)
```

39

Sunday, April 2, 2023 11:44 AM

```
            self._id.append(-1)

    ##########################################
    #  All public functions below
    ##########################################
    def P(self):
        #NOTHING CAN BE CHANGED
        if (self._n < 20):
            u = Util()
            u.print_index(self._n)
            u.print_list(self._id)
        print("U = ",self._numUnion," F = ",self._numFind, " H = ",self._maxHeight) ;

    def H(self):
        #NOTHING CAN BE CHANGED
        return self._maxHeight
```

dol

1 2 0 1 2 3 4

-1 -1

0 0 1 2 3 4

-1 -1 4 -2

0

2

40

Sunday, April 2, 2023 11:48 AM

```
##############################################
# Number of element in group a
##############################################
def num(self,a:'int')->'int':
    print("WRITE CODE. HOW many elements a has")

##############################################
#  Smart Find
#  Who is the parent of a
##############################################
def F1(self,a:'int')->'int':
    print("WRITE CODE. Who is the parent of a")
    print("Mange number of hops and update self._maxHeight ONLY in this routine"

##############################################
#  Smart Find
#  is a connected to b
#  Almost constant
##############################################
def F2(self,a:'int',b:'int')->'bool':
    print("WRITE CODE. Is a connected to b. Hint: Use F1")

##############################################
# Smart Union
# Make union of a and b  if not connected
# union by size
# Almost constant
##############################################
def U(self,a:'int',b:'int')->'bool':
    x = self.F2(a,b)
    y = self.F2(b,a)
    assert(x == y)
    if (x == False):
        print("WRITE CODE")
        return True
    else:
        return False

##############################################
```

num

a, b

5

5 5

5

4 2

3

20

PATH Compression

Slim

num find

a b 20

23

True for F1(3)

-4

5

-1

5

A B

PATH

23

4

10 Code

UNION

True

True

5 Lin UNION

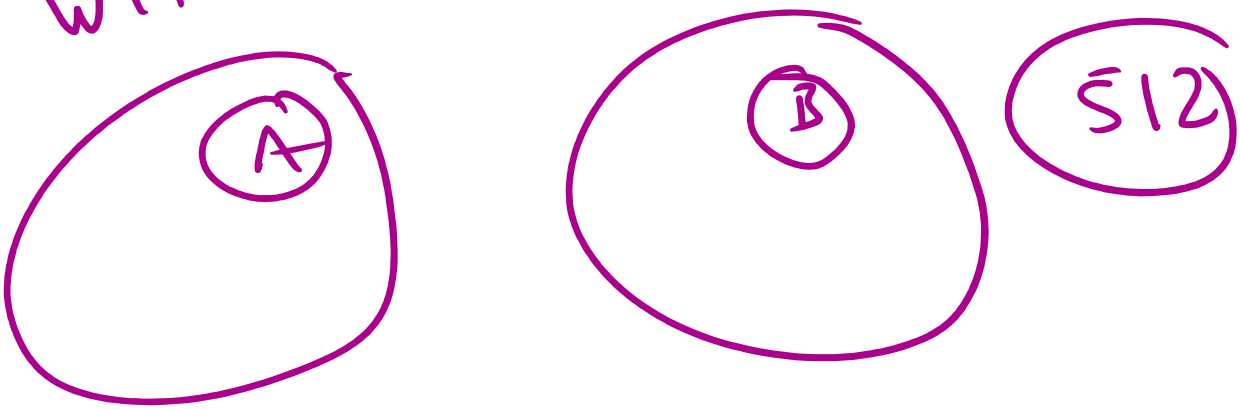
41

Sunday, April 2, 2023 11:55 AM

Merge

UNION WITHOUT FIND

M(512, 256)



42

Sunday, April 2, 2023 11:56 AM

```
#[N,Q]= [int(x) for x in input().split()]
N, Q = map(int,input().strip().split(' '))
ds = [[i,1] for i in range(0,N+1)]
s = SUSF(N)
for q in range(Q):
    inp = input().split()
    if inp[0] == 'M':
        a = int(inp[1])
        b = int(inp[2])
        s.U(a,b)
    else:
        a = int(inp[1])
        print(s.num(a))
```

SUSF(N)

-1	-1	—

S.U(a,b)

num(a)

4