CS 116

Livestream.

Start @ 1:05pm.

```
# (a)  O(n)
# Let n = len(L)
def fn_a(L):
    L1 = list(map(lambda x: x%2, L))   O(n)
    L2 = list(filter(lambda y: y<5, L1)) O(n)   O(n)
    L3 = list(map(lambda z:5-z, L2))  O(n)
    return len(L3)  -> O(1)

# (b) O(2^n)
# Let n = len(s)
def fn_b(s):
    if len(s)==0:  O(1)
        return ""  O(1)      O(n)
    else:          O(1)
        return fn_b(s[1:])+fn_b(s[2:])

# (c) O(n^2)
# Let n = len(L)
def fn_c(L):
    def helper_c(r):
        a = []       O(1)   n-1
        for k in range(len(L)):
            a.append(r)  O(1)   } n times O(n)
        return a
    return list(map(helper_c, L)) O(n * helper-c runtime) = O(n^2)

# (d) O(n^2)
# Let n = len(L)
def fn_d(L, x):
   for i in range(len(L)):
      j=i  O(1)
      while j<len(L):
         if L[i]+L[j]==x:  O(1)
            return i+j     O(1)  (n)  } O(n)    O(n^2)
         j=j+1             O(1)
   return -1

# (e) O(n^2)
# Let n = len(L)
def fn_e(L):
    ans = 0   O(1)
    while L!=[]:
        ans=ans+L[0]
        L=L[1:]
    return ans>100   O(1)
```

$T(n) = O(n) + T(n-1) + T(n-2)$
$= O(2^n)$

if i != 0 : j = len(L)
else : j = i

i	steps
0	4n+2
1	4(n-1)+2
2	4(n-2)+2
⋮	
n	2
sum	$O(n^2)$

L	steps
L	1+1+n-1
L[1:]	1+1+n-2
L[2:]	1+1+n-3
⋮	⋮
[one ele]	1+1+1

$\to O(n^2)$

```
# (f) O(n log n) .
# n is a natural number.
def fn_f(n):
    def helper_f(x):
        while x>1:
            x=x//2
    for k in range(n):
        helper_f(k)
```

$O(\log_2(n))$

$O(n * \text{runtime of helper}) = O(n \log_2 n)$

$= O(n \log n)$

if x=n, helper f cuts n in half as many times as it can.

$2^{\tilde{k}} = n \qquad 2^{\tilde{\kappa}} = x$

$\tilde{k} = \log_2 n . \qquad \tilde{\kappa} = \log_2 x$

"Exact" runtime $\sum_{k=1}^{n} \log_2 k = \log_2 1 + \log_2 2 + \log_2 3 + \dots + \log_2 n$

$= \log_2(n!)$

Stirling's Formula: $n! \approx n^n \quad \approx \log_2(n^n)$

$= n \log_2(n)$

$= n \lg(n)$

```python

#Next Question NlN
def somefun1(x):
    y = x
    while y > 0:
        print(list(range(x)))
        y = y//2
```

$O(x \log x)$

$O(x)$ $O(x)$ } $O(\log x)$ Total $O(x \log x)$

#of ones

x	Step
x	$3x+1$
$x/2$	$3\frac{x}{2}+1$
$x/4$	$3\frac{x}{4}+1$
.	.
1	$3+1$

$$\text{Sum} = \log_2 x + 3\sum_{i=0}^{\log_2 x} \frac{x}{2^i}$$

$$\leq \log_2 x + 3x\sum_{i=0}^{\infty} \frac{1}{2^i}$$

$$= \log_2 x + 3x \cdot 2$$

$$= O(x)$$

```python

#Next Question NlN
def somefun2(x):
    while x > 0:
        print(list(range(x)))
        x = x // 2
```

$O(x)$

$\rightarrow O(x)$

$\rightarrow (O \log_2 x)$

$O(x \log_2 x)$?

```python

#Next Question C
def somefun3(x):
    print(list(range(x % 10)))
```

$O(1)$.

b/w 0 & 9

```python

#End of Big-Oh Notation Questions.


# Question 6:
# Write a function, connected-pieces, that consumes a graph,
#G, (represented using an adjacency matrix) and produces a list
#(in any order) of the connected pieces in G.
# Each connected piece is presented by a list of vertices
#(in any order) for that piece.


#Some DFS examples

#Mergesort - print and discuss.

#Something with File Input/Output.
#Maybe take a file consisting of integers and return the mean, median and mode.

```

fin = open(filename, 'r')

⋮

fin.close()

```python
#Big-Oh notation questions:

# Q25 NN:
# Let n = len(L)
def fn25(L):
    ans1 = list(range(1,len(L),3))    n/3 steps = O(n).
    ans2 = []
    i = 0                  n/3
    while i < len(ans1):                                            list of size ≤ n.
        ans2.append(L[ans1[i]])    O(1)
        ans2.append(sum(L))   O(n)
        ans2.append(len(list(map(lambda x: x==L[0], ans2))))    O(n)
        i = i + 4
    return ans2

#Next question NN:

def fn(n):
  n = n %10+1
  i=0
  while (i < n):
    i = i*2
  return i

#Next question NN:

def fn2(n):
  for i in range(n):
    j=1
    while j < n:
      j +=3
  return

#Next question NN:

def fn3(n):
  s=0
  for i in range(n):
    for j in range(2*i,n):
      s +=1
  return s

#Next Question NrtN:

def fn4(n):
  for i in range(1,n):
    j=n
    while i*i < j:
      j -= 1
  return n
```

ans 2	steps
[ ]	$1+n+2\cdot 2+1+1+1 = 4+n+2\cdot 2$
[3 elets]	$1+n+5\cdot 2+1+1+1 = 4+n+5\cdot 2$
[6 elets]	$= 4+n+8\cdot 2$
[9 elets]	$= 4+n+11\cdot 2$
$\vdots$	$\vdots$
$3\frac{n}{12}$ elets	$= 4+n+\left(\frac{n}{4}+2\right)\cdot 2$

Sum $\leq 4\left(\frac{n}{4}\right) + n\cdot\frac{n}{4} + \frac{n}{4}\left(\frac{n}{4}+2\right)\cdot 2$

$= O(n^2)$.

Lines 9-13 run in O(n * upper bound of one loop).

Line 10 $O(1)$

Line 11 $O(n)$

Line 12 #elets of ans2 $\leq \underset{\text{\#ans2.append per loop}}{3} \cdot \underbrace{\text{len}(\text{ans1})}_{\text{while loop bound}} = 3 \cdot \frac{n}{3} = n$.

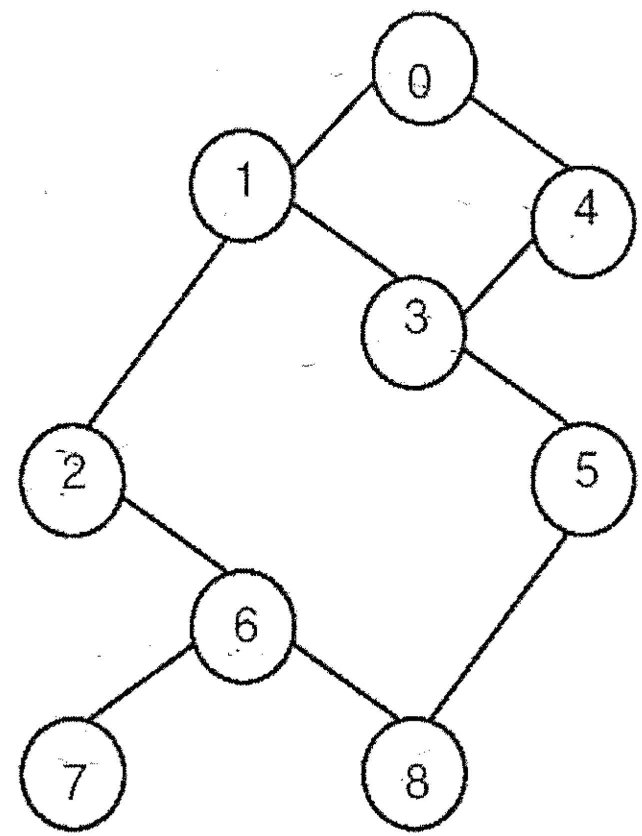
There fore, Line 12 is $O(n)$.

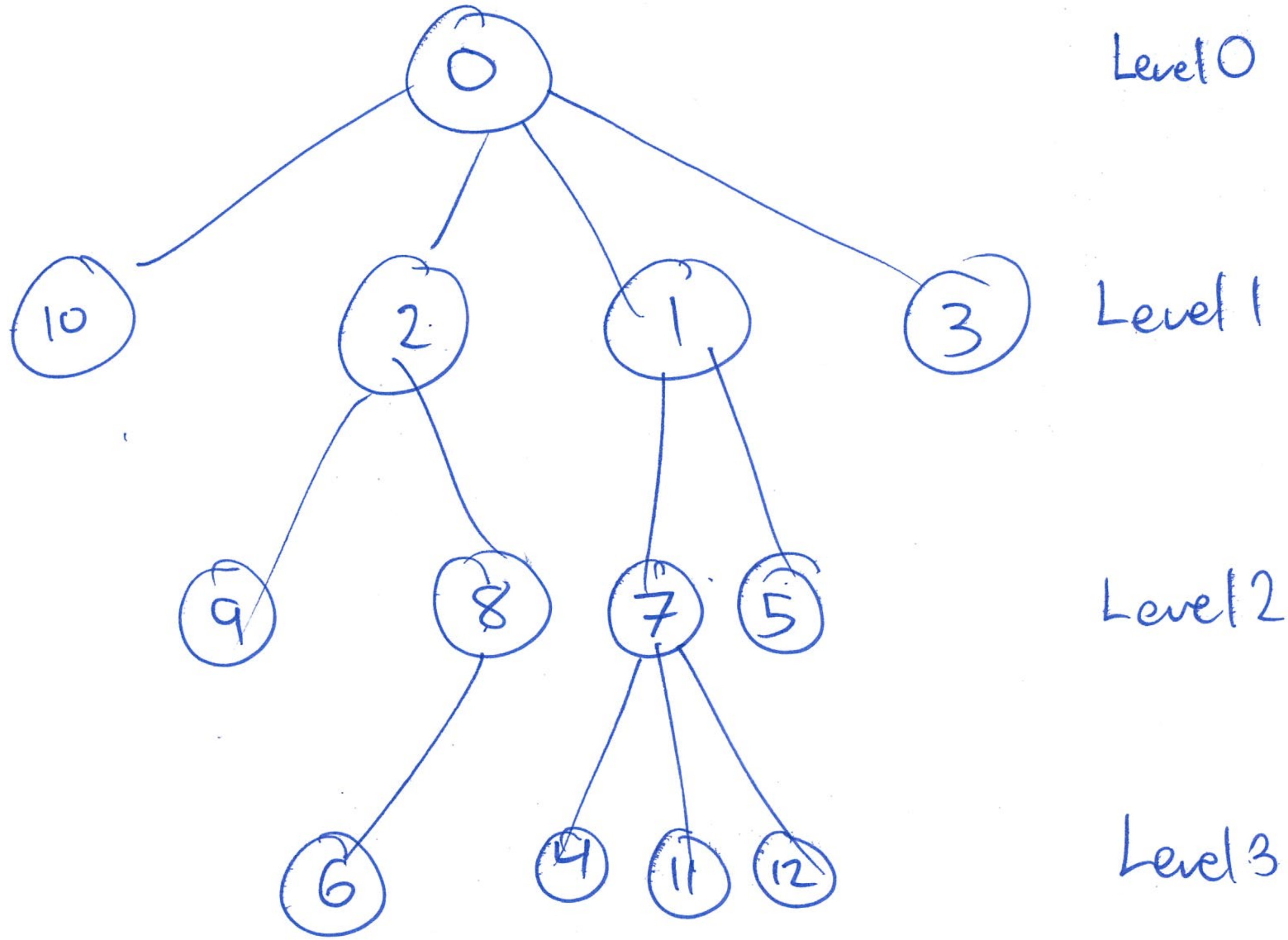
Lines 10-13 take $O(n)$ time.

$\therefore$ Lines 9-13 take $O(n * n) = O(n^2)$ time

Implementation of bfs traversal

```
def bfs(graph, v):
  all = []
  Q = []
  Q.append(v)
  while Q != []:
    v = Q.pop(0)
    all.append(v)
    for n in graph[v]:
      if n not in Q and\
         n not in all:
         Q.append(n)
  return all
```



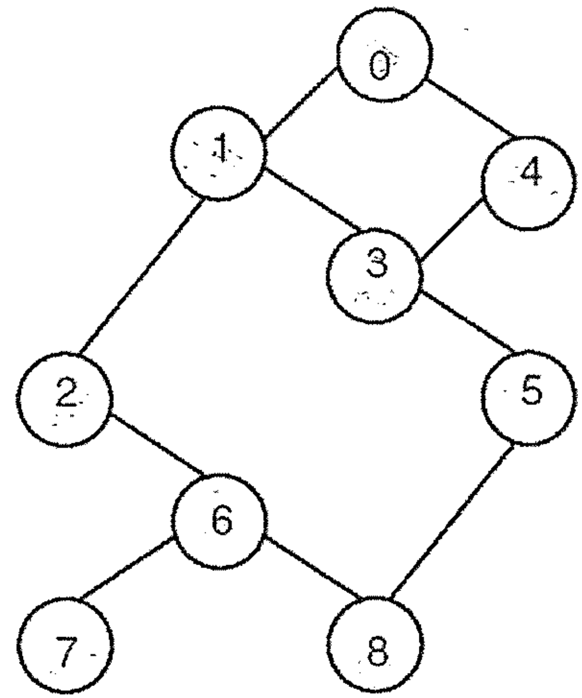


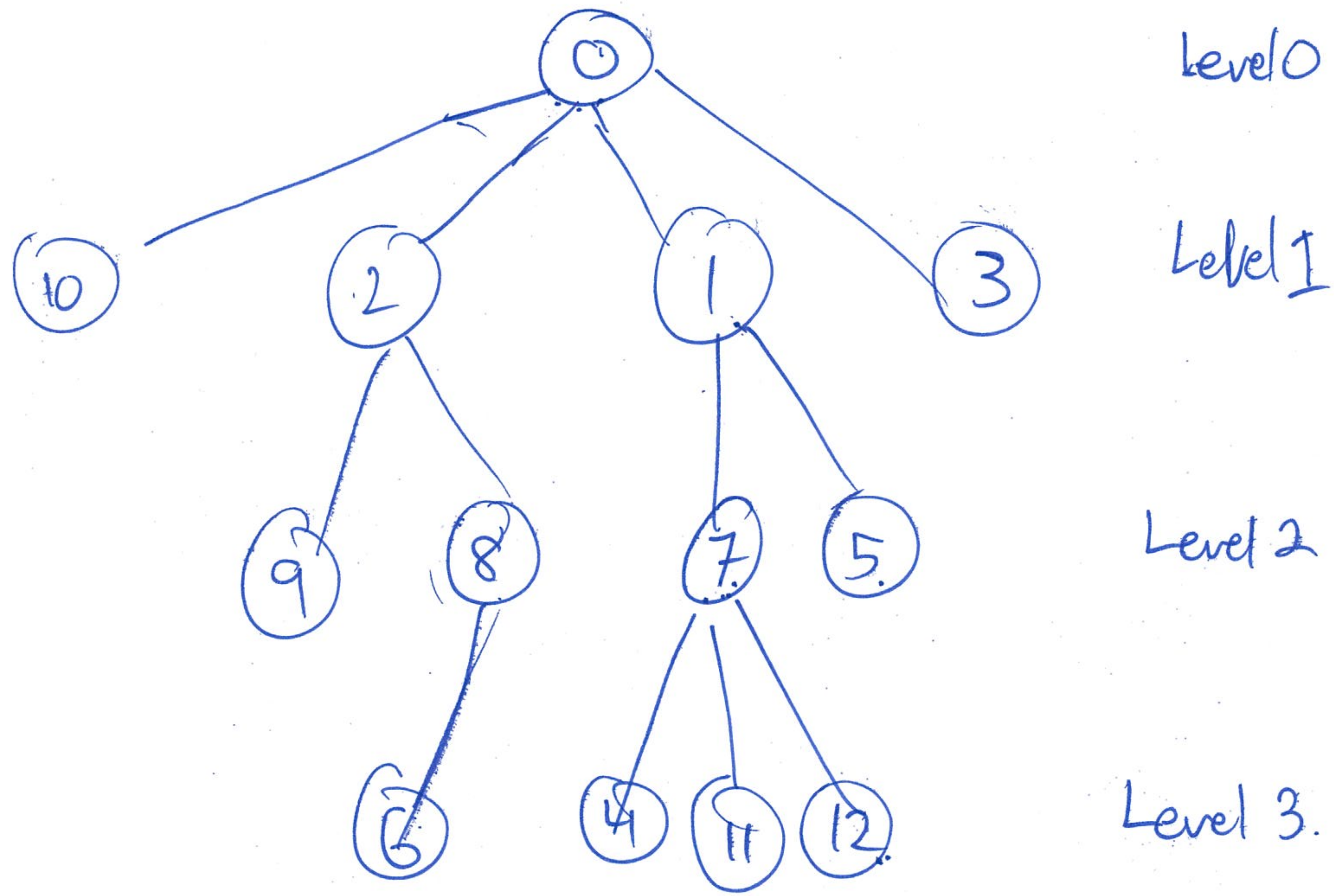
bfs 0, 10, 2, 1, 3, 9, 8, 7, 5, 6, 4, 11, 12.

bfs. 0, 1, 2, 3, 10, 7, 5, 8, 9, 4, 11, 12, 6.

A depth first search traversal solution

```
def dfs(graph, v):
    visited = []
    S = [v]
    while S != []:
        v = S.pop()
        if v not in visited:
            visited.append(v)
            for w in graph[v]:
                if w not in visited:
                    S.append(w)
    return visited
```





dfs 0, 3, 1, 7, 12, 4, 11, 5, 10, 2, 8, 6, 9.

```
def mergesort(L):
    if len(L) < 2: return
    mid = len(L)//2
    L1 = L[:mid]
    L2 = L[mid:]
    mergesort(L1)
    mergesort(L2)
    merge(L1,L2,L)
```

Split `L` into two pieces

Mutate each list into sorted order

Merge the two parts together, and put back in to `L`

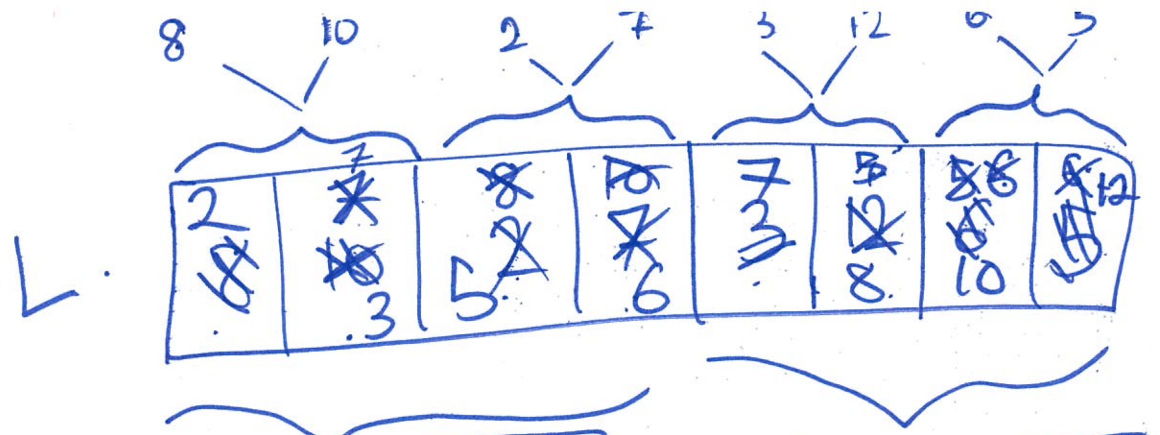
Running time:

$$T(n) = O(n) + 2T\left(\frac{n}{2}\right) \rightarrow O(n \log n)$$

```
def merge(L1,L2,L):
    pos1,pos2,posL = 0,0,0
    while (pos1 < len(L1)) and (pos2 < len(L2)):
        if L1[pos1] < L2[pos2]:
            L[posL] = L1[pos1]
            pos1 += 1
        else:
            L[posL] = L2[pos2]
            pos2 += 1
        posL += 1
    while (pos1 < len(L1)):
        L[posL] = L1[pos1]
        pos1, posL = pos1+1, posL+1
    while (pos2 < len(L2)):
        L[posL] = L2[pos2]
        pos2, posL = pos2+1, posL+1
```

Note: `L1` and `L2` must be sorted before **`merge`** is called, and `L` is combined length of `L1` and `L2`

`pos1`, **`pos2`**, **`posL`** are list positions



$L_1 = $

2	7	8	10

L_2

3	5	6	12

temp = e1
e1 = e2
e2 = temp.