# DAAPY 9

Wednesday, November 1, 2023 4:42 PM

1 ----------- class 9 11/1/2023 ---------- ✓

1

2 ----------- class 10 11/8/2023 ---------- ✓

2

3 ----------- class 11 11/15/2023 ----------

3

----------- class 12 11/22/2023 ----------

NO CLASS. THANKS GIVING FALL BREAK

----------- class 13 11/29/2023 ----------

4

----------- 12/6/2023 ---------- WED

NO CLASS

Dec 10 SUNDAY

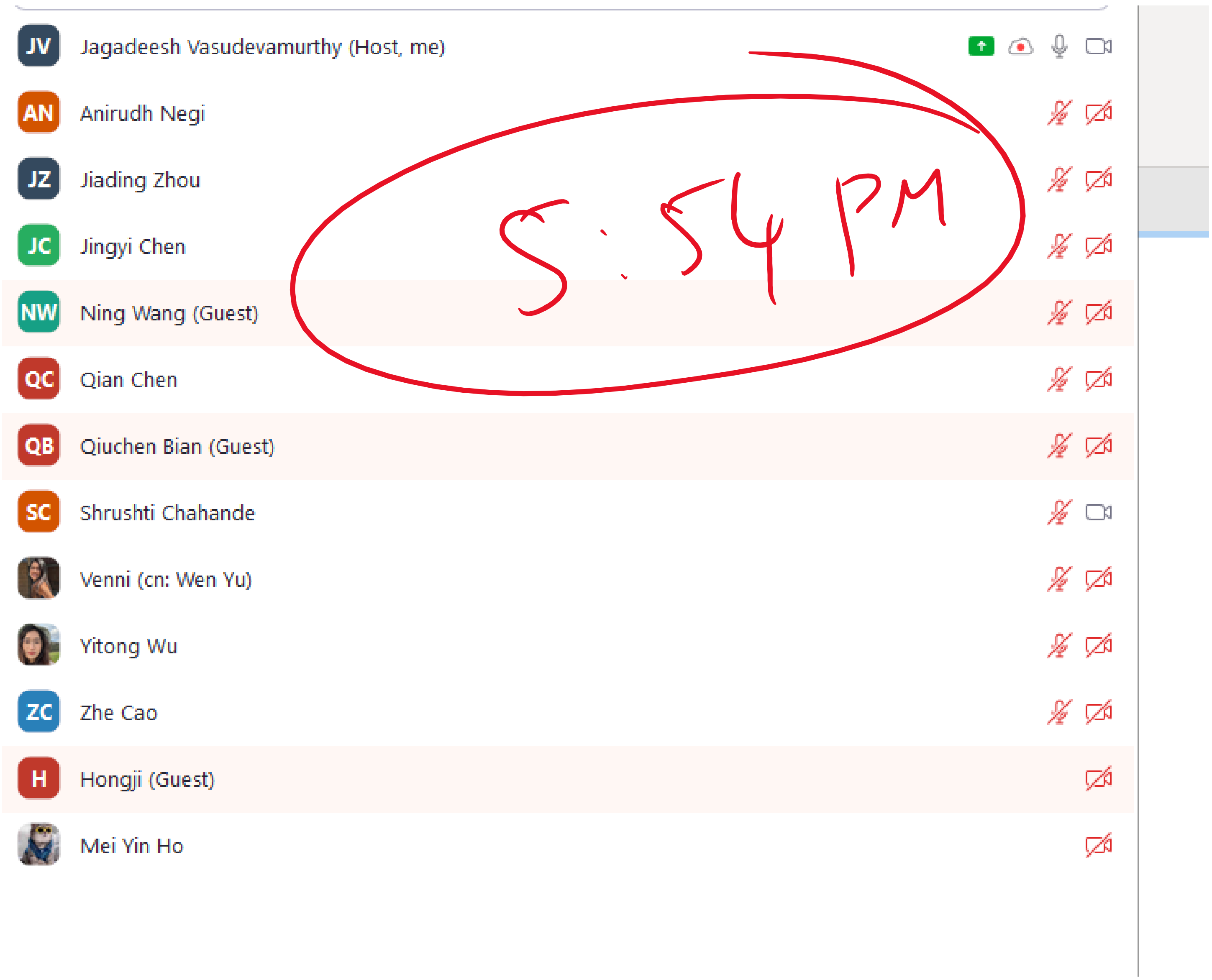
------- class 14 12/10/2023 ---

FINAL

9 AM PST

# attendence

Wednesday, November 1, 2023 5:54 PM



JV Jagadeesh Vasudevamurthy (Host, me)

AN Anirudh Negi

JZ Jiading Zhou

JC Jingyi Chen

NW Ning Wang (Guest)

QC Qian Chen

QB Qiuchen Bian (Guest)

SC Shrushti Chahande

Venni (cn: Wen Yu)

Yitong Wu

ZC Zhe Cao

H Hongji (Guest)

Mei Yin Ho

5:54 PM

# complexity

Wednesday, November 1, 2023 5:41 PM

| | BUS [ ]<br>LIST | Train<br>SLIST<br>2 ptr | HASH | Heap LIST [ ] |
|---|---|---|---|---|
| Insert | append $\Theta(1)$<br>prepend $\Theta(n)$<br>Anywhere $\Theta(n)$ | app $\Theta(1)$<br>P $\Theta(1)$<br>$\Theta(n)$ | $\Theta(1)$ | $O(\log n)$ |
| a[i] | $\Theta(1)$ | $O(n)$ | X | X |
| FIND | $\Theta(n)$ | $\Theta(n)$ | $O(1)$ | $O(1)$ |
| delete | $\Theta(n)$ | FIRST $\Theta(1)$<br>$\Theta(n)$<br>$O(n)$ | $O(1)$ | $O(\log n)$ |
| MIN | $\Theta(n)$ | $\Theta(n)$ | $\Theta(n)$ | $\Theta(1)$ |
| MAX | $\Theta(n)$ | $\Theta(n)$ | $\Theta(n)$ | $\Theta(1)$ |

# 1

Wednesday, November 1, 2023 5:04 PM

1
2
3

$\{1\ \ 2\ \ 3\ \ 4\}$

$n = 4$

Brute-force

Truth table

$\{2, 5, 6, 3\}$ (0 1 2 3)

$n * 2^n$

Brute force

$2^n * n$

Gener

| n | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 = 3 |
| 0 | 0 | 1 | 0 = 6 |
| ~~0~~ | ~~0~~ | ~~1~~ | ~~1~~ |
| 0 | 1 | 0 | 0 = 5 |
| 0 | 1 | 0 | 1 = 8 |
| ~~0~~ | ~~1~~ | ~~1~~ | ~~0~~ |
| ~~0~~ | ~~1~~ | ~~1~~ | ~~1~~ |
| 1 | 0 | 0 | 0 = 2 |
| 1 | 0 | 0 | 1 = 5 |
| 1 | 0 | 1 | 0 = 8 |
| ~~1~~ | ~~0~~ | ~~1~~ | ~~1~~ |
| ~~1~~ | ~~1~~ | ~~0~~ | ~~0~~ |
| ~~1~~ | ~~1~~ | ~~0~~ | ~~1~~ |
| ~~1~~ | ~~1~~ | ~~1~~ | ~~0~~ |
| ~~1~~ | ~~1~~ | ~~1~~ | ~~1~~ |

$4 \rightarrow 2^4 = 16$

| $n$ | $2^n$ |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

$\{80\}$
0
80

$2^{20}$

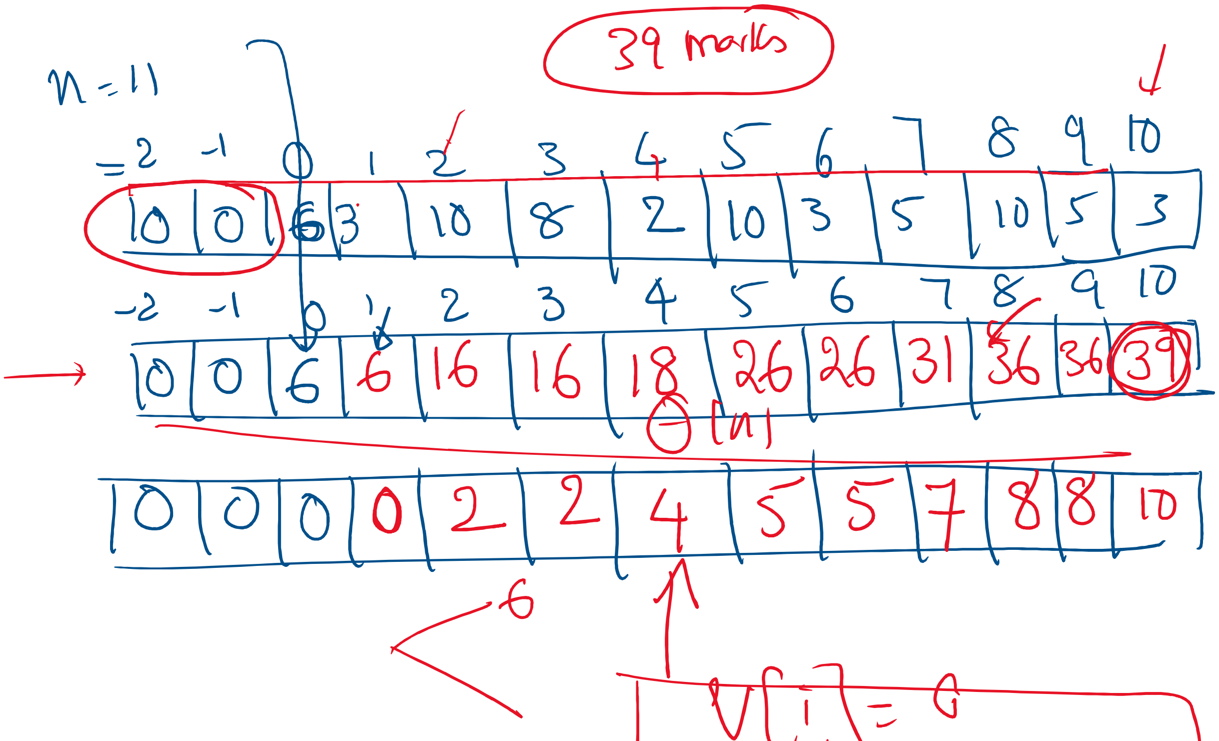
20

$2^n$

# 2

Wednesday, November 1, 2023 5:12 PM

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 3 | 10 | 8 | 2 | 10 | 3 | 5 | 10 | 5 | 3 |



39 marks

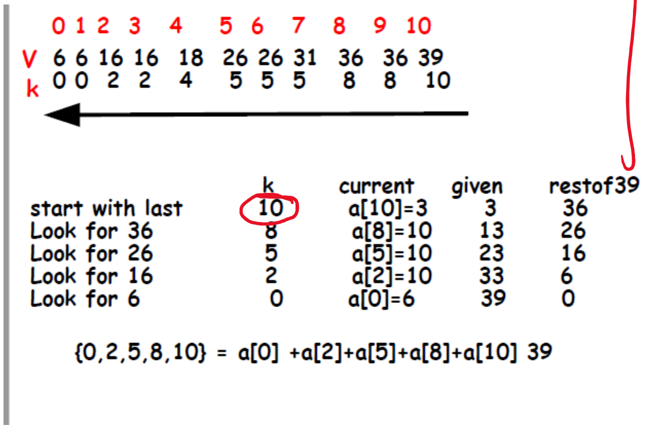
n=11

|   | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|   | 10 | 0 | 6 | 3 | 10 | 8 | 2 | 10 | 3 | 5 | 10 | 5 | 3 |
| V | 10 | 0 | 6 | 6 | 16 | 16 | 18 | 26 | 26 | 31 | 36 | 36 | 39 |
| K | 10 | 0 | 0 | 0 | 2 | 2 | 4 | 5 | 5 | 7 | 8 | 8 | 10 |

$\Theta(n)$

6

$$V[i] = \max(V[i-1],\ a[i] + V[i-2])$$



|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| V | 6 | 6 | 16 | 16 | 18 | 26 | 26 | 31 | 36 | 36 | 39 |
| k | 0 | 0 | 2 | 2 | 4 | 5 | 5 | 5 | 8 | 8 | 10 |

|   | k | current | given | restof39 |
|---|---|---|---|---|
| start with last | 10 | a[10]=3 | 3 | 36 |
| Look for 36 | 8 | a[8]=10 | 13 | 26 |
| Look for 26 | 5 | a[5]=10 | 23 | 16 |
| Look for 16 | 2 | a[2]=10 | 33 | 6 |
| Look for 6 | 0 | a[0]=6 | 39 | 0 |

{0,2,5,8,10} = a[0] +a[2]+a[5]+a[8]+a[10] 39

# 3

Wednesday, November 1, 2023 5:23 PM

$\Theta(n)$

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| V | 6 | 6 | 16 | 16 | 18 | 26 | 26 | 31 | 36 | 36 | 39 |
| k | 0 | 0 | 2 | 2 | 4 | 5 | 5 | 5 | 8 | 8 | 10 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 3 | 10 | 8 | 2 | 10 | 3 | 5 | 10 | 5 | 3 |

i, i+1

| | k | current | given | restof39 |
|---|---|---|---|---|
| start with last | 10 | a[10]=3 | 3 | 36 |
| Look for 36 | 8 | a[8]=10 | 13 | 26 |
| Look for 26 | 5 | a[5]=10 | 23 | 16 |
| Look for 16 | 2 | a[2]=10 | 33 | 6 |
| Look for 6 | 0 | a[0]=6 | 39 | 0 |

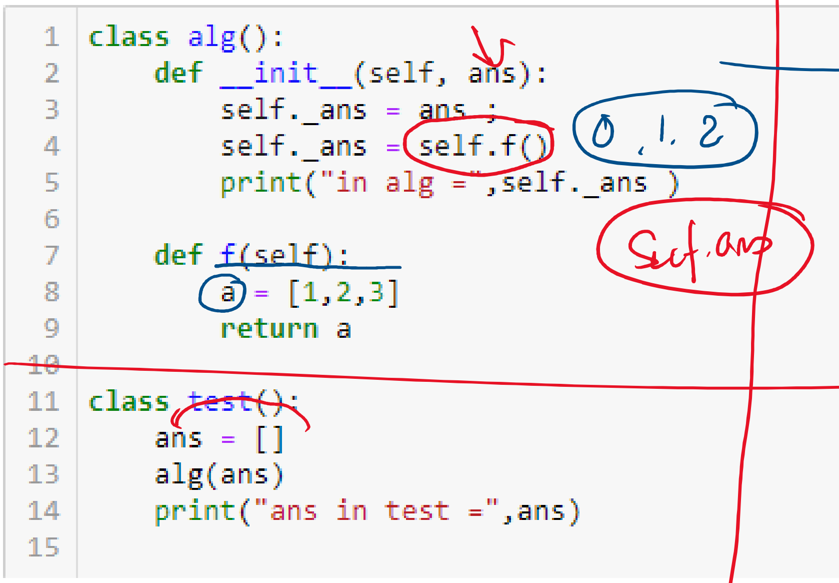
One of the Best

{0,2,5,8,10} = a[0] +a[2]+a[5]+a[8]+a[10] 39 = 39

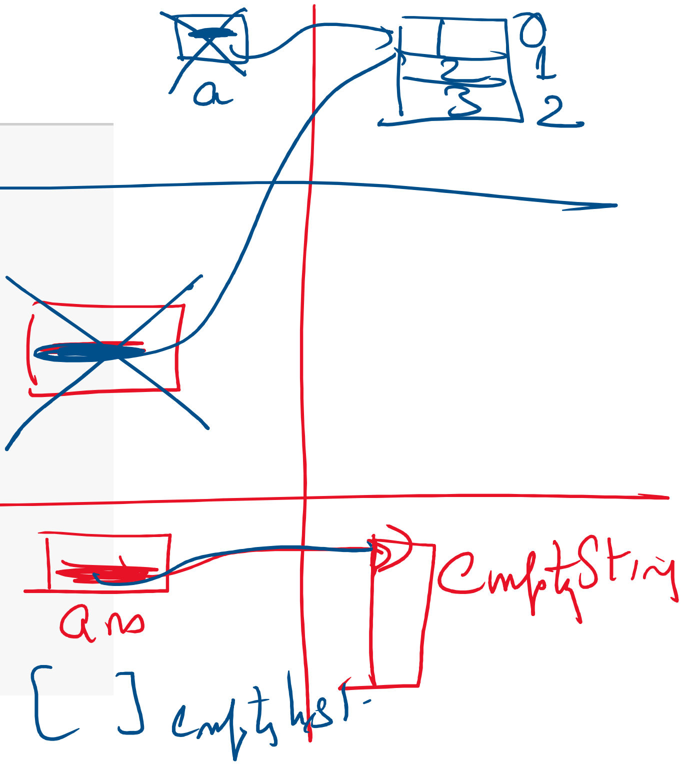
-an alphabet =

Set = {10, 8, 5, 2, 0}

39
3
36
10
26
10
16
10
6 6 | 0

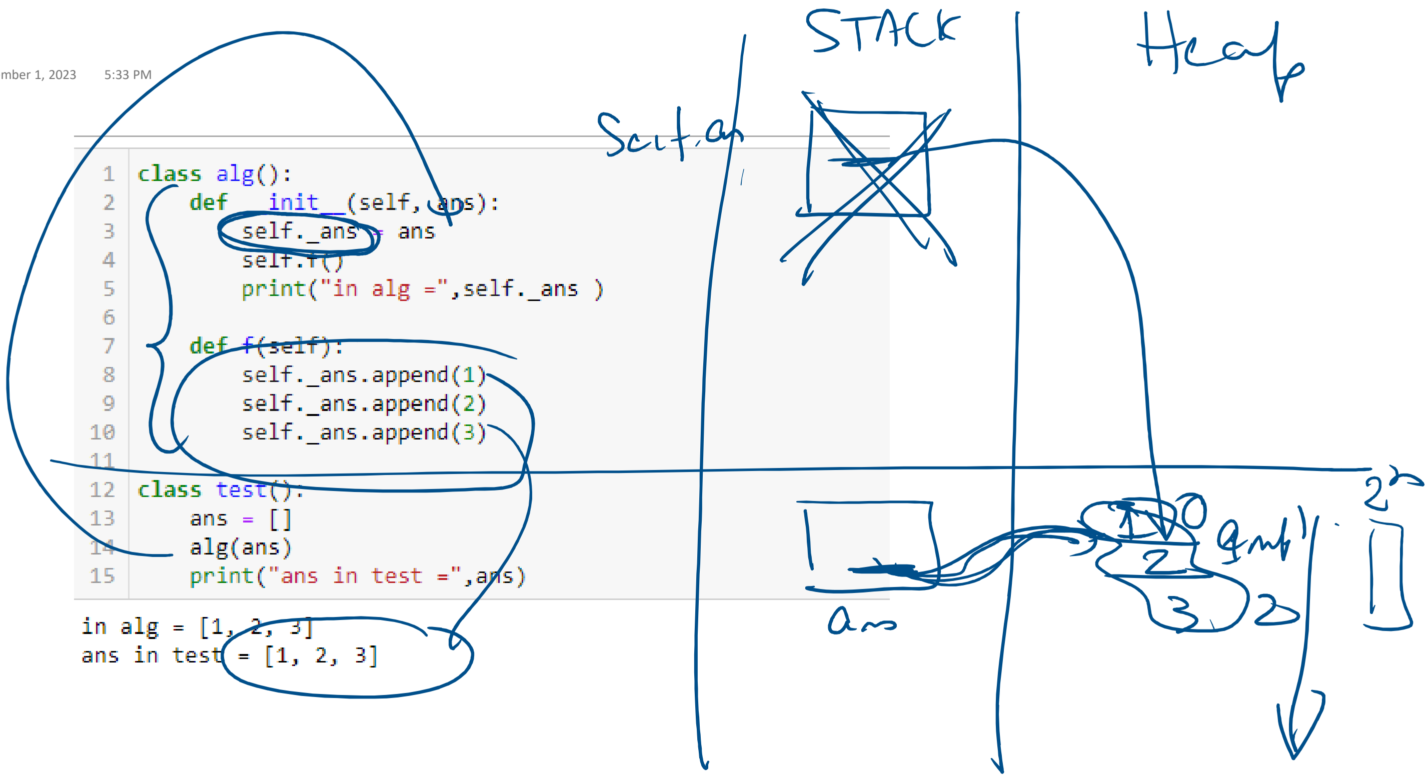


```
class alg():
    def __init__(self, ans):
        self._ans = ans ;
        self._ans = self.f()
        print("in alg =",self._ans )

    def f(self):
        a = [1,2,3]
        return a

class test():
    ans = []
    alg(ans)
    print("ans in test =",ans)
```



# 5

Wednesday, November 1, 2023 5:33 PM

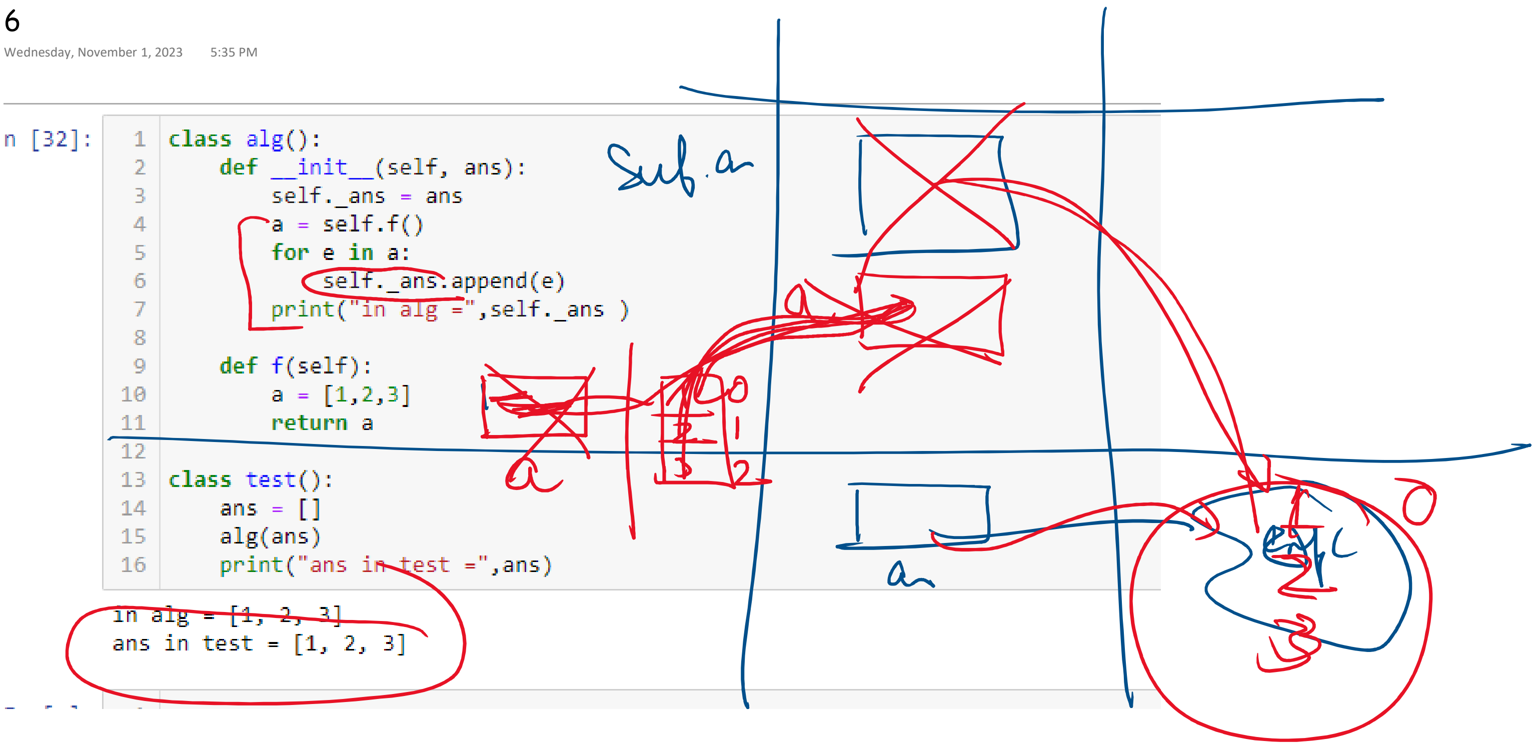


```python
class alg():
    def __init__(self, ans):
        self._ans = ans
        self.f()
        print("in alg =",self._ans )

    def f(self):
        self._ans.append(1)
        self._ans.append(2)
        self._ans.append(3)

class test():
    ans = []
    alg(ans)
    print("ans in test =",ans)
```

```
in alg = [1, 2, 3]
ans in test = [1, 2, 3]
```

# 6

Wednesday, November 1, 2023 5:35 PM



```
n [32]:
class alg():
    def __init__(self, ans):
        self._ans = ans
        a = self.f()
        for e in a:
            self._ans.append(e)
        print("in alg =",self._ans )

    def f(self):
        a = [1,2,3]
        return a

class test():
    ans = []
    alg(ans)
    print("ans in test =",ans)
```

```
in alg = [1, 2, 3]
ans in test = [1, 2, 3]
```

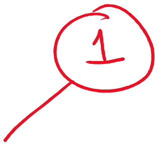
# 7

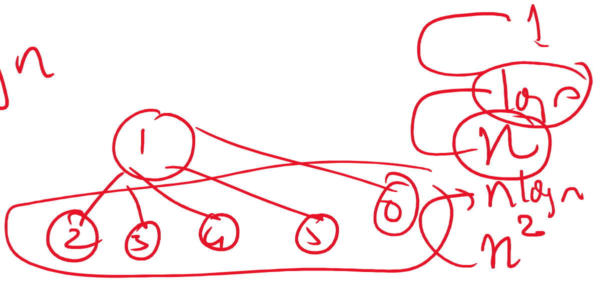
Wednesday, November 1, 2023 5:46 PM

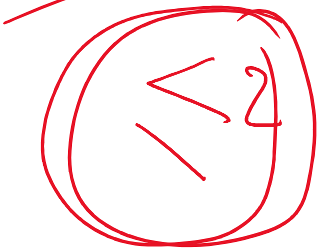
HEAP TREE

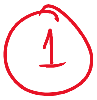
Non-linear DATA STructure

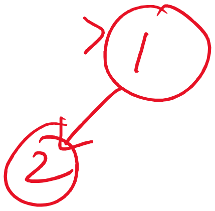
$n \longrightarrow \log n$

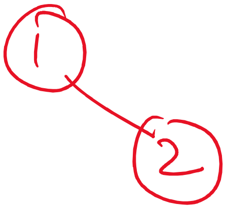


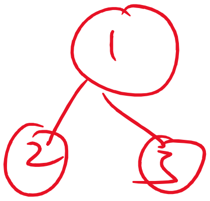


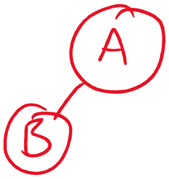


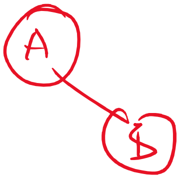


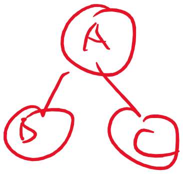


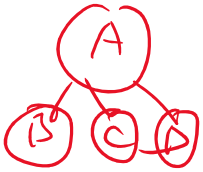






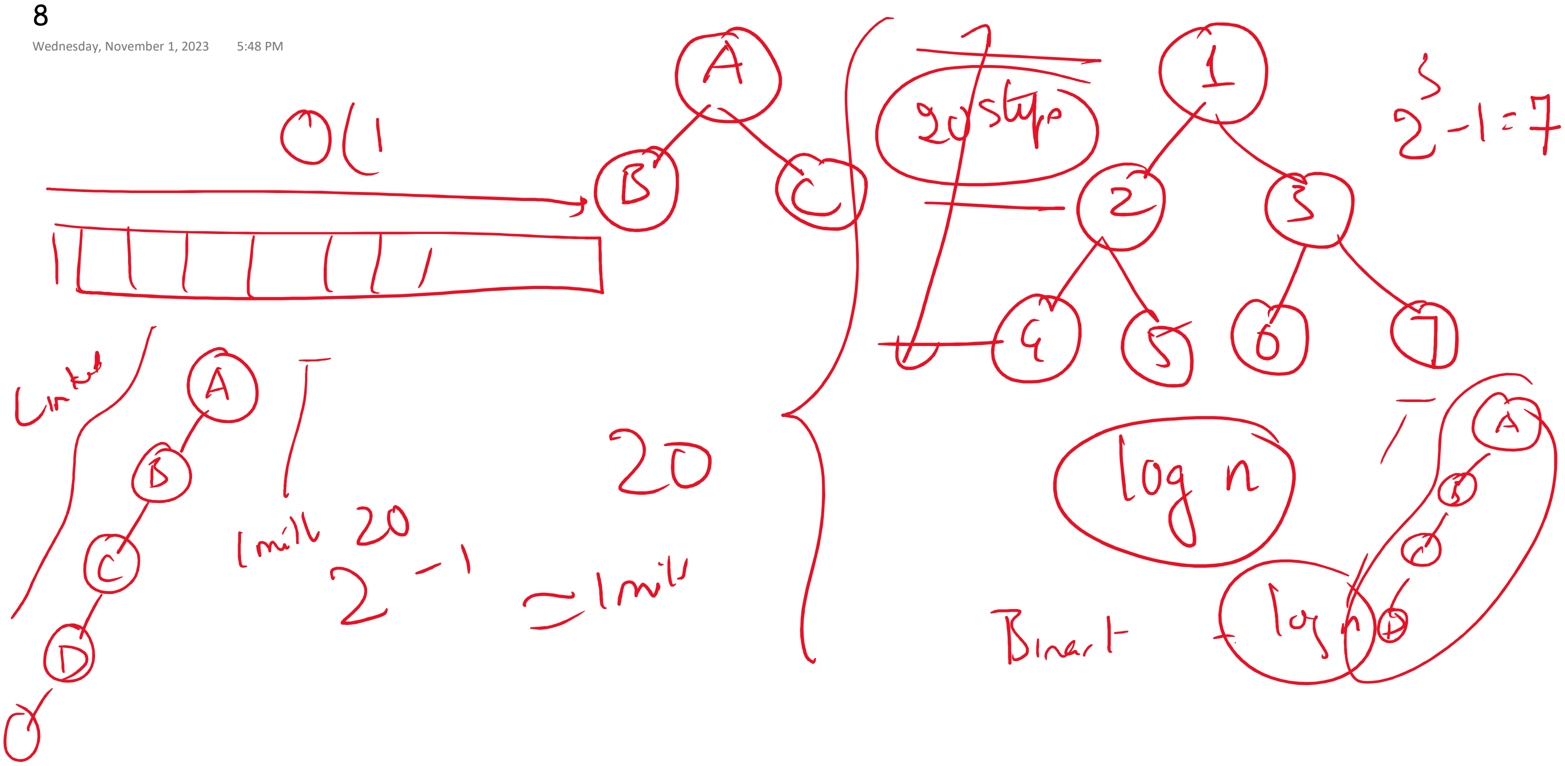






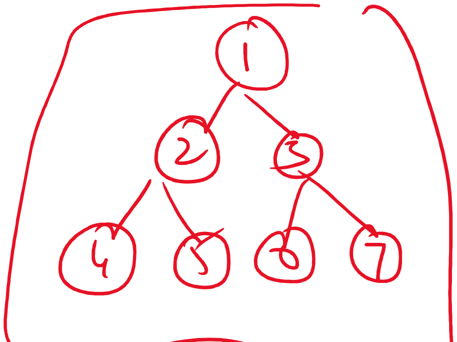
# 8

Wednesday, November 1, 2023 5:48 PM



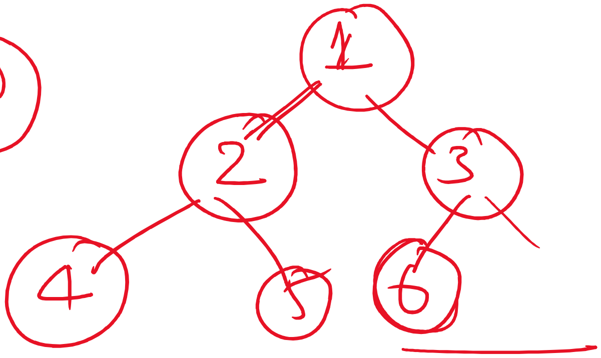
# 9

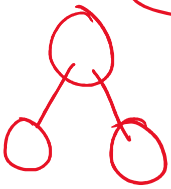
Wednesday, November 1, 2023 5:51 PM



$n=1$
$n=3$
$n=7$
$n=15$

$n=10$





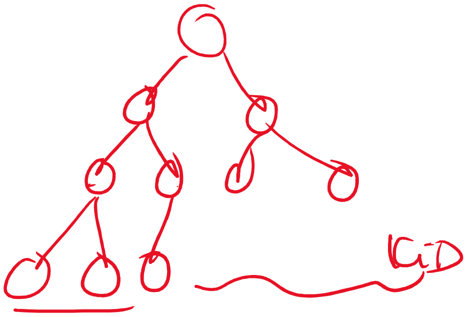
Heap

$n=100$

$n=6$

$n$

Heap



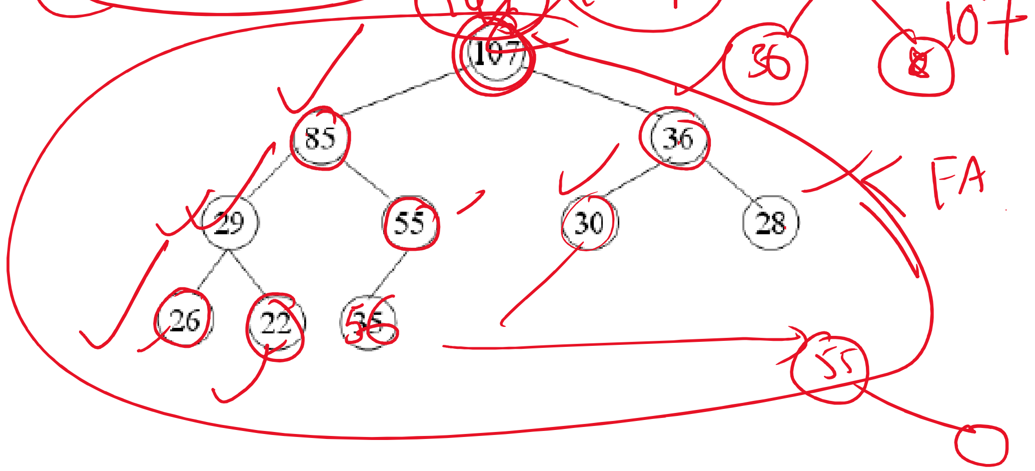
# 10

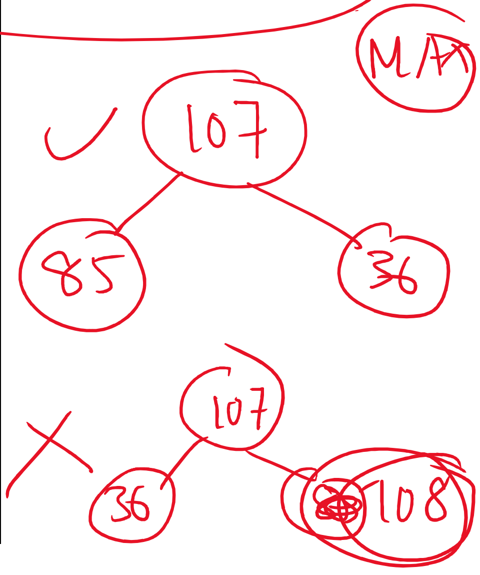
Wednesday, November 1, 2023 5:53 PM

MAx heap 107, 107, vool

FATHER & KiD

- A heap *T* is a complete Binary tree in which either *T* is empty or
- each item in ***Left(T)*** is <= ***Root*** item of ***T***
- each item in ***Right(T)*** is <= ***Root*** item of ***T***
- ***Left and Rights*** are heaps





# 11

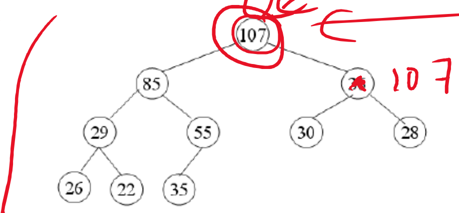
Wednesday, November 1, 2023 6:01 PM

Maxheap

MAx heap

FATHER → KiD
AGE AGE

- A heap *T* is a complete Binary tree in which either *T* is empty or
- each item in *Left(T)* is <= *Root* item of *T*
- each item in *Right(T)* is <= *Root* item of *T*
- *Left and Rights* are heaps

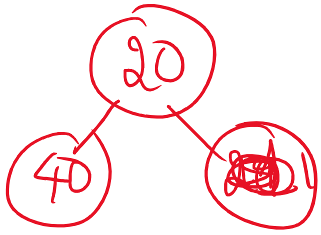


Root- Will have the max

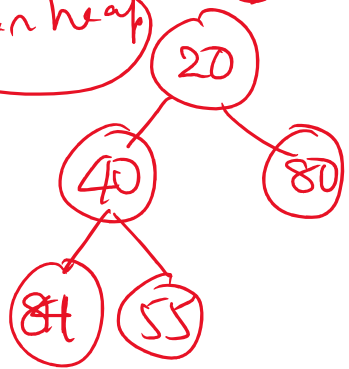
$\Theta(1)$

Min heap

Min heap



mi



Minheap

Min heap

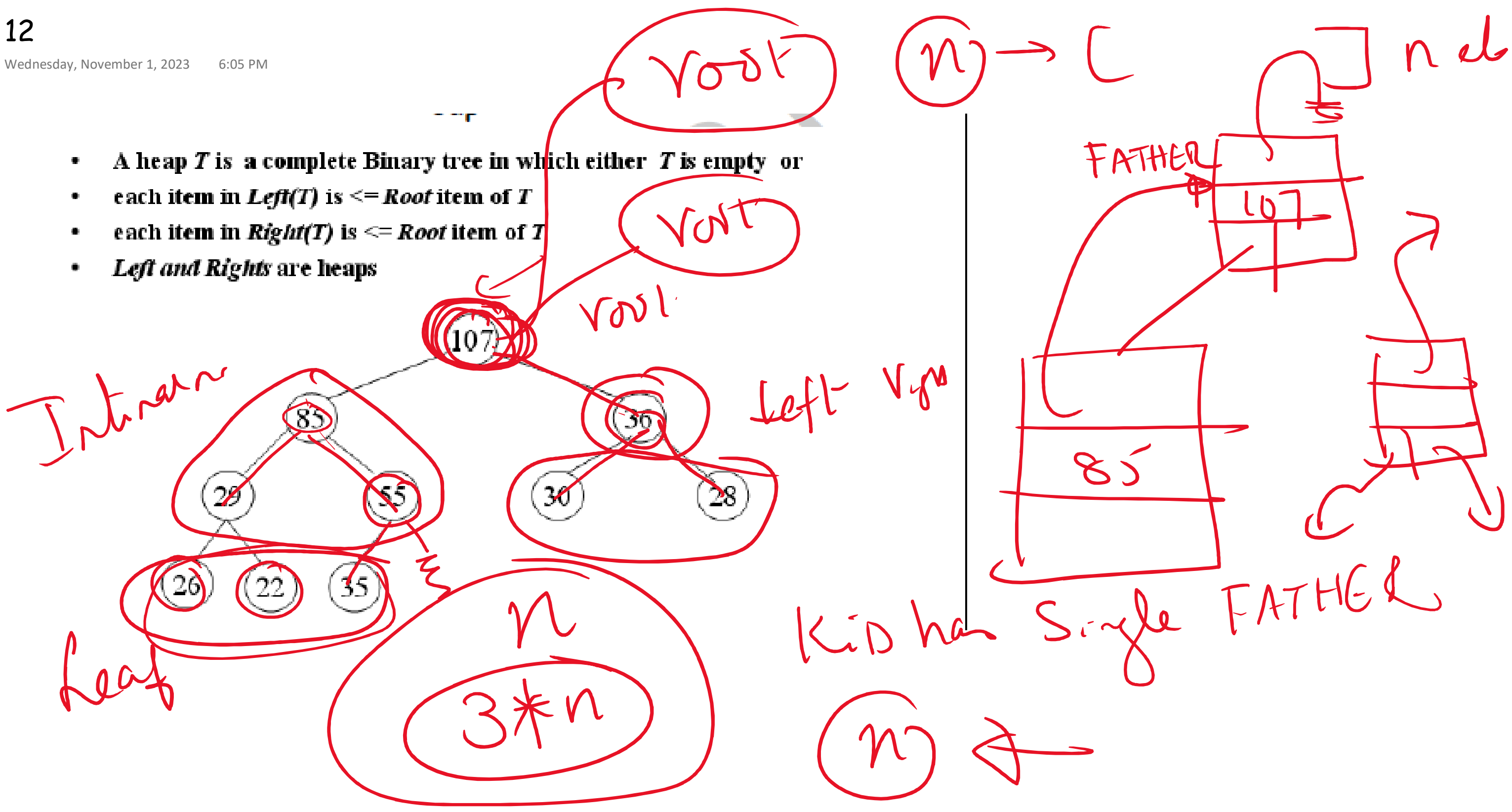
FATHER SALARY

KiD SALARY

12

Wednesday, November 1, 2023 6:05 PM

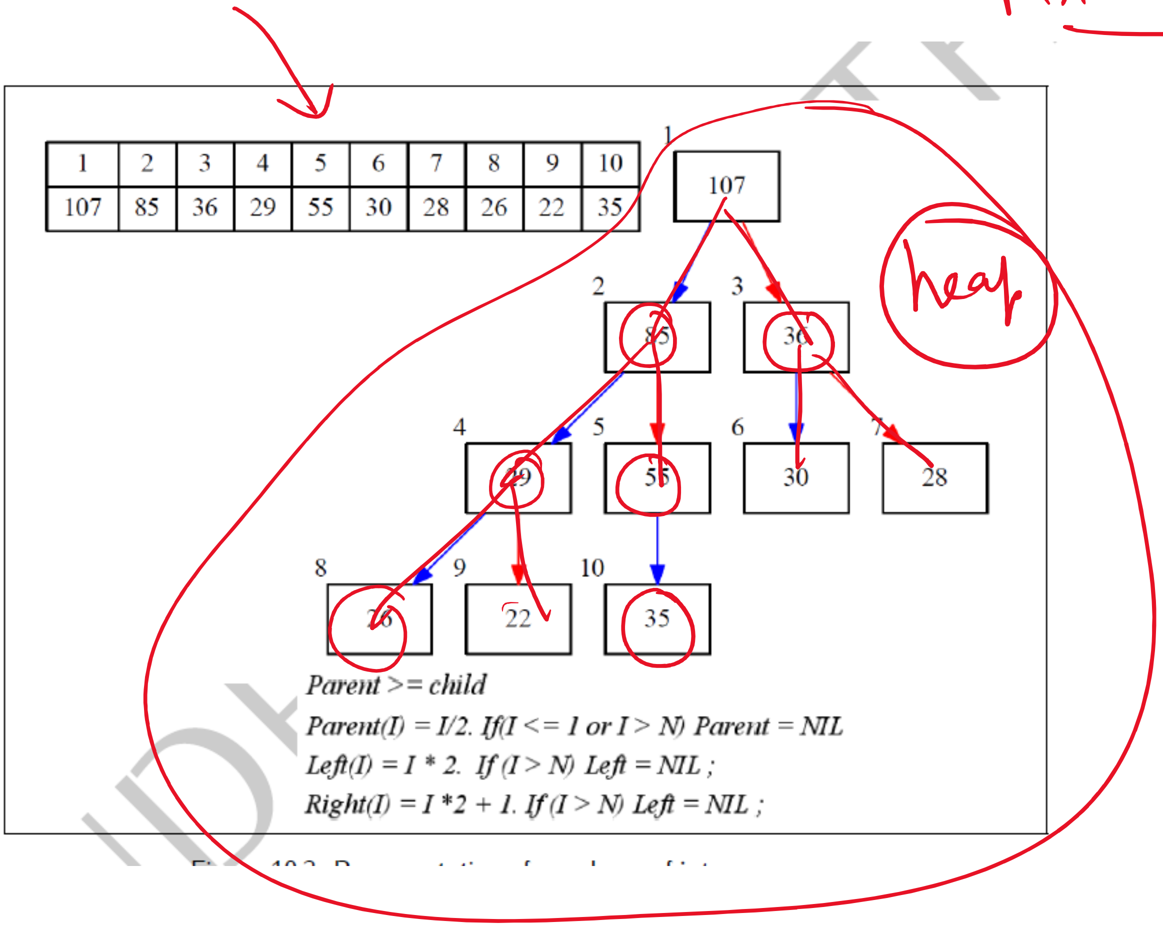
- A heap *T* is a complete Binary tree in which either *T* is empty or
- each item in *Left(T)* is <= *Root* item of *T*
- each item in *Right(T)* is <= *Root* item of *T*
- *Left and Rights* are heaps



# 13

Wednesday, November 1, 2023 6:08 PM

MAx heap



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 107 | 85 | 36 | 29 | 55 | 30 | 28 | 26 | 22 | 35 |

*Parent >= child*

*Parent(I) = I/2. If(I <= 1 or I > N) Parent = NIL*

*Left(I) = I * 2. If (I > N) Left = NIL ;*

*Right(I) = I *2 + 1. If (I > N) Left = NIL ;*

a.append(one) Θ(1)

a.ai

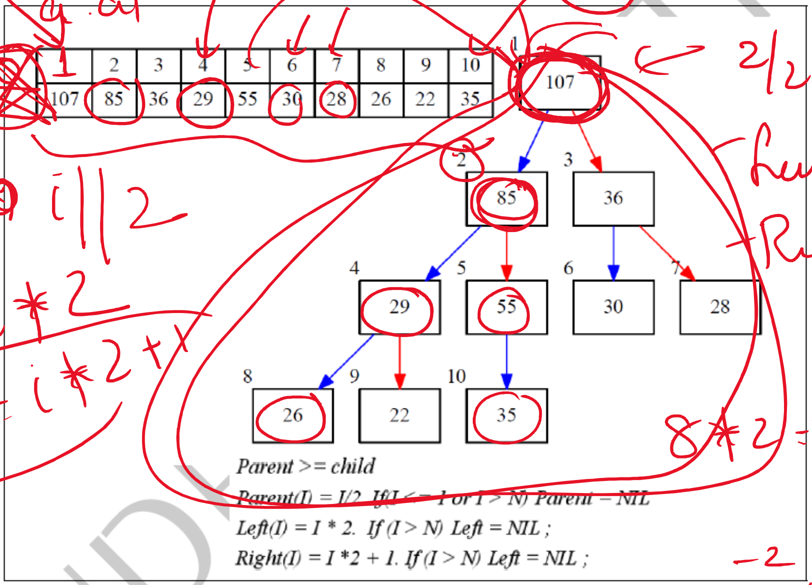
a

10

n = 1/2 = 0

Minheap or MAX

2/2 = 1



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 107 | 85 | 36 | 29 | 55 | 30 | 28 | 26 | 22 | 35 |

*Parent >= child*

*Parent(I) = I/2. If(I <= 1 or I > N) Parent = NIL*

*Left(I) = I * 2. If (I > N) Left = NIL ;*

*Right(I) = I *2 + 1. If (I > N) Left = NIL ;*

Figure 10.3 Representation of a heap of integers

FATHER = i//2

Leftkid = i*2

Right k = i*2+1

LeftKiD = 2*i

Right = 2*i+1

Θ(

8*2 = 16

5//2 = 2

-2 -1

Θ(1)

MAx-heap

MAx-heap

106

Θ(1)

a.append(106)

1 millin

20 steps

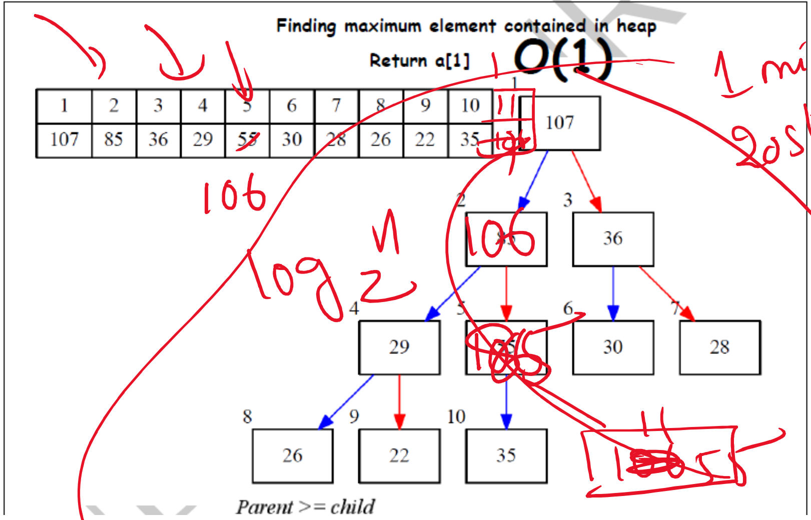
Heapify-up

5//2 =

11/2 = 5

Θ(1)

a[i]



Finding maximum element contained in heap

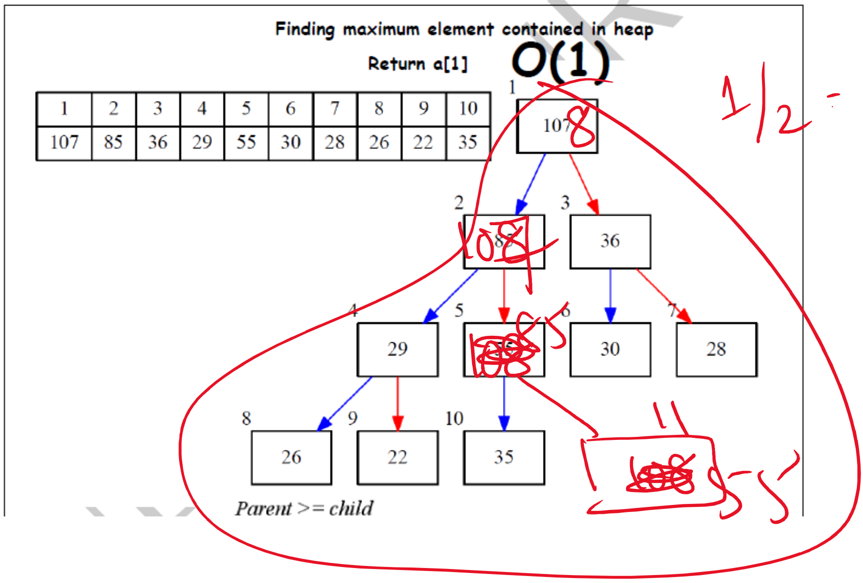
Return a[1] O(1)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|
| 107 | 85 | 36 | 29 | ~~55~~ | 30 | 28 | 26 | 22 | 35 | 106 |

106

$\log_2 n$

Parent >= child



Finding maximum element contained in heap

Return a[1] O(1)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 107 | 85 | 36 | 29 | 55 | 30 | 28 | 26 | 22 | 35 |

*Parent >= child*

108

1/2 = 0

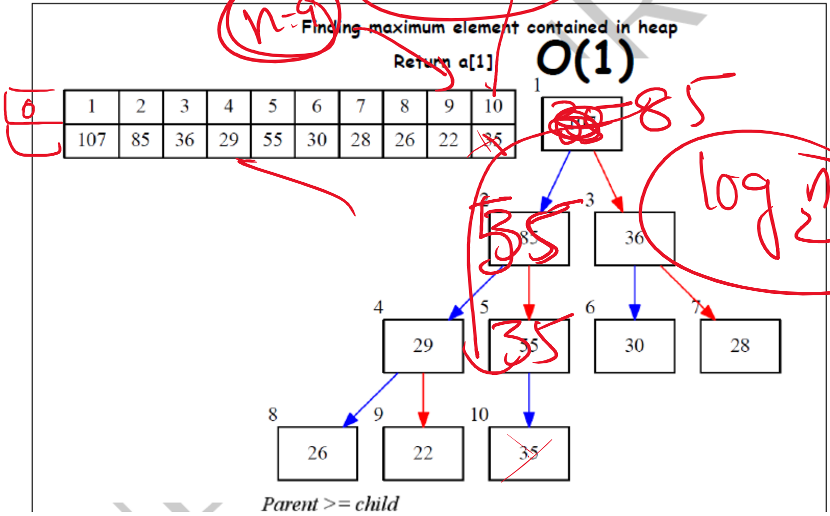
Insert = O(log n)

Heapify up

# 17

Wednesday, November 1, 2023 6:21 PM

Heapify-down



n=10

n=9

Finding maximum element contained in heap

Return a[1]

$O(1)$

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | 107 | 85 | 36 | 29 | 55 | 30 | 28 | 26 | 22 | ~~35~~ |

*Parent >= child*

$\log_2 n$

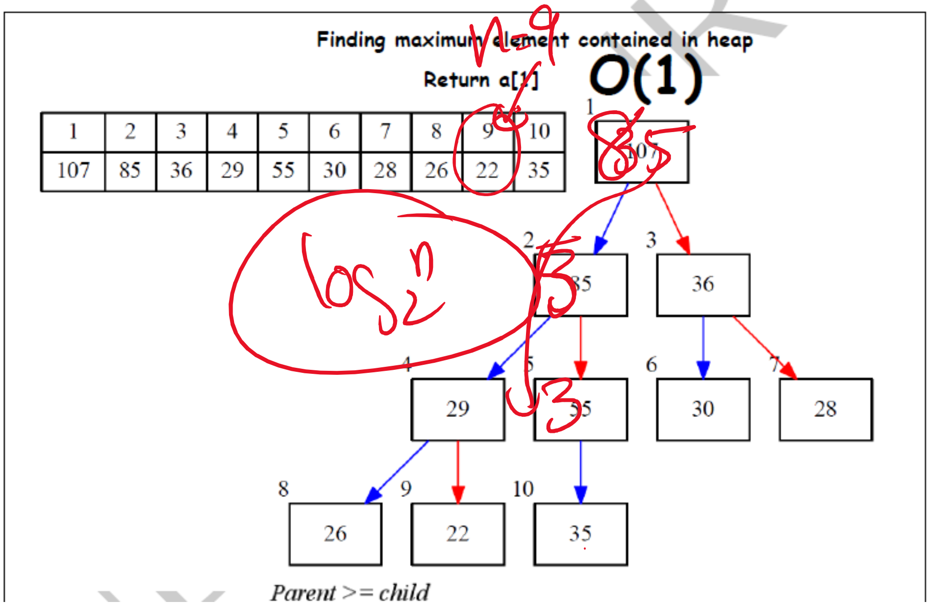
Find : $\Theta(1)$

Insert : $\Theta(\log n)$

Delete $O(\log n)$

Wednesday, November 1, 2023 6:25 PM

107



Finding maximum element contained in heap

n=9

Return a[1]

O(1)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 107 | 85 | 36 | 29 | 55 | 30 | 28 | 26 | 22 | 35 |

$\log_2^n$

*Parent >= child*

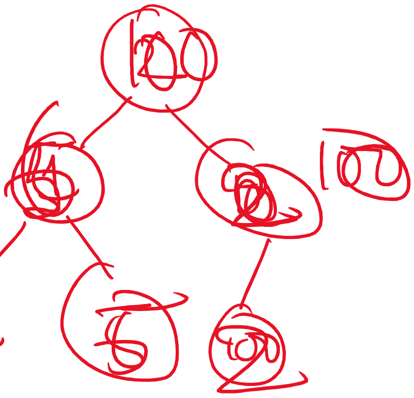
How to you BUILD heap

| 0 | 1 | 2 | 3 | 4 | |
|---|---|---|---|---|---|
| 2 | 1 | 10 | 5 | 6 | 100 |

a

```
h = heap()          n
for(e : a)
    h.insert(e)  <- log n
```

n.log n



build(int [ ] a )

n log n → n

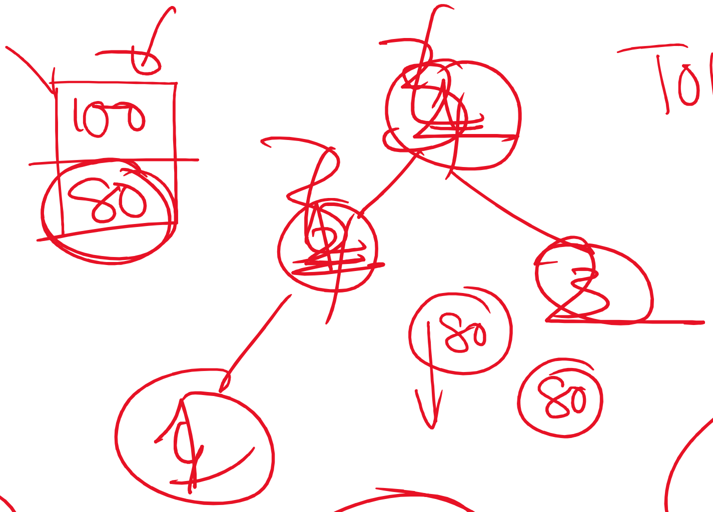
# 20

Wednesday, November 1, 2023 6:30 PM

Ascendin

```
int [] a = {1,2,3,4,5,6,7,8} ;
```

MAx heap



n.log n

nlogn

n

Top to Bottom Approach

6:55

Record

# 21

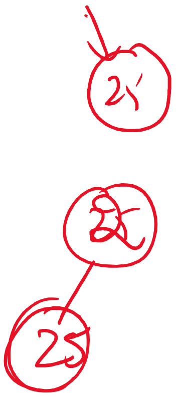
Wednesday, November 1, 2023 6:55 PM

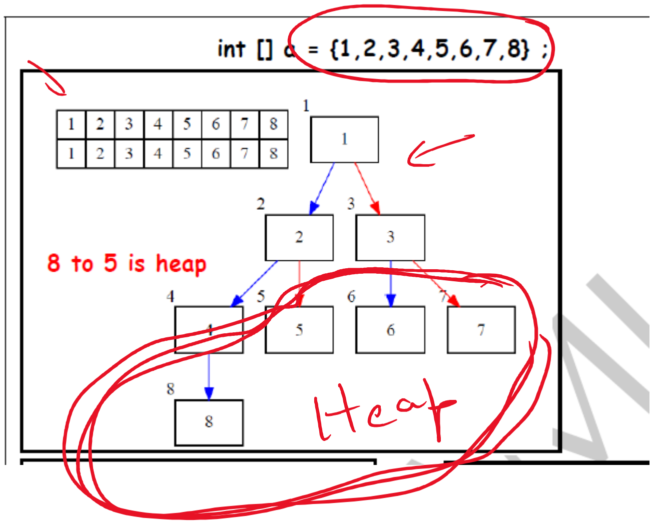
Tough coa

MAx heap

8

n = 1024




int [] a = {1,2,3,4,5,6,7,8} ;

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

8 to 5 is heap

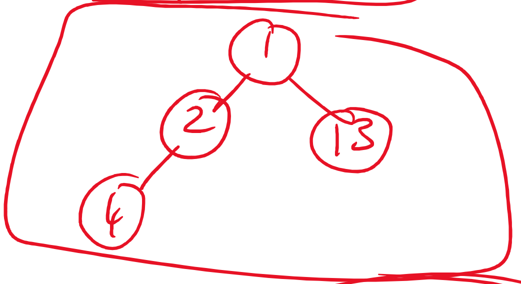
8

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|

1 2 13 4

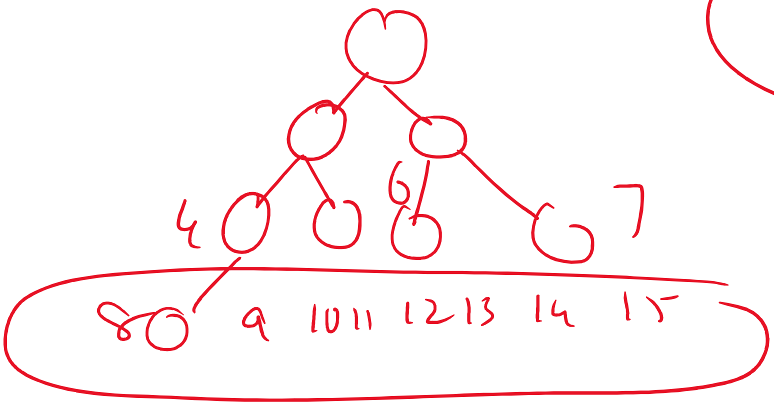
0

| 1 | 2 | 13 | 4 |
|---|---|---|---|

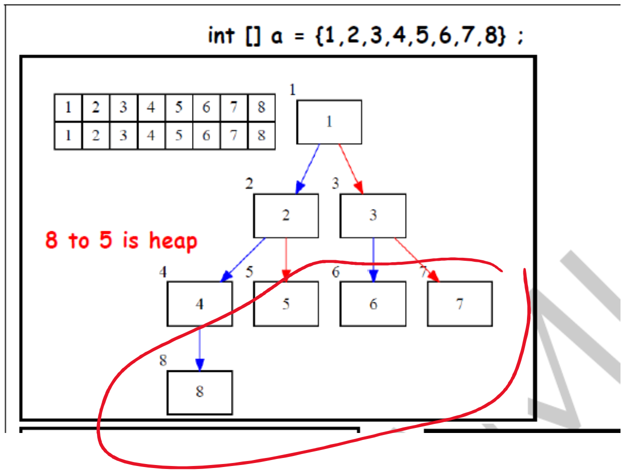


n = 15

$n \to n/2$



Wednesday, November 1, 2023 6:59 PM



int [] a = {1,2,3,4,5,6,7,8} ;

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

8 to 5 is heap

n → for(i=0; i < n; i++)

→ h.insert(a[i])

logn

nlogn

n/2

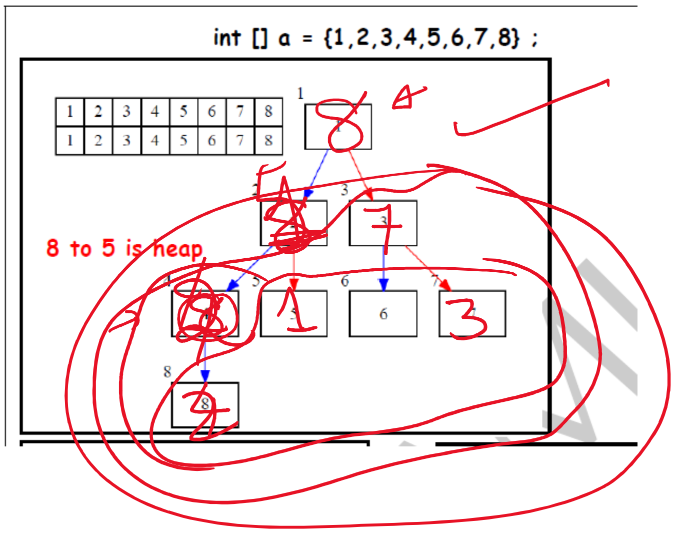
for(i=n/2; i >= 0;

i--){

3

# 23

Wednesday, November 1, 2023 7:00 PM



int [] a = {1,2,3,4,5,6,7,8} ;

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

8 to 5 is heap

MAx-heap

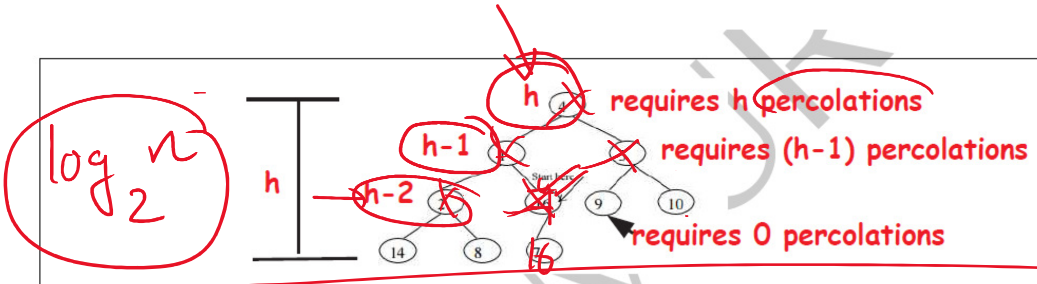
$n/2$ to 0

10

1 mill

500,000 to 0

Bottom to UP



S= h + 2(h-1) + 4(h-2) + ---+ 8(h-3) +16(h-4) + . . .

$S = h + 2(h-1) + 4(h-2) + 8(h-3) + \ldots 2^{(h-2)} \cdot 2 + 2^{(h-1)} \cdot 1$ (1)

log_2 n

Multiply by 2:

$2S = 2h + 4(h-1) + 8(h-2) + 16(h-3) + \ldots 2^{(h-1)} \cdot 2 + 2^{(h)} \cdot 1$ (2)

Subtract (1) from (2):

$S = h - 2h + 2 + 4 + 8 + \ldots 2^{h}$

$= -h - 1 + (2^{h+1} - 1)$

$= (2^{h+1} - 1) - (h + 1)$

Note: $1 + 2 + 4 + 8 + \ldots 2^{h} = (2^{h+1} - 1)$

Hence $S = (2^{h+1} - 1) - (h + 1)$ h = log n $2^{\log_2 n}$ = 2n = O(n)

Hence the complexity of building a heap with N nodes is linear **O(N)**

Figure 10.10: Analysis of building a heap of *n* elements **from bottom to top**

$2^{\log_2 n} = 2n$

Θ(n)

2n stp

Θ(n)

Library

heap(int a[ ]

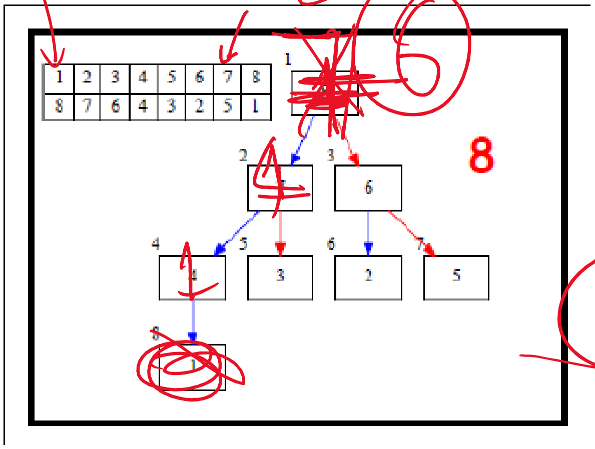
In place

B Θ(n) n log n → n

# 25

Wednesday, November 1, 2023 7:07 PM

a[1]



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 8 | 7 | 6 | 4 | 3 | 2 | 5 | 1 |

h = heap ①

XXX()

```
while( h.empty() == false)
{
    x = h.gettop() Θ(1)
    PRINT(x);
    delete top   log n
}
```

lo

log n

Heap Sort   n log n

Quick Sort = n log n

8 7

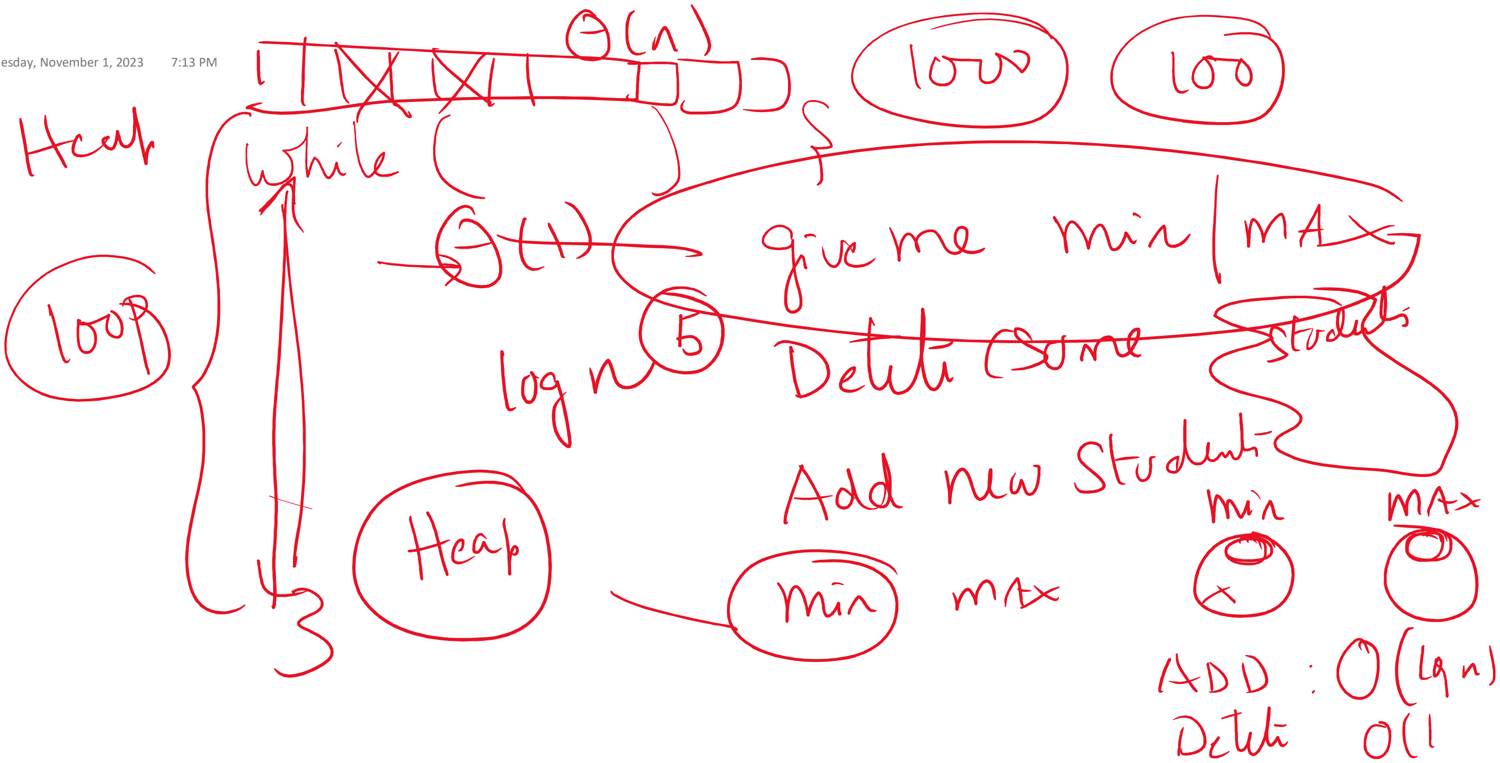
MAX heap → Descending order

Min heap → Ascending order

Heap Sort   n log n

# 26

Wednesday, November 1, 2023 7:13 PM



ADD : O(log n)

Delete O(1

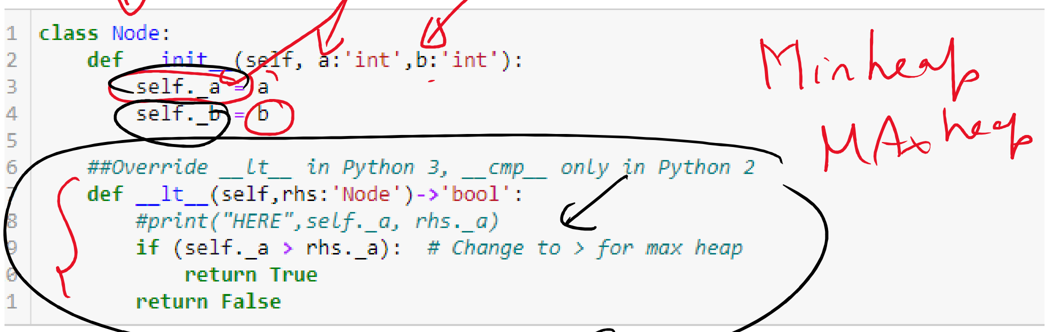
27

Wednesday, November 1, 2023 7:17 PM

heap PYTHON

import heapq

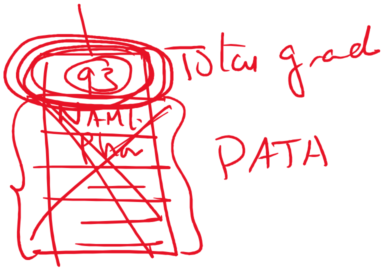
n = Node(5, 25)

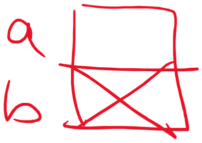


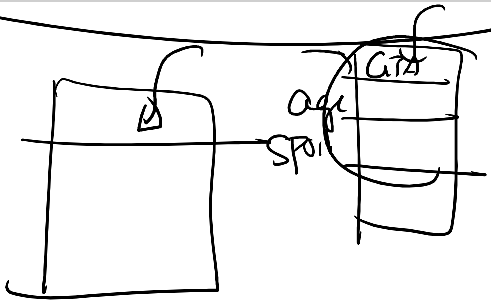
```
class Node:
    def __init__(self, a:'int',b:'int'):
        self._a = a
        self._b = b

    ##Override __lt__ in Python 3, __cmp__ only in Python 2
    def __lt__(self,rhs:'Node')->'bool':
        #print("HERE",self._a, rhs._a)
        if (self._a > rhs._a):  # Change to > for max heap
            return True
        return False
```

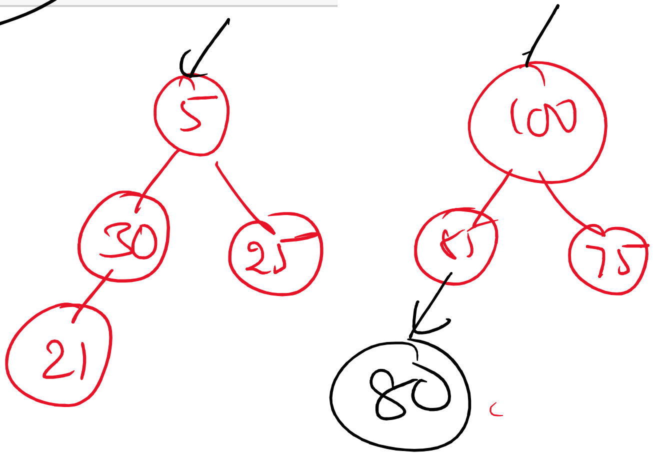
Min heap

MAx heap



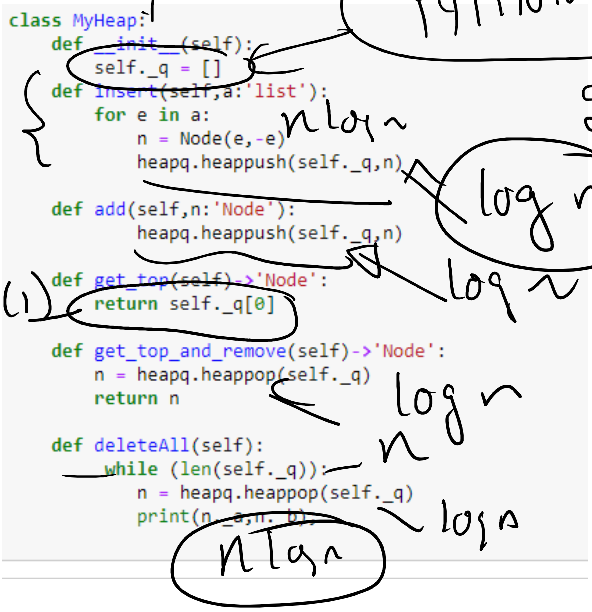






# 28

Wednesday, November 1, 2023 7:23 PM



```
class MyHeap:
    def __init__(self):
        self._q = []
    def insert(self,a:'list'):
        for e in a:
            n = Node(e,-e)
            heapq.heappush(self._q,n)

    def add(self,n:'Node'):
            heapq.heappush(self._q,n)

    def get_top(self)->'Node':
        return self._q[0]

    def get_top_and_remove(self)->'Node':
        n = heapq.heappop(self._q)
        return n

    def deleteAll(self):
        while (len(self._q)):
            n = heapq.heappop(self._q)
            print(n._a,n._b)
```

<

PYTHON LIST

n log n

log n

log n

Θ(1)

log n

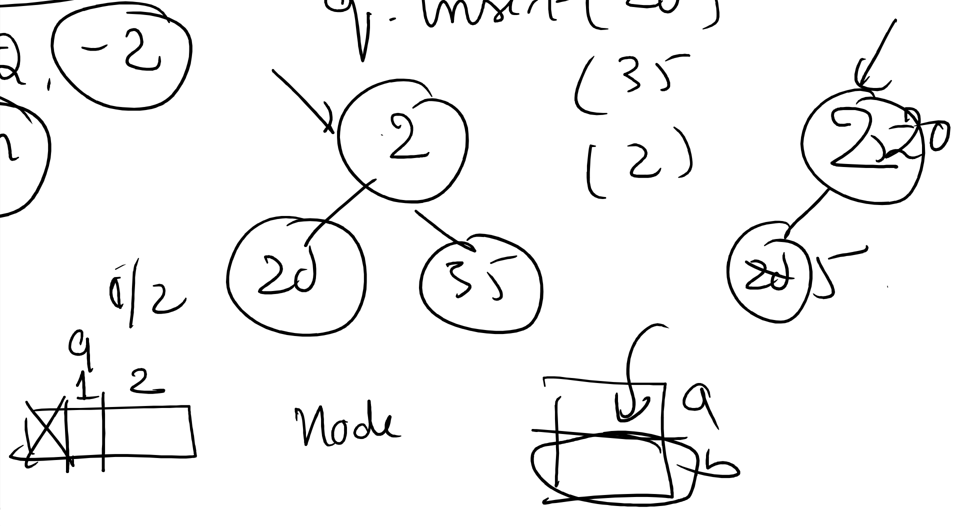
n

log n

n log n

q = MyHeap() LT

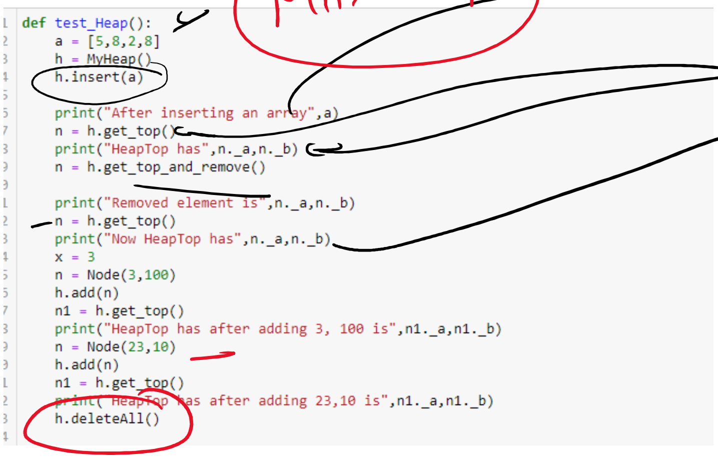
q.insert(20)



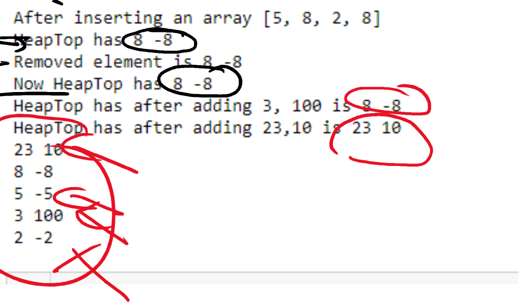
# 29

Wednesday, November 1, 2023 7:28 PM

MAxheap



```
def test_Heap():
    a = [5,8,2,8]
    h = MyHeap()
    h.insert(a)

    print("After inserting an array",a)
    n = h.get_top()
    print("HeapTop has",n._a,n._b)
    n = h.get_top_and_remove()

    print("Removed element is",n._a,n._b)
    n = h.get_top()
    print("Now HeapTop has",n._a,n._b)
    x = 3
    n = Node(3,100)
    h.add(n)
    n1 = h.get_top()
    print("HeapTop has after adding 3, 100 is",n1._a,n1._b)
    n = Node(23,10)
    h.add(n)
    n1 = h.get_top()
    print("HeapTop has after adding 23,10 is",n1._a,n1._b)
    h.deleteAll()
```

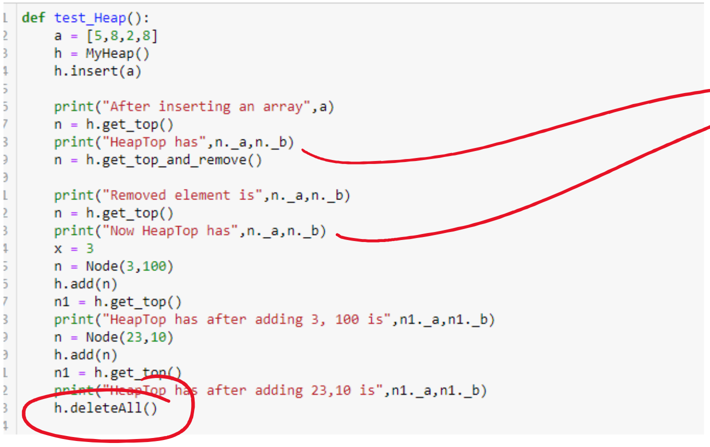


```
After inserting an array [5, 8, 2, 8]
HeapTop has 8 -8
Removed element is 8 -8
Now HeapTop has 8 -8
HeapTop has after adding 3, 100 is 8 -8
HeapTop has after adding 23,10 is 23 10
23 10
8 -8
5 -5
3 100
2 -2
```

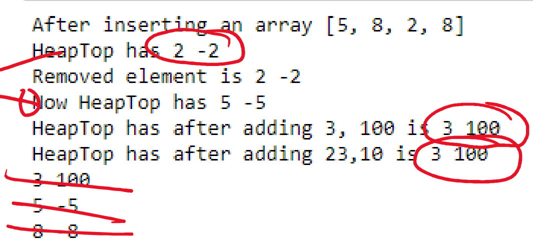
8
-8

# 30

Wednesday, November 1, 2023 7:33 PM



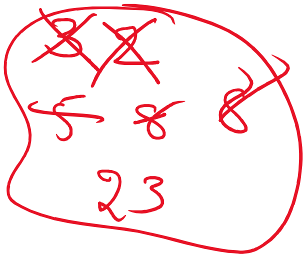
```python
def test_Heap():
    a = [5,8,2,8]
    h = MyHeap()
    h.insert(a)

    print("After inserting an array",a)
    n = h.get_top()
    print("HeapTop has",n._a,n._b)
    n = h.get_top_and_remove()

    print("Removed element is",n._a,n._b)
    n = h.get_top()
    print("Now HeapTop has",n._a,n._b)
    x = 3
    n = Node(3,100)
    h.add(n)
    n1 = h.get_top()
    print("HeapTop has after adding 3, 100 is",n1._a,n1._b)
    n = Node(23,10)
    h.add(n)
    n1 = h.get_top()
    print("HeapTop has after adding 23,10 is",n1._a,n1._b)
    h.deleteAll()
```



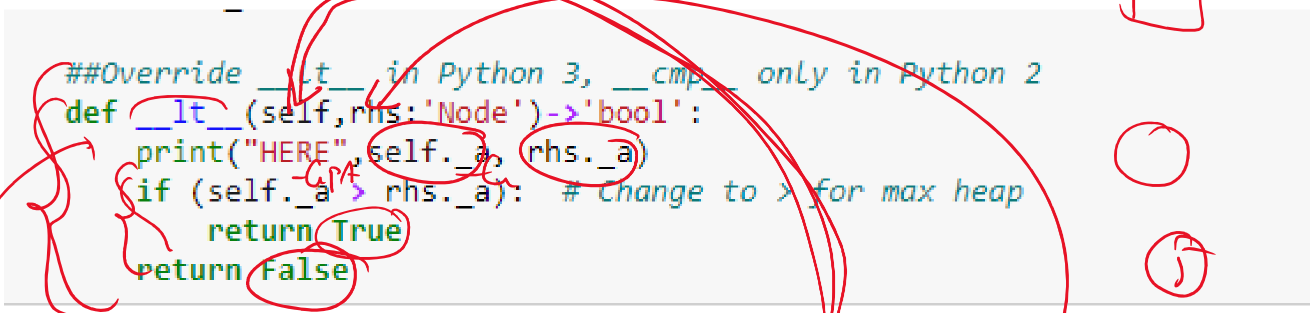
```
After inserting an array [5, 8, 2, 8]
HeapTop has 2 -2
Removed element is 2 -2
Now HeapTop has 5 -5
HeapTop has after adding 3, 100 is 3 100
HeapTop has after adding 23,10 is 3 100
~~3 100~~
~~5 -5~~
~~8 8~~
~~8 -8~~
23 10
```

23

log n



Wednesday, November 1, 2023 7:35 PM



```
##Override __lt__ in Python 3, __cmp__ only in Python 2
def __lt__(self,rhs:'Node')->'bool':
    print("HERE",self._a, rhs._a)
    if (self._a > rhs._a):   # Change to > for max heap
        return True
    return False
```

CPA

a < b

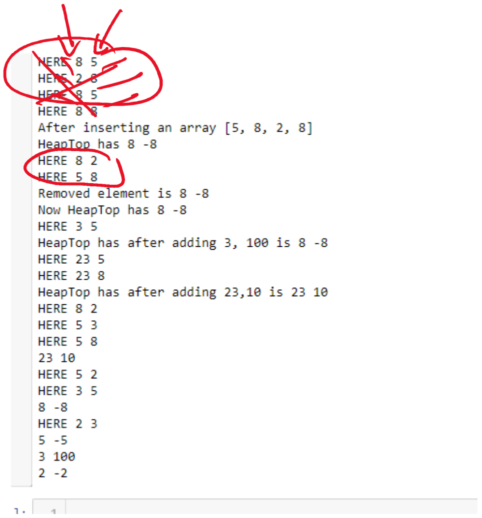
23 5

5 23

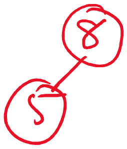
23 < 5

5<25

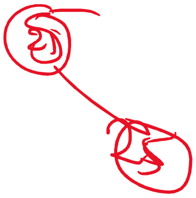
5>25



```
HERE 8 5
HERE 2 8
HERE 8 5
HERE 8 8
After inserting an array [5, 8, 2, 8]
HeapTop has 8 -8
HERE 8 2
HERE 5 8
Removed element is 8 -8
Now HeapTop has 8 -8
HERE 3 5
HeapTop has after adding 3, 100 is 8 -8
HERE 23 5
HERE 23 8
HeapTop has after adding 23,10 is 23 10
HERE 8 2
HERE 5 3
HERE 5 8
23 10
HERE 5 2
HERE 3 5
8 -8
HERE 2 3
5 -5
3 100
2 -2
```



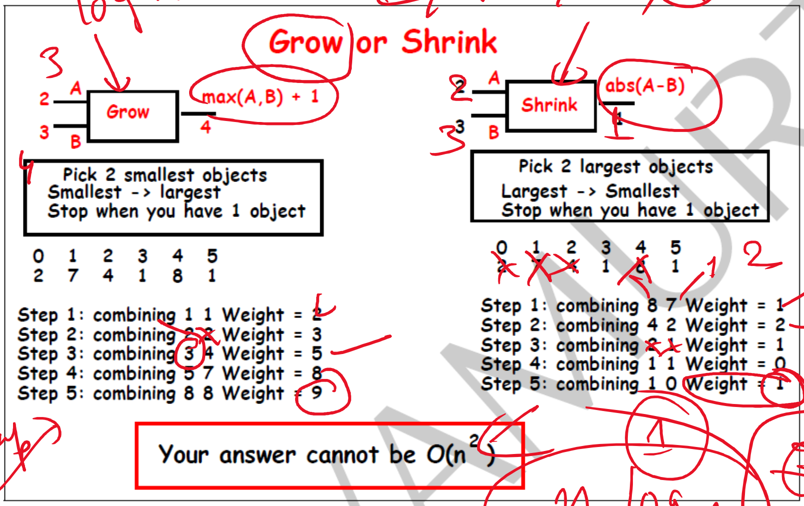
heap.f



O(1)

log n

MAX



## Grow or Shrink

A
B
2
3
Grow
max(A,B) + 1
4

Pick 2 smallest objects
Smallest -> largest
Stop when you have 1 object

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

Step 1: combining 1 1 Weight = 2
Step 2: combining 2 2 Weight = 3
Step 3: combining 3 4 Weight = 5
Step 4: combining 5 7 Weight = 8
Step 5: combining 8 8 Weight = 9
```

A
B
2
3
Shrink
abs(A-B)
1

Pick 2 largest objects
Largest -> Smallest
Stop when you have 1 object

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

Step 1: combining 8 7 Weight = 1
Step 2: combining 4 2 Weight = 2
Step 3: combining 2 1 Weight = 1
Step 4: combining 1 1 Weight = 0
Step 5: combining 1 0 Weight = 1
```

Your answer cannot be $O(n^2)$

2 → 1

Minheap

n log n

O(n)

0

4 - 2 = 1

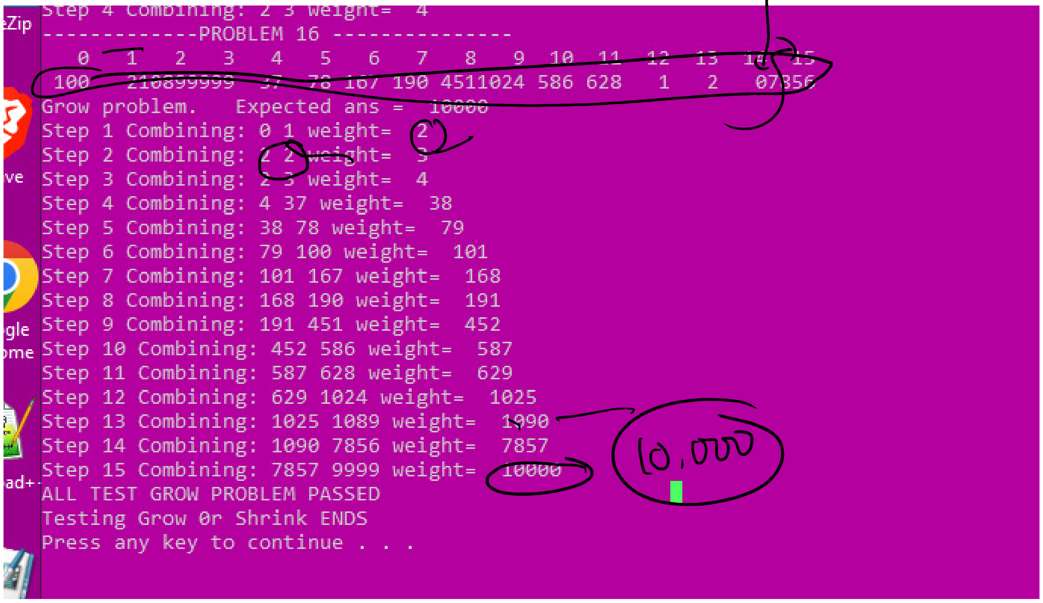
1 1 = 0

1 1 = 0

Figure 19.75: Grow or Shrink

# 33

Wednesday, November 1, 2023 7:55 PM



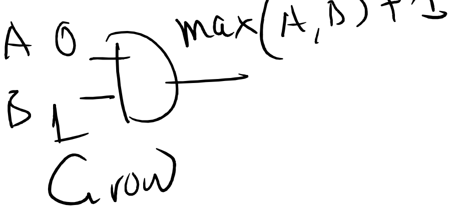
```
Step 4 Combining: 2 3 weight=  4
-------------PROBLEM 16 ---------------
   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
 100  210899999  37  78 167 190 4511024 586 628   1   2   07856
Grow problem.   Expected ans =  10000
Step 1 Combining: 0 1 weight=  2
Step 2 Combining: 2 2 weight=  3
Step 3 Combining: 2 3 weight=  4
Step 4 Combining: 4 37 weight=  38
Step 5 Combining: 38 78 weight=  79
Step 6 Combining: 79 100 weight=  101
Step 7 Combining: 101 167 weight=  168
Step 8 Combining: 168 190 weight=  191
Step 9 Combining: 191 451 weight=  452
Step 10 Combining: 452 586 weight=  587
Step 11 Combining: 587 628 weight=  629
Step 12 Combining: 629 1024 weight=  1025
Step 13 Combining: 1025 1089 weight=  1090
Step 14 Combining: 1090 7856 weight=  7857
Step 15 Combining: 7857 9999 weight=  10000
ALL TEST GROW PROBLEM PASSED
Testing Grow Or Shrink ENDS
Press any key to continue . . .
```

10,000

0 1

0 7856

0,


A 0
B 1
max(A,B)+1
Grow