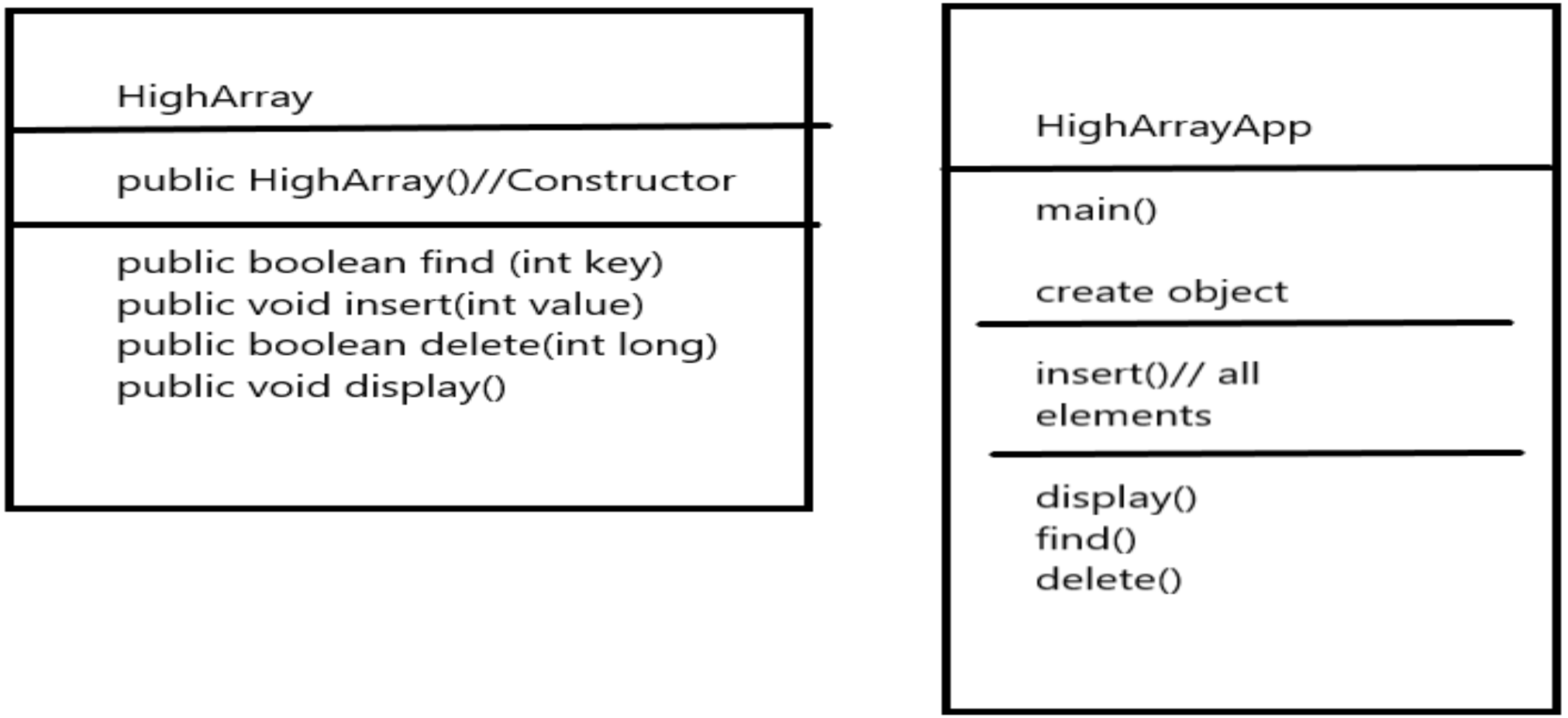
Algorithms & Data Structure

Kiran Waghmare

Program 2


HighArray
public HighArray()//Constructor
public boolean find (int key)
public void insert(int value)
public boolean delete(int long)
public void display()
HighArrayApp
main()
create object
insert()// all elements
display()
find()
delete()

Performance of Algorithms

Algorithm: set of rules required to perform calculations.

-complexity

- -Time complexity
- -Space complexity

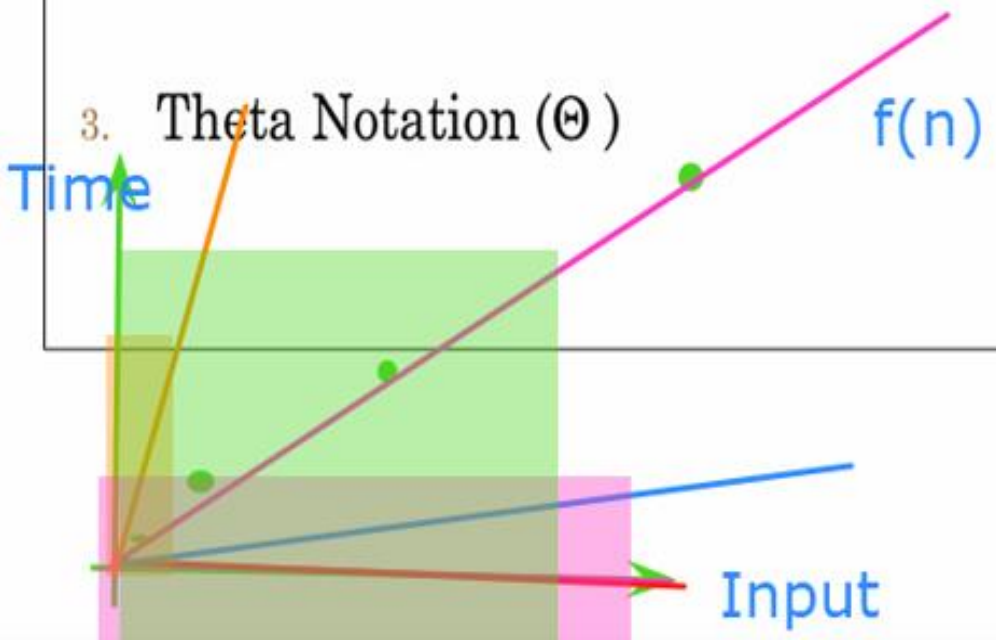
WHY IMPORTANT ??

- Give a simple characterization of an algorithm's efficiency.
- Allow comparison of performances of various algorithms

1-1sec
10-10sec
1000-1000sec
1lac-1lacsec

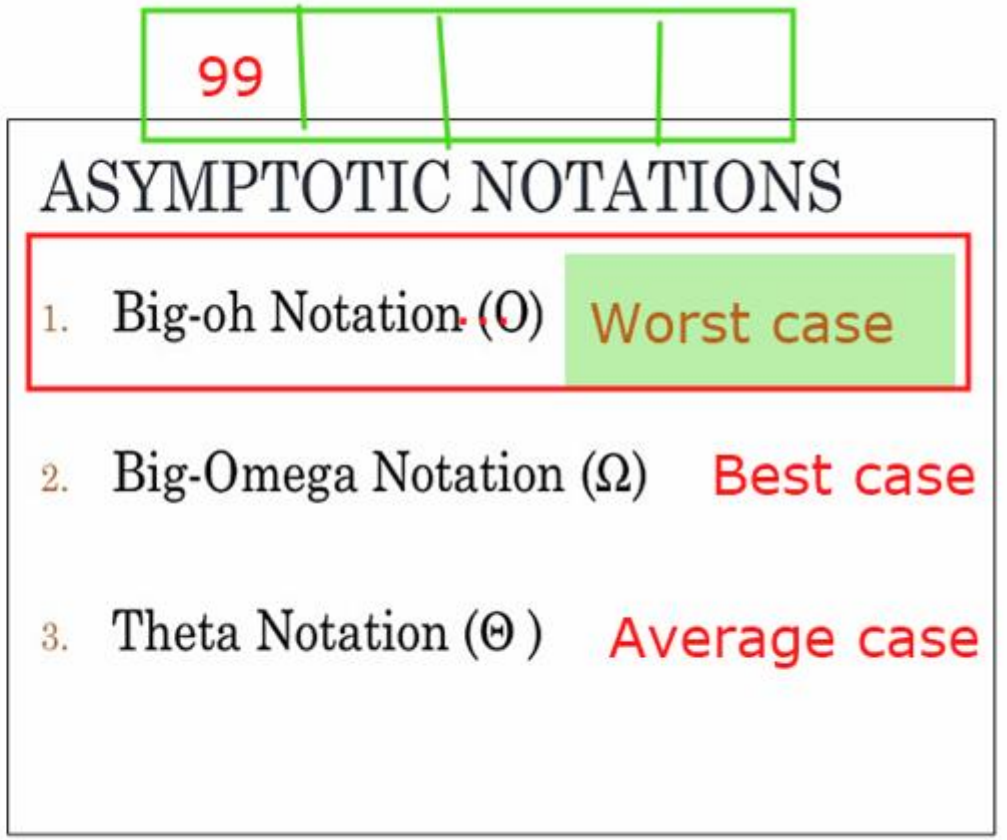
ASYMPTOTIC NOTATIONS

1. Big-oh Notation (O)
2. Big-Omega Notation (Ω)
3. Theta Notation (Θ)



WHY IMPORTANT ??

- Give a simple characterization of an algorithm's efficiency.
- Allow comparison of performances of various algorithms



BIG-OH NOTATION (O)

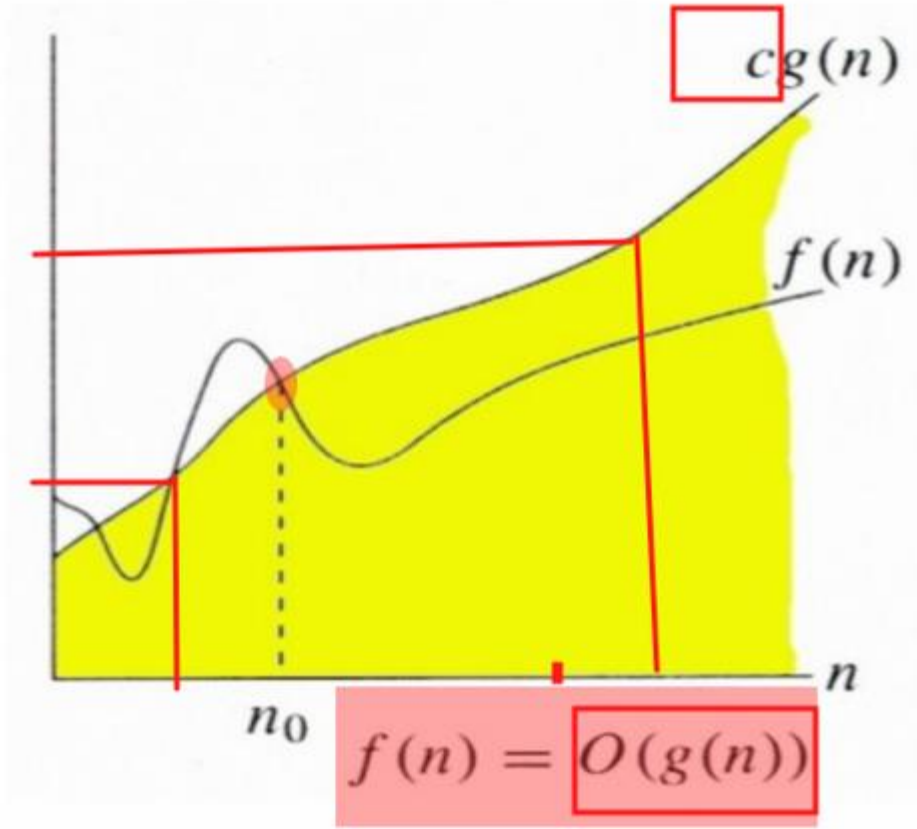
Who can see what you share here? Recording On

T(n)=T(n/2)+n

- Gives the upper bound of algorithm's running time.
- Let f: N-> R be a function. Then O(f) is the set of functions

  O(f) = { g: N-> R | there exists a constant c and a natural number n_0 such that

  $|g(n)| <= c|f(n)|$ for all $n >= n_0$ }



BIG-OMEGA NOTATION (Ω)

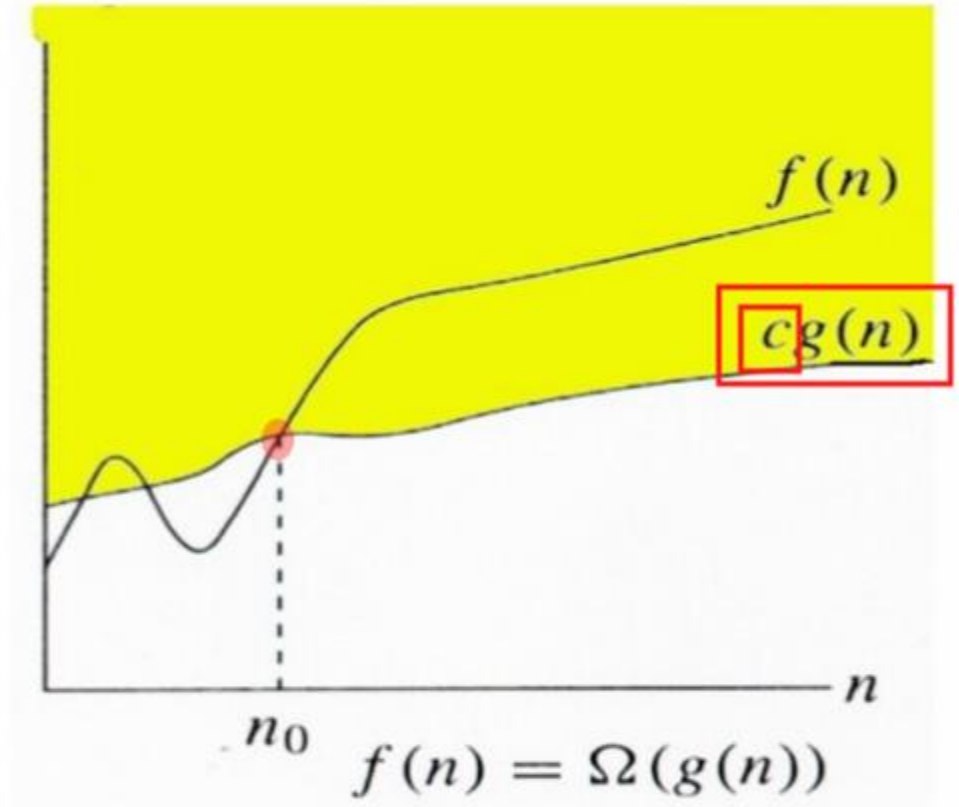
Who can see what you share here? Recording On

- Gives the lower bound of algorithm's running time.
- Let f, g: N-> R be functions from the set of natural numbers to the set of real numbers.

  We write $g \in \Omega(f)$ if and only if there exists some real number n_0 and a positive real constant c such that

  $g(n) \mid >= c \mid f(n) \mid$

for all n in N satisfying $n >= n_0$.



5

BIG-OMEGA NOTATION (Ω)

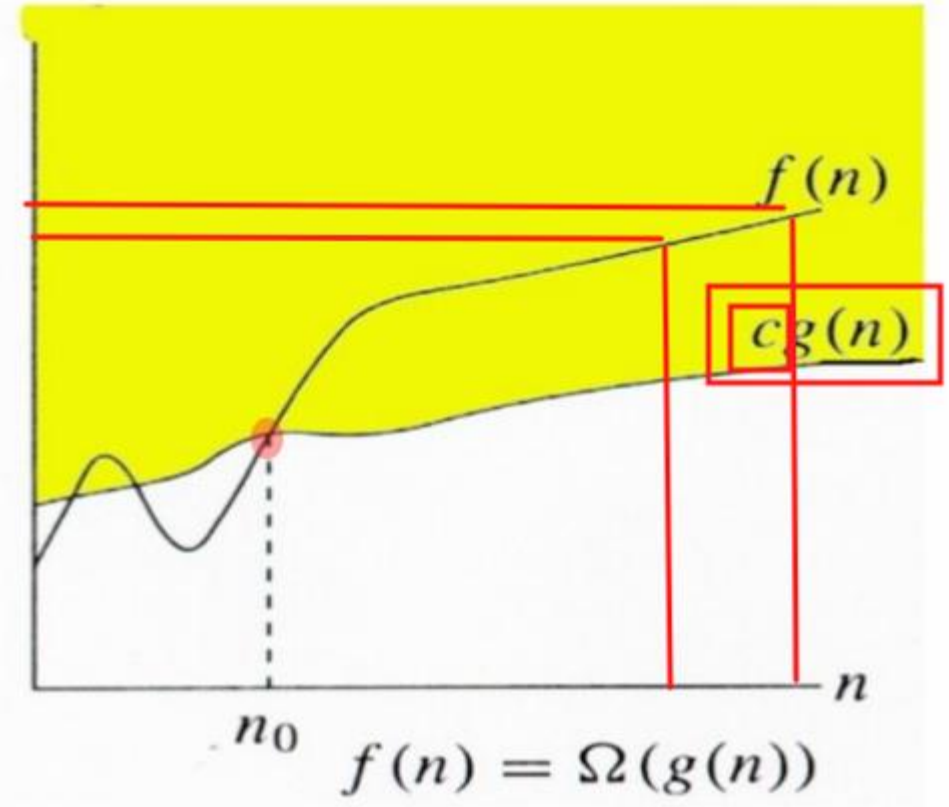
Who can see what you share here? Recording On

- Gives the lower bound of algorithm's running time.
- Let f, g: N-> R be functions from the set of natural numbers to the set of real numbers.

  We write $g \in \Omega(f)$ if and only if there exists some real number n_0 and a positive real constant c such that

  $g(n) \mid >= c \mid f(n) \mid$

for all n in N satisfying $n >= n_0$.



5

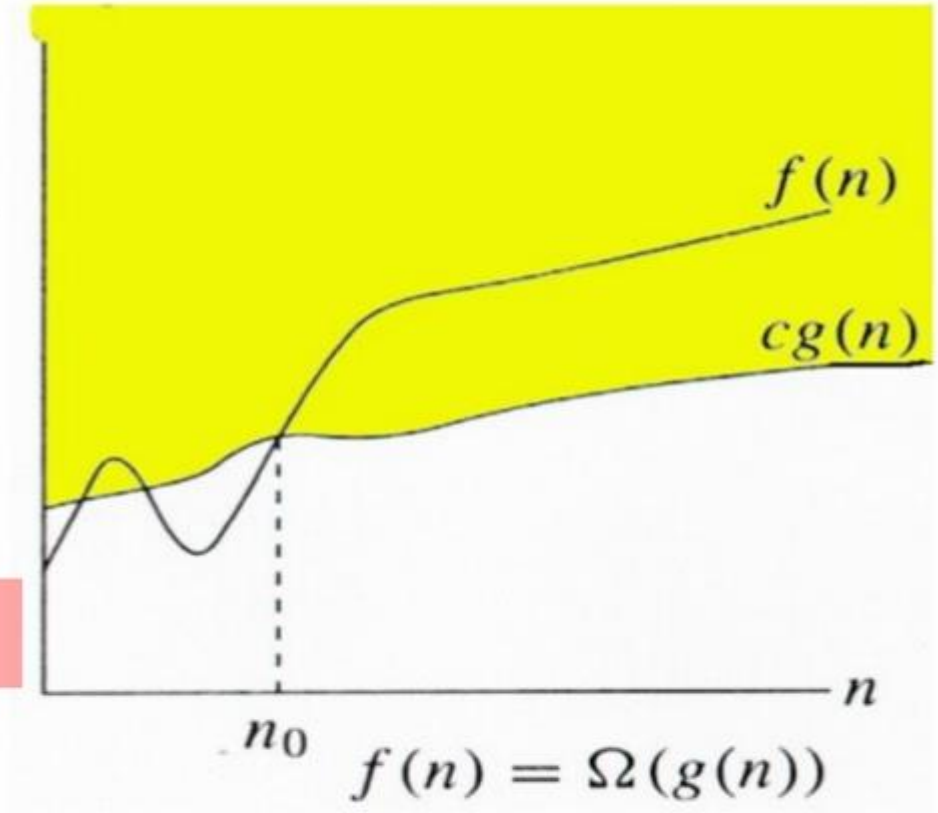
THETA NOTATION (Θ)

Who can see what you share here? Recording On

- If f and g are functions from S to the real numbers, then we write $g \in \Theta(f)$ if and only if there exists some real number n_0 and positive real constants C and C' such that

$$C\,|f(n)| <= |g(n)| <= C'\,|f(n)|$$

for all n in S satisfying $n >= n_0$.

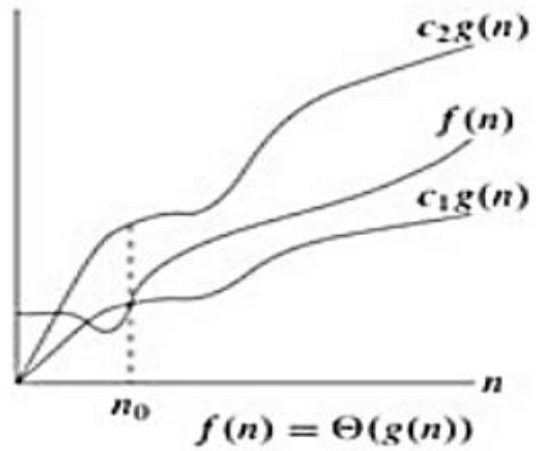


Thus, $\Theta(f) = O(f) \cap \Omega(f)$

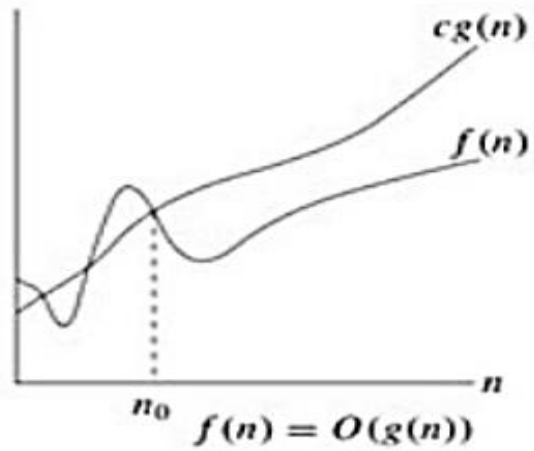
6

INTUITION ABOUT THE NOTATIONS

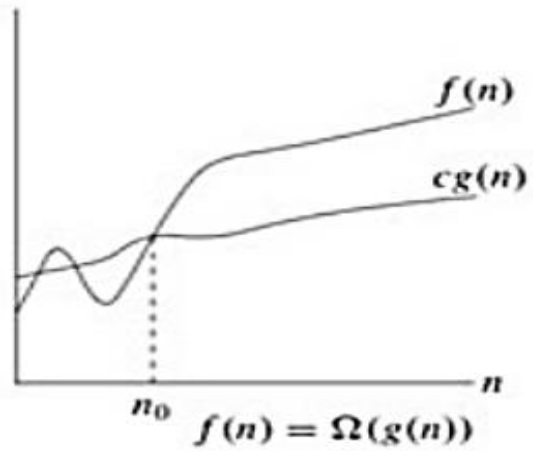
notation	intuition
O (Big-Oh)	$f(n) \leq g(n)$
Ω (Big-Omega)	$f(n) \geq g(n)$
Θ (Theta)	$f(n) = g(n)$



$f(n) = \Theta(g(n))$



$f(n) = O(g(n))$



$f(n) = \Omega(g(n))$

Analogy to Arithmetic Operators

$$f(n) = O(g(n)) \approx a \leq b$$

$$f(n) = \Omega(g(n)) \approx a \geq b$$

$$f(n) = \Theta(g(n)) \approx a = b$$

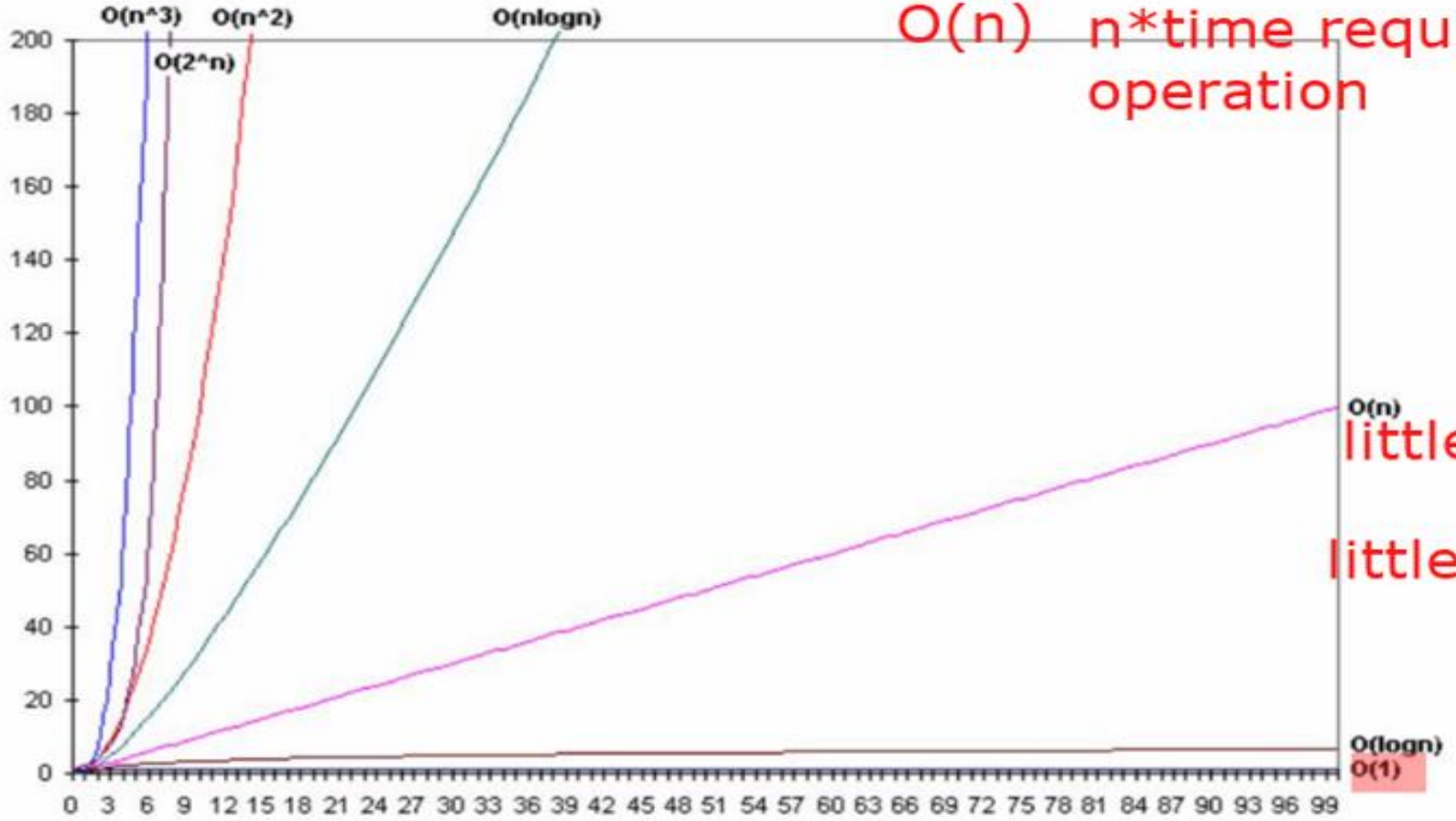
$$f(n) = o(g(n)) \approx a < b$$ little-Oh

$$f(n) = \omega(g(n)) \approx a > b$$ little-Omega

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo
Who can see what you share here? Recording On

Is there a "real" difference?

- Growth of functions



O(1) const time for each operation

O(n) n*time required for that operation

little-Oh

little-Omega

Best, Worst, Average Cases

Not all inputs of a given size take the same time to run.

Sequential search for K in an array of n integers:

- Begin at first element in array and look at each element in turn until K is found

Best case:

Worst case:

Average case:

Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- O Notation
- Ω Notation
- θ Notation

Worst Case

int a1[10];

Access O(1)

Insert O(1)--->O(n):n elements

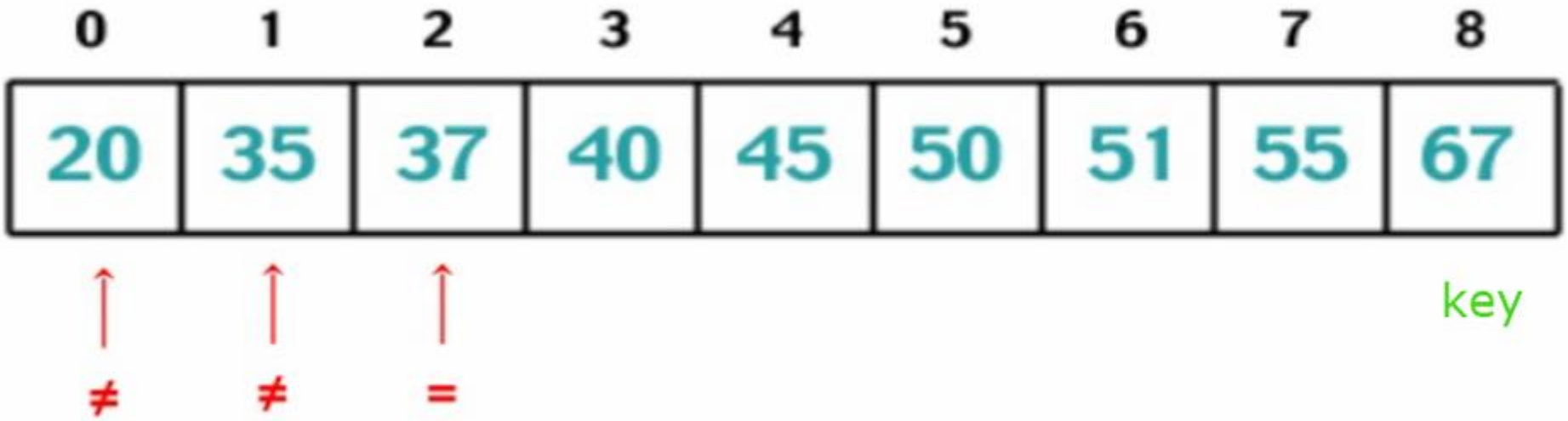
Search O(1)--->O(n):n elements

Delete O(1)--->O(n):n elements

Linear Search

- Find 37?

Key=37



Return 2

10 20 30 40 50 60 70 1 5 8

Binary Search

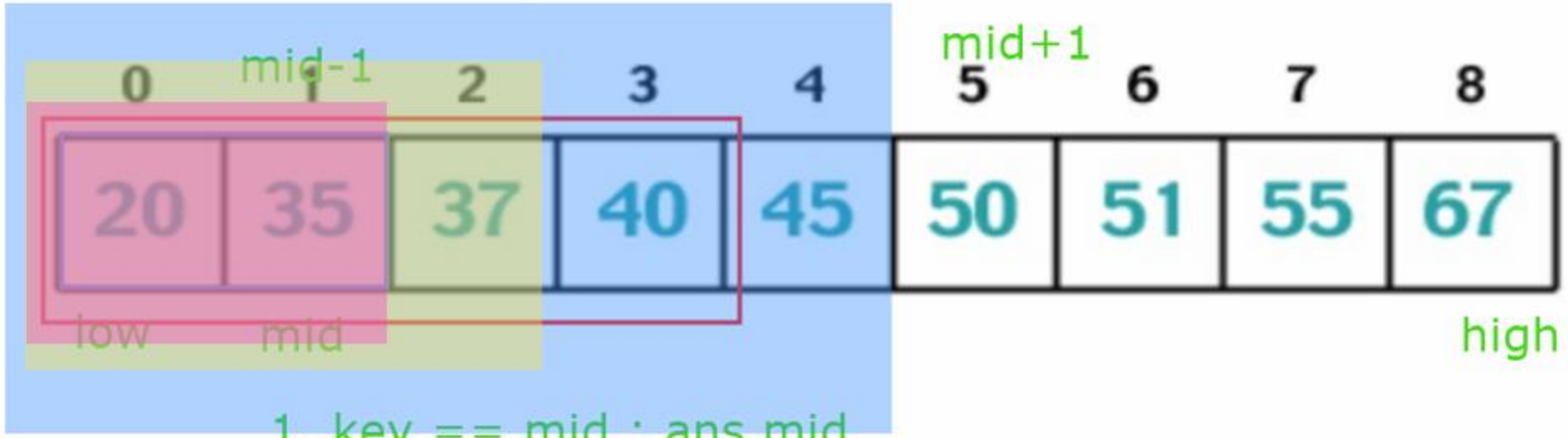
mid=(0+8)/2=4

Key=37

Key=35

O(log n)

- Find 37?
  1. Sort Array.

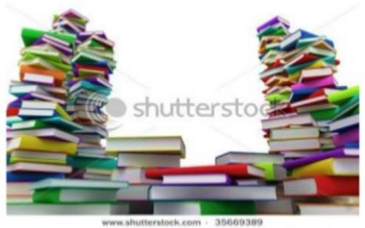


1. key == mid ; ans mid
2. key < mid ; Left
3. key >mid ; Right

```
class BS
{
    static int bsearch(int a1[], int key)
    {
        int l=0, h=a1.length-1;
        while(l <= h)
        {
            int m=l+(h-l)/2;

            if(a1[m] == key)
                return m;

            if(a1[m] < key)
                l=m+1;
            else
                h=m-1;
        }
        return -1;

    }

    public static void main(String args[])
```

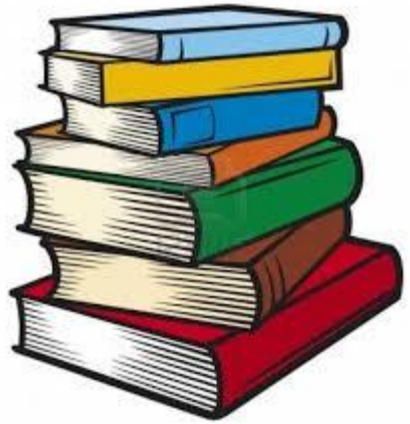
h=(7-4)/2
=1.5

Examples of stack



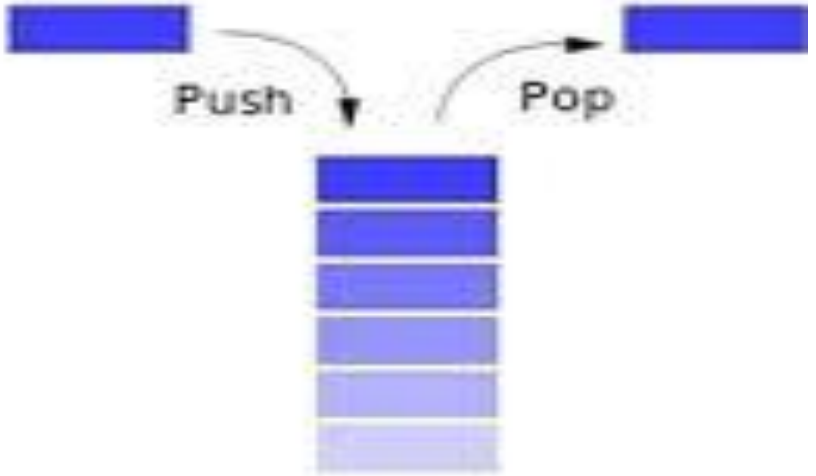
Stacks

Kiran Waghmare



Stack of books

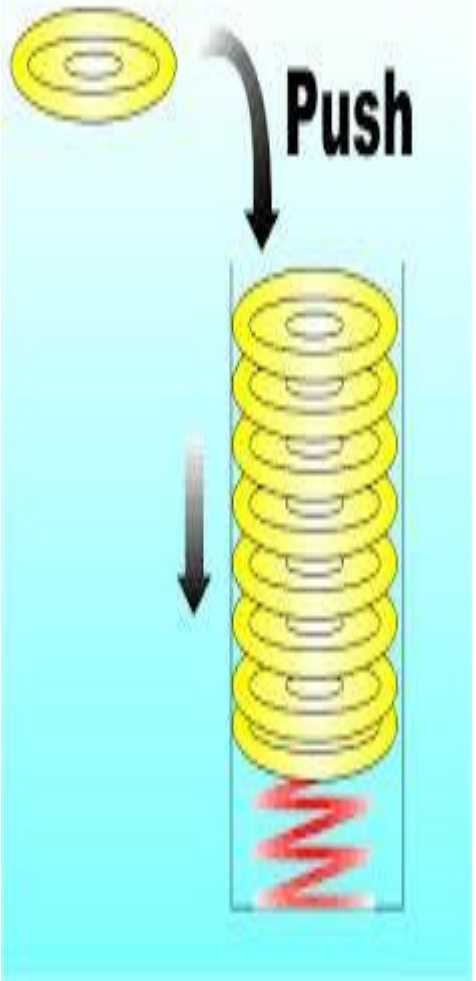
Stack of Coins



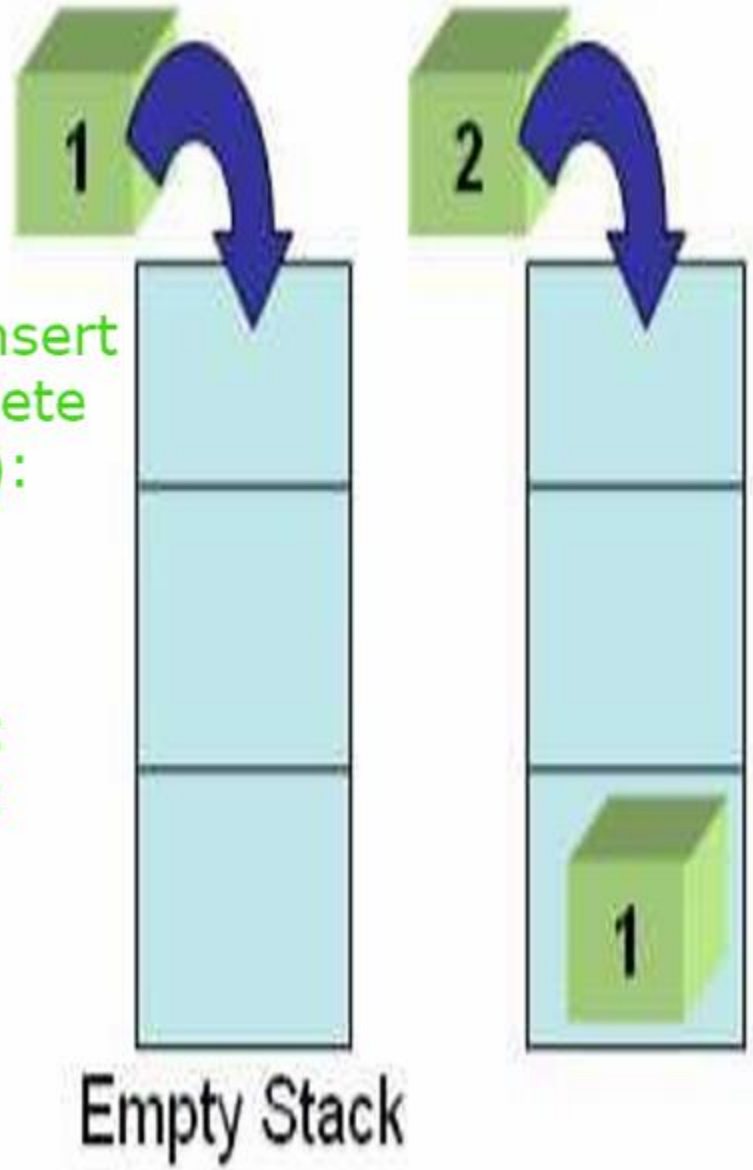
Memory stack

Stack

- Stack is an ordered list of similar data type.
- Stack is a LIFO structure. (Last in First out).
- push() function is used to insert new elements into the Stack and pop() is used to delete an element from the stack. Both insertion and deletion are allowed at only one end of Stack called Top.
- Stack is said to be in Overflow state when it is completely full and is said to be in Underflow state if it is completely empty.



PUSH : Insert
POP : Delete
isEmpty():
isFull():
peek():
count():
change():
display():



Who can see what you share here? Recording On

10 20

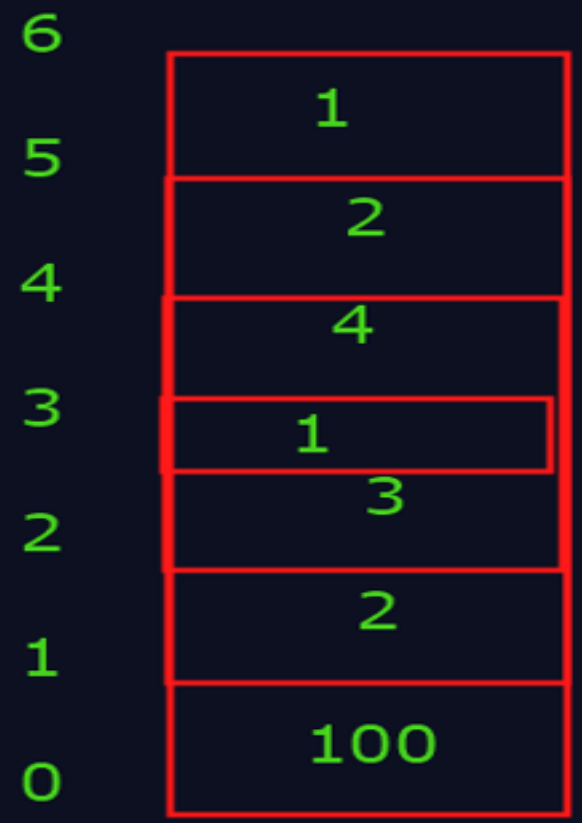
20 TOP
0 10 TOS

Talking: Kiran Waghmare

2
1

the
first object to
come out.

Array Representation of Stack




```
int S[7];
TOS=30
TOS=-1;
push(10);
push(20);
Push(30);
Push(40);
Pop();
pop(20);
pop();
pop();
```

Underflow:stack is empty & we are trying to delete element

Overflow: Stack is full & we are trying to insert an element