All Objects Do work

Car accelerate , fish swim , people walk .Similarly in software , we create objects that have particular function and behaviour and we ask those software objects to carry out those functions.As a result ,the application does as expected. Software objects do all the work in an application.

There is a default constructor in a class , which we need not define as we have done here.

Human h = **new** Human("Ananth",30,158,"blue"); - this is creating an object

Car c1 = new car(); //1st object

Car c2 = new car(); //2nd object

Classes contains methods which have all of the behaviour defined. Classes are blueprint or a design. We use that blueprint to create actual objects.

Contructors are used to construct objects . Objects Exist During Application Runtime.

It will create a variable of type animal1 , it allocates space for that variable.

Somewhere else in memory , it creates a object **new** Animal(12,"M",23);

Animal animal1=**new** Animal(12,"M",23);

animal1.eat();

This animal1(variable) is going to the point to the memory address where the object(new Animal) is located.

When you create a class , without having any body in it and run it , it will throw error – it will complain.

In java – instructions need to be wrapped in a named block. Wrapped block is a method. Method do all instructions in java

If you write a for loop in a empty class with no methods, it is an inaccurate java code. You can declare some variables but not more than that. It looks for a method and that is called main method.

There are 2 memory locations that become reserved for your java applications to run successfully. These are resources of your computer RAM .

* One is stack.
* The other is heap

Stack:

When you hit the run button , the first method invocation is main method .A frame is going to be created inside stack.The main method there inside frame.

All the variables inside main method are local varaibles

Int age;

Local to the main method

The interpreter is going to execute line by line , when it reaches the line int age , in the main frame , a specific reservation (name age) is going to be made for the variable.A reserved place to maintain the data for the age variable

Age = 12;

doWork()- there could be variable in this function too. It could be string

Now we have doWork() method

Inside Do work method , there could be DoMore method too.

Once the doMore() method is done executing , the doMore Method frame is removed from the stack.

Similarly , once the doWork() method is done executing , the doWork Method frame is removed from the stack.

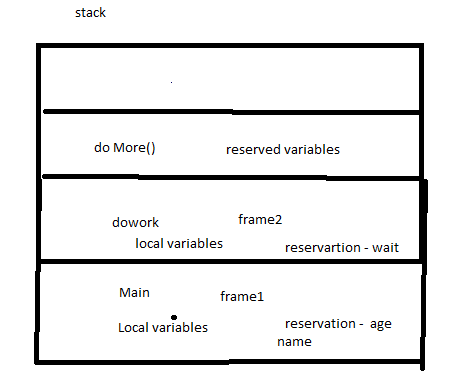
No we just have main method in the stack.

Stack maintains method invocation . Methods are going to be piled up on the stack.

Once the execution is complete , it is going to be removed from the stack.

When the main method is done running , it is going to be removed as well(from the stack- the main method)

All the memory is cleared.



Local Reference variable

Car MyCar;

Special reference memory is going to be created for local reference variable

My car is going to point to the actual object. We just have declared it . We have not assigned it.

MyCar = new Car();

Car is not a primitive data type . It is a complex data type .It is a class.

Reference Variable Mycar() doesn’t contain the object

My car refers to the actual object new Car();that is going to reside in memory . New Car is going to reside in heap.

MyCar variable is going to point to car object which IS IN HEAP.Where car is going to reside in heap , that memory address is assigned to My car variable.

Whereas age =12 , contains the actual data

If you insert another line Mycar = **new** Car(); , another car object is going to be created in the heap and My car is going to contain the new car object memory address in the heap.

So , we have lost connection with old car object. This is where garbage collection comes in.The old object is a rouge object that is a candidate to be deleted.

Garbage collection is a process that runs in the heap and looks for those objects that has no relationship to them and deletes them freeing the space.

Mycar = **new** Car();

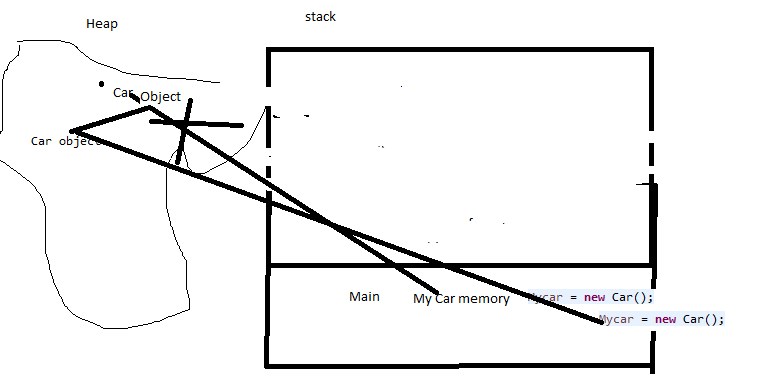
Mycar = **new** Car();

Is pointing to same car object

All are local variabls .My car is a reference variable , eventhough it is local to main method , Mycar is not containing the actual object , it contains the reference to where the object is on the heap.

Instance Variable

Varaiable that belongs to particular instance.



Car MyCar2 = new Car();

Car object is going to be created in the heap . MyCar2 is going to contain the reference to where those object is located

**public** **class** referenceLocalVariable {

**public** **static** **void** main(String[] args) {

Car Mycar; //reference local variable

Mycar = **new** Car();

Mycar = **new** Car();

Car MyCar2=**new** Car();

MyCar2.hp=120; //every object has hp and will be created in heap

Car MyCar3=**new** Car();

MyCar3.hp=140;

}

}

**public** **class** Car {

**int** hp; //horse power

}

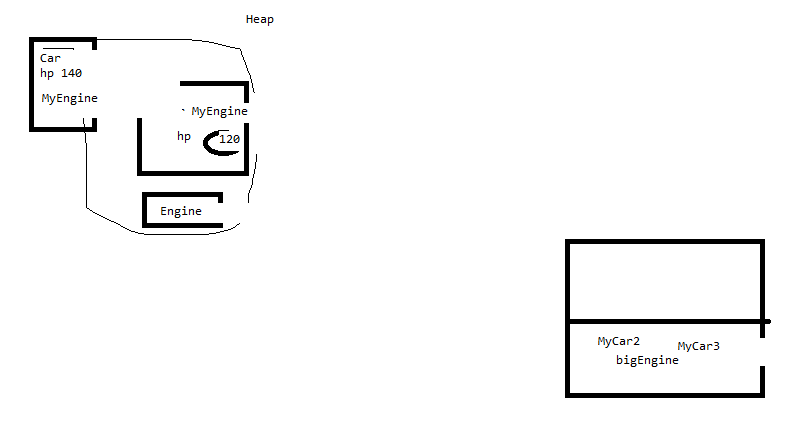
Hp is an instance variable belonging to every single object that we create of this class . It is going to have a spl slot in the heap that is going to maintain data for hp for every single object

Instance variable is used to maintain state for that particular object

// code – reference local variable and engine

Int hp – primitive data type.

is not a primitive datatype but a complex datatype



In heap , my engie hp also has car object.

**public** **class** referenceLocalVariable {

**public** **static** **void** main(String[] args) {

Car Mycar; //reference local variable

Mycar = **new** Car();

Mycar = **new** Car();

Car MyCar2=**new** Car();

MyCar2.hp=120;

Car MyCar3=**new** Car();

MyCar3.hp=140;

Engine bigEngine=**new** Engine();

MyCar3.MyEngine=bigEngine;

//MyEngine – complex data type

}

}

**public** **class** Car {

**int** hp;

Engine MyEngine;

}

**public** **class** Engine {

}

ADT

Abstract data type

Data types – set of values and operations you can run on those values

Int data type – add , divide

Int – primitive data type – its predefined by java language as a reserved keyword.

It is a basic building block of language.

Abstract data type – trustworthy containers capable of doing things with data.

You can use them as a tool by creating an instance of a class and invoking their methods. ADT representation is meant to be hidden by the client that uses it.

The implementation of abstract datatype is supposed to be hidden

Abstract datatypes – they are containers for data and operations and you canuse them to do things in a computer program.

Linkedlist mylist = new LinkedList();

Click on LinkedList using control..it will show all the methods.

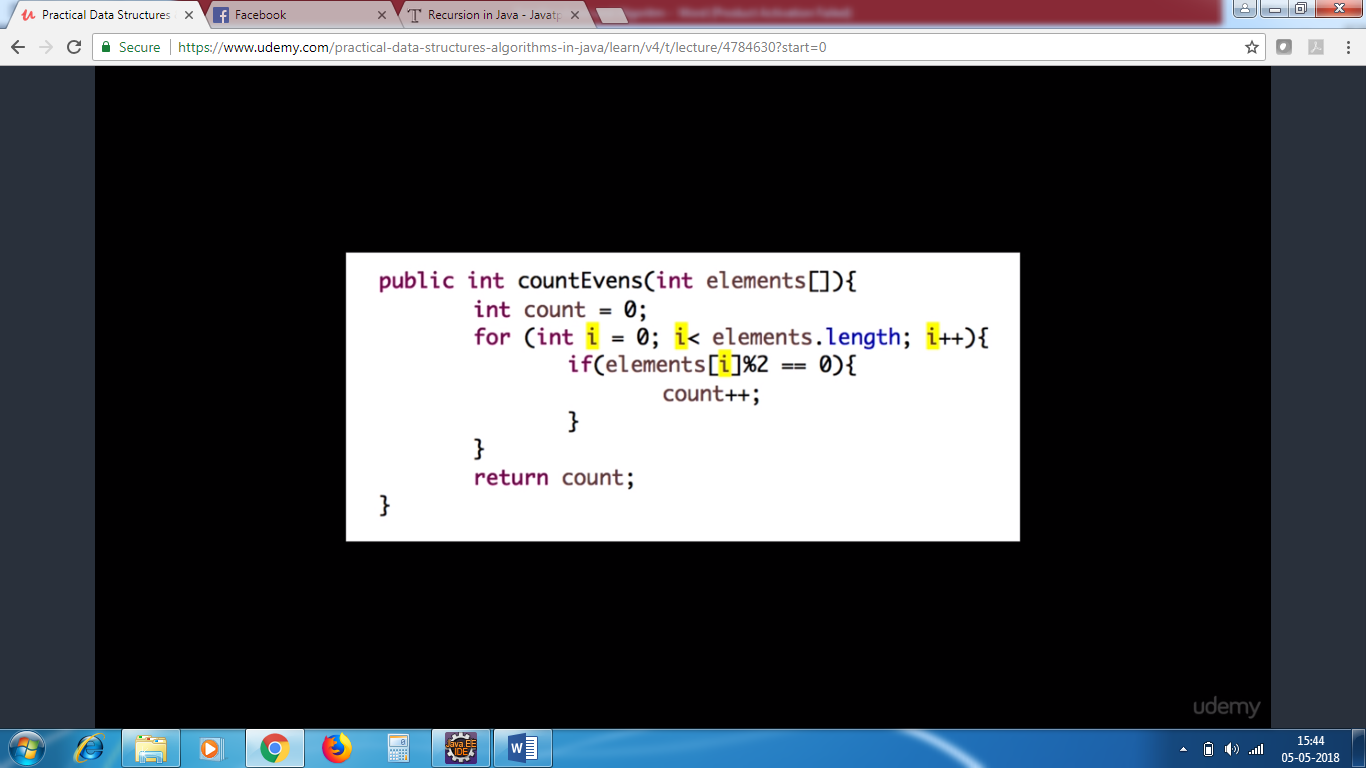
All the public methods will be accessed by mylist. Linkedlist is abstract data type.

Private method cannot be used outside.

INTRODUCION TO ALGORITM ANALYSIS AND BIG O NOTATION

A computer program is basically a list of step by step instructions on how the computer should go about solving a problem. There is a term – step by step is called algorithm.

Count no of even

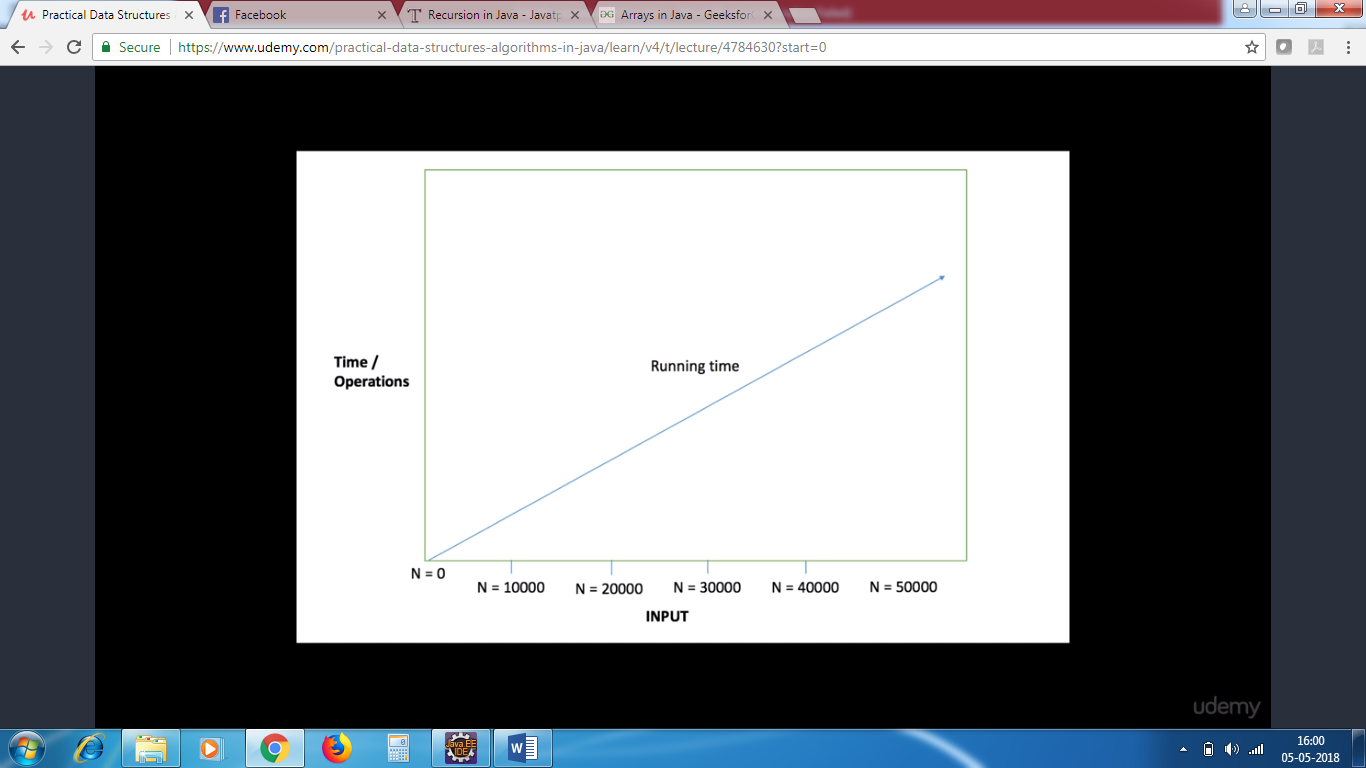


The lines of code is called as algorthim.

Some algorithms perform very slow and few are very efficient.

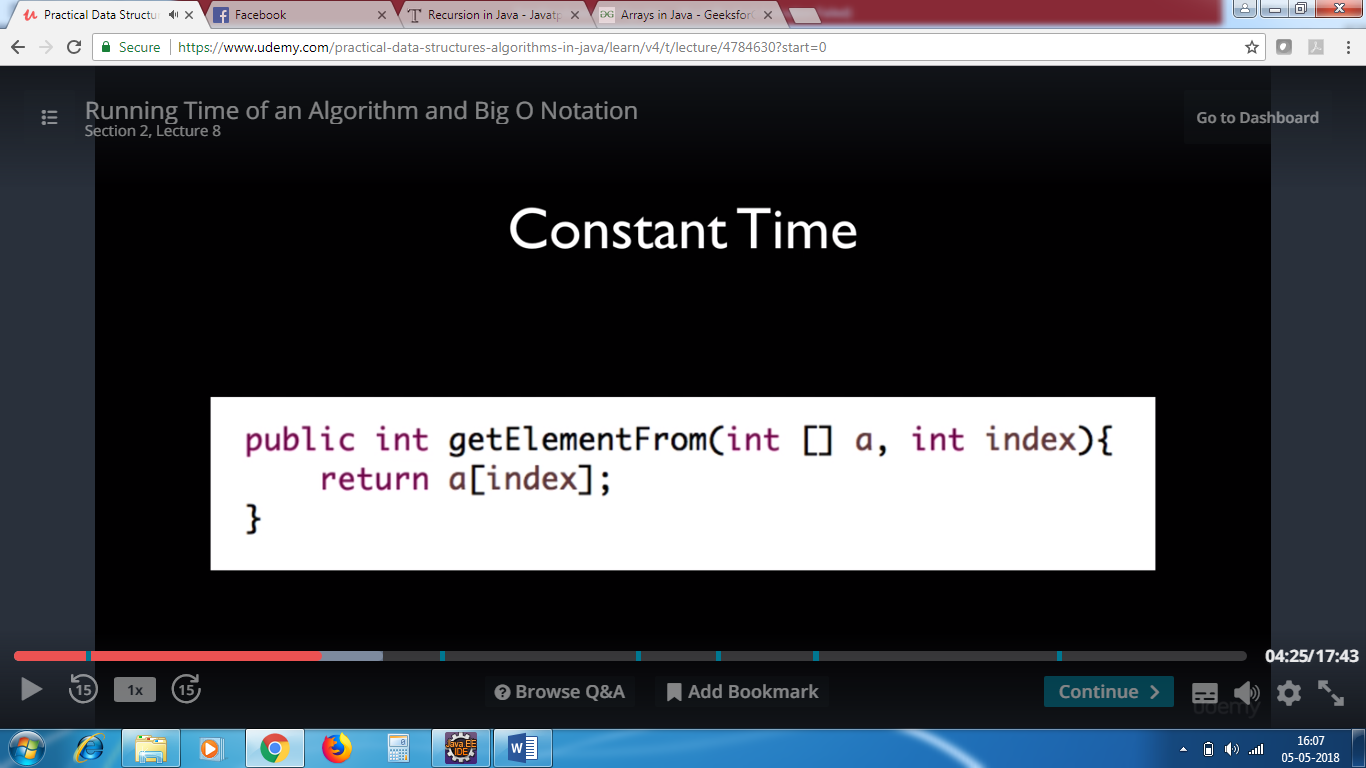
In the above example , the loop will run 50 times if the element size is 50.if you have 100,000 elements , it will run 100,000 times.

Time depends on the size of the input.



They run in linear manner. The loop runs based on the number of input. This algorithm is generally acceptable.

Based on the input size, the loop will run, this is linear approach.

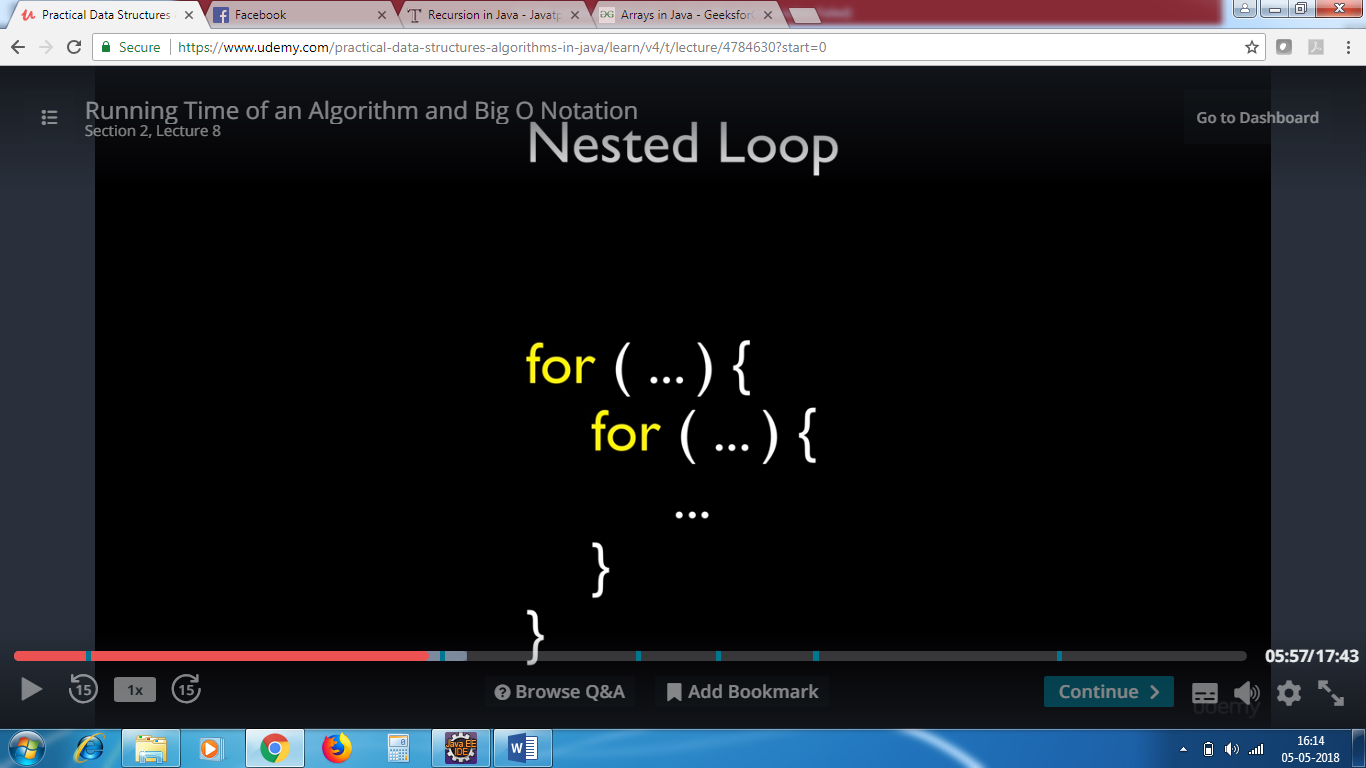


The above program is superfast . Regardless of the no of inputs in an array , the time is constant.

It returns the data based on the index. There is no looping



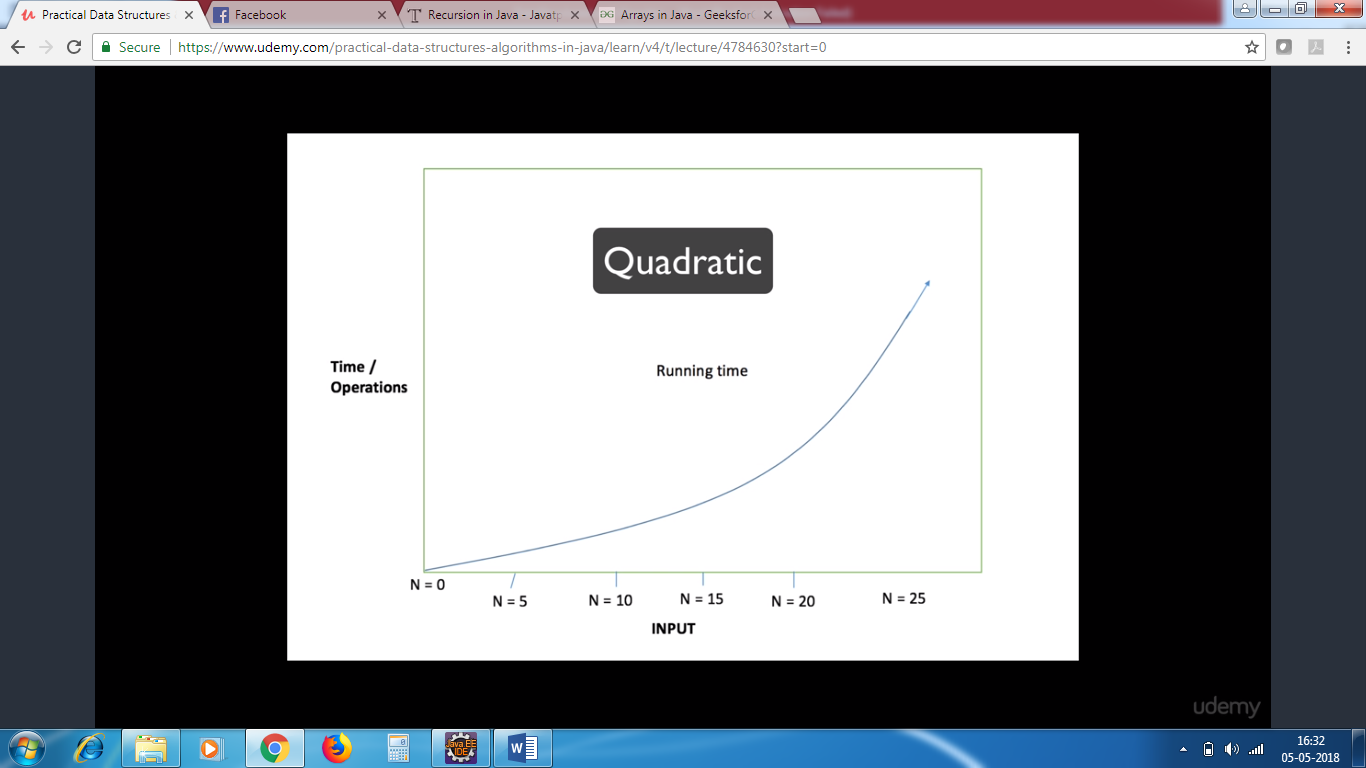
Algorithm that runs very slower

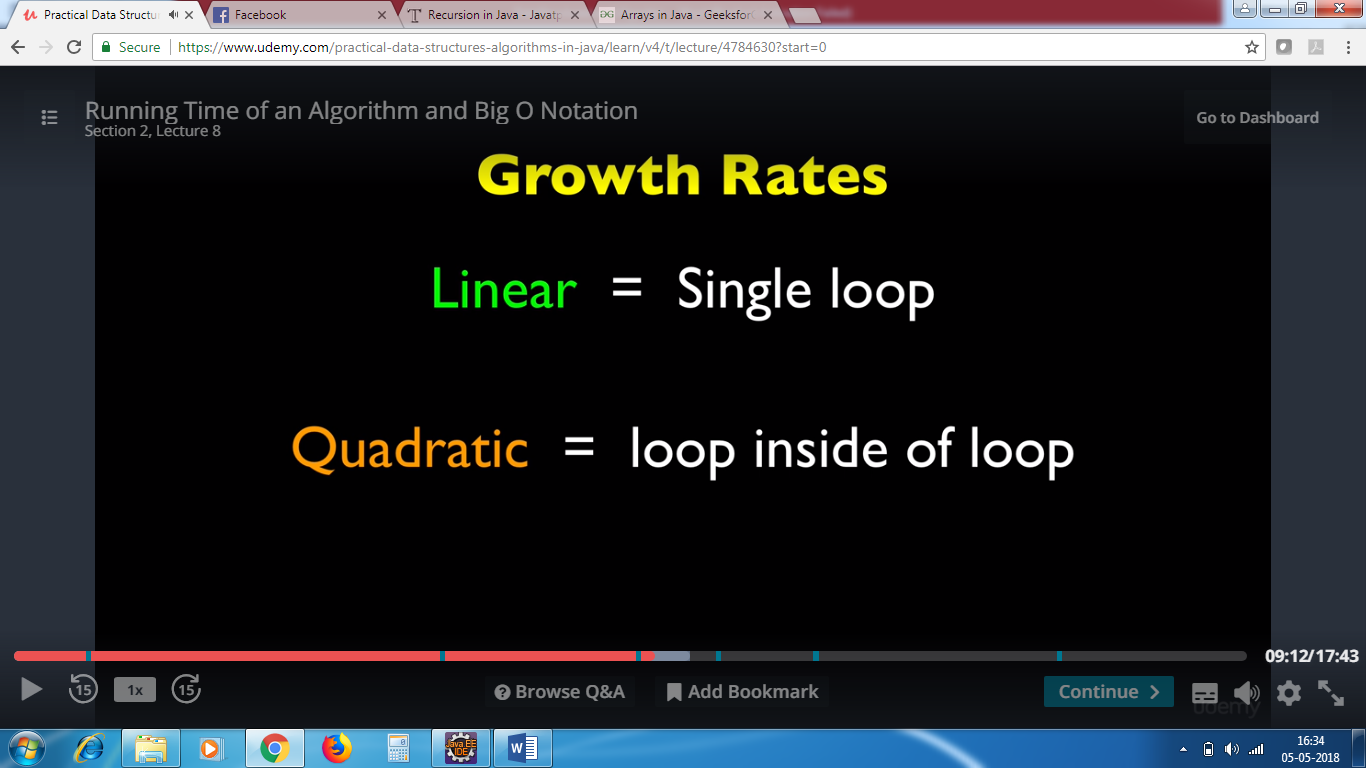




This is not linear. If the array 1 contains 10 elements an d array 2 contains 10 elemnts , it results in 100 iterations. That is ok. If 1st contains 1000 elements and 2nd array contains 1000  elements ,then it results in 100,000 times looping which is not gud.

The number of iterations will be size of the input square. This is not linear / constant but quadratic in nature.





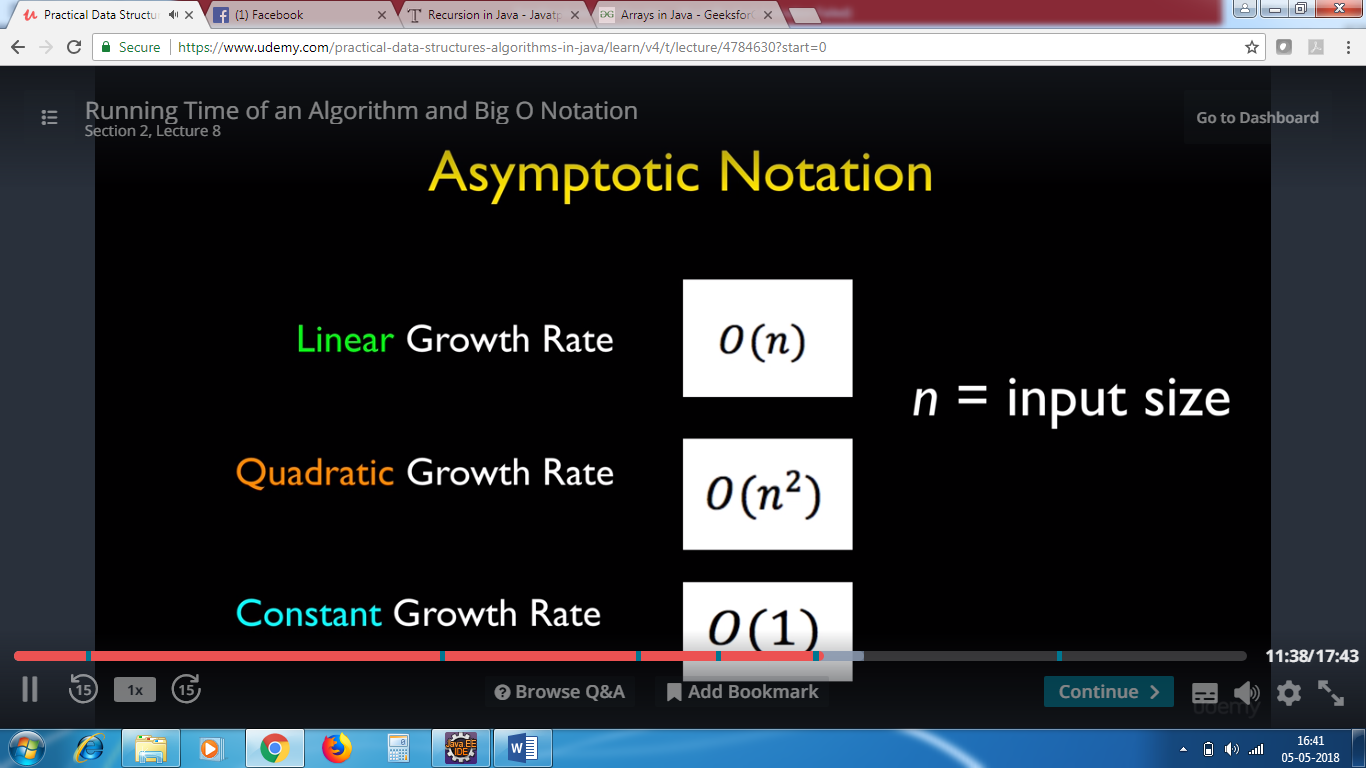
Asymptotic Notation

When an algorithm size linearly – it is o(n)

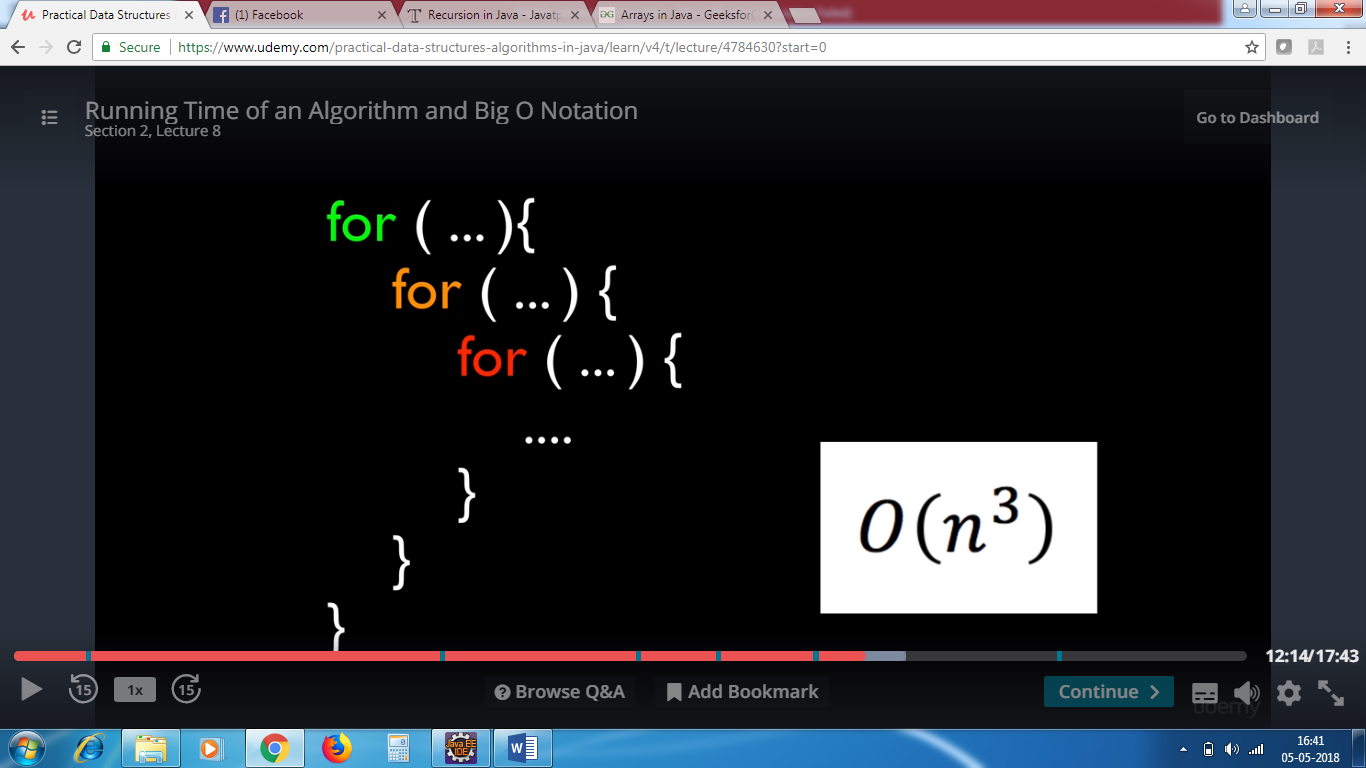
BigO(n)

BigO(n^2)

n=input size

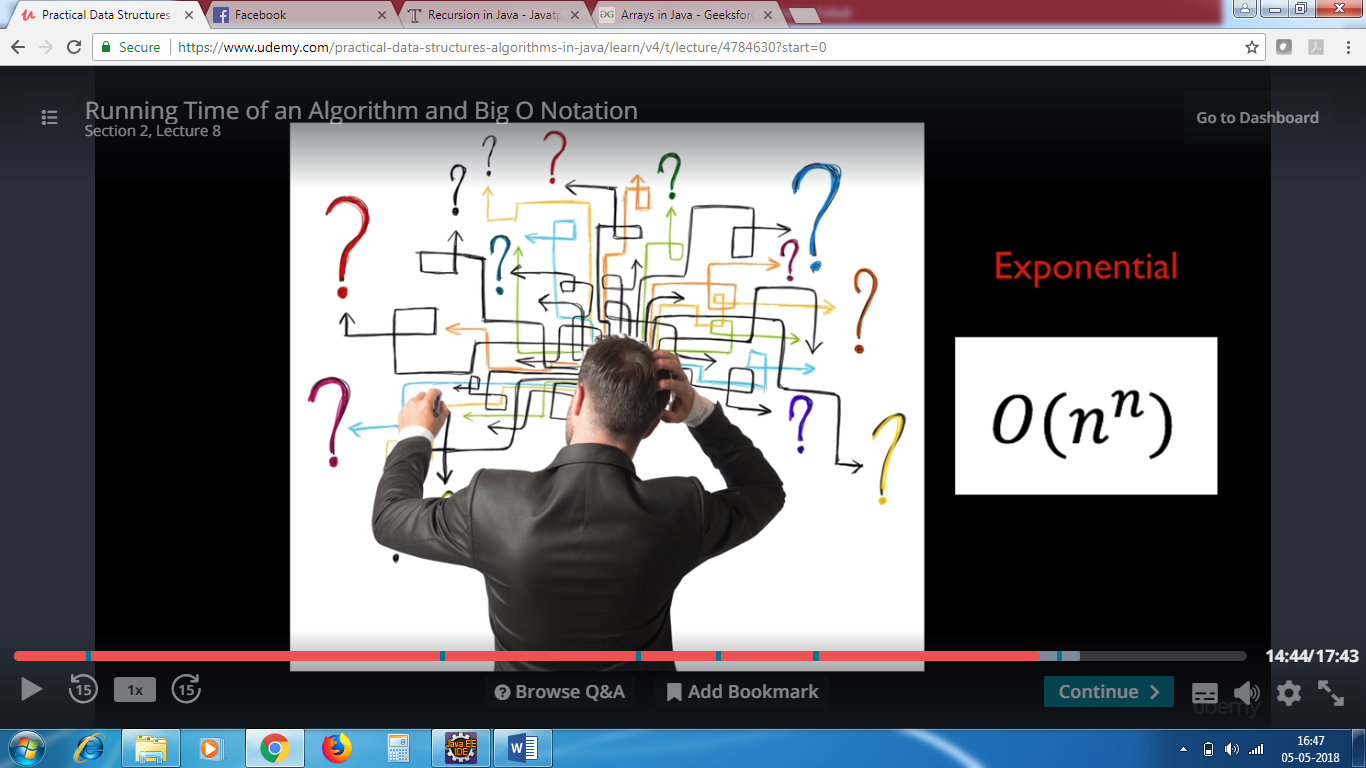


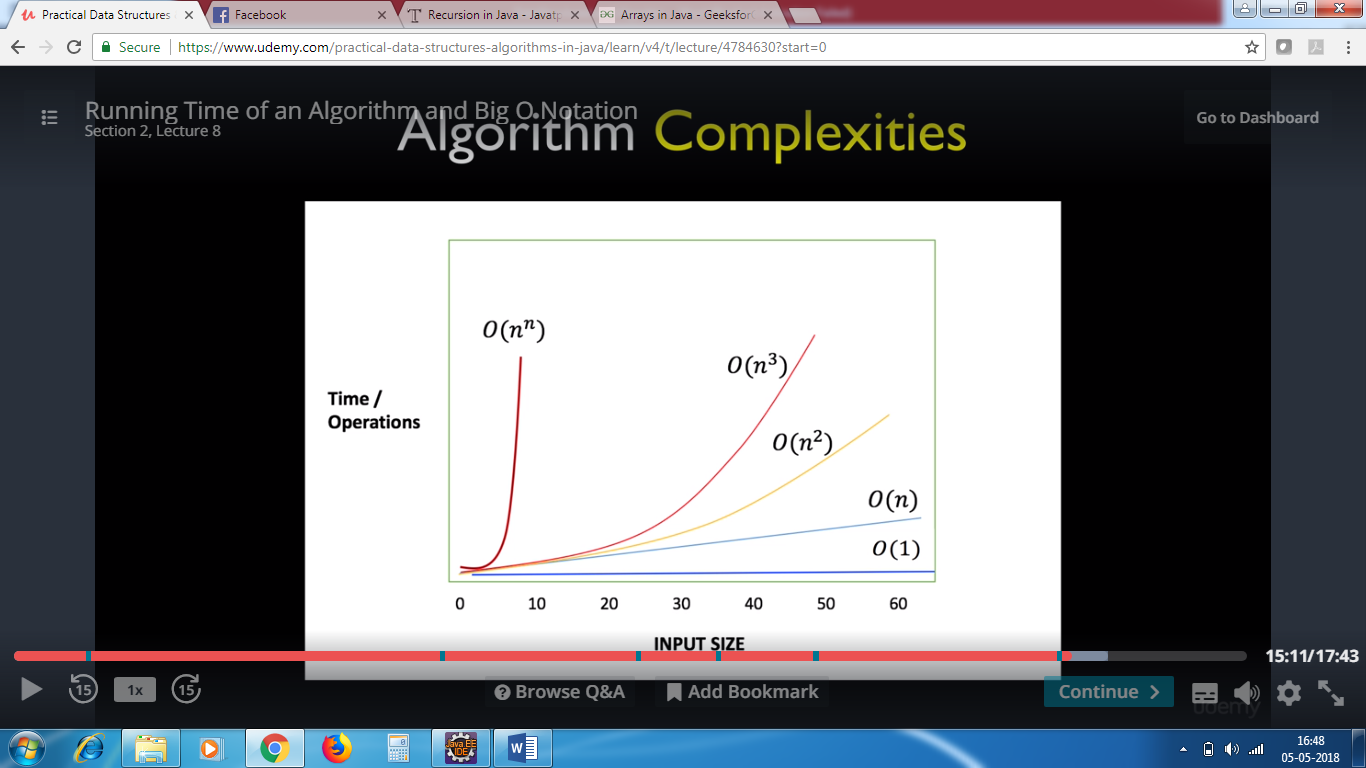
Cubic growth rate



If there are 10 elements – 10\*10\*10 = 1000 iterations

100 elements = 100\*100\*100 = 1 million





Linear / Growth rate –loop grows based on size of input – 1 loop

Constant growth rate – no loop , by index you retrieve o(1)

Quadratic growth rate – 2 loops o(n square)

Cubic growth rate – 3 loops o(n cube)

Exponential growth rate – o(n^n) – if we don’t know how many loops are required.

STACK

Array – not dynamic

STACK :Bunch of things piled up of each other.

Mails – after few days it does get high enough .Most ppl start from top of envelope. This tower of things is an example of stack.

Items that are last inserted in the stack are the ones that are first removed. Old mails get buried at last.

STACK - LIFO – Last IN FIRST OUT

INSERTING MAIL – PUSH

REMOVING MAIL – POP

QUEUE

First item inserted is the first to be addressed. First cum first serve. Maintain the order

Eg : hospitals – patients – first in

PRINT QUEUE – FIFO

When we insert things in queue , they go at the end

When leaving ,it is the opposite

Linked List :

Storing collection of items. Similar to arrays .Arrays also is used for collection of items .

Drawbacks in Array:

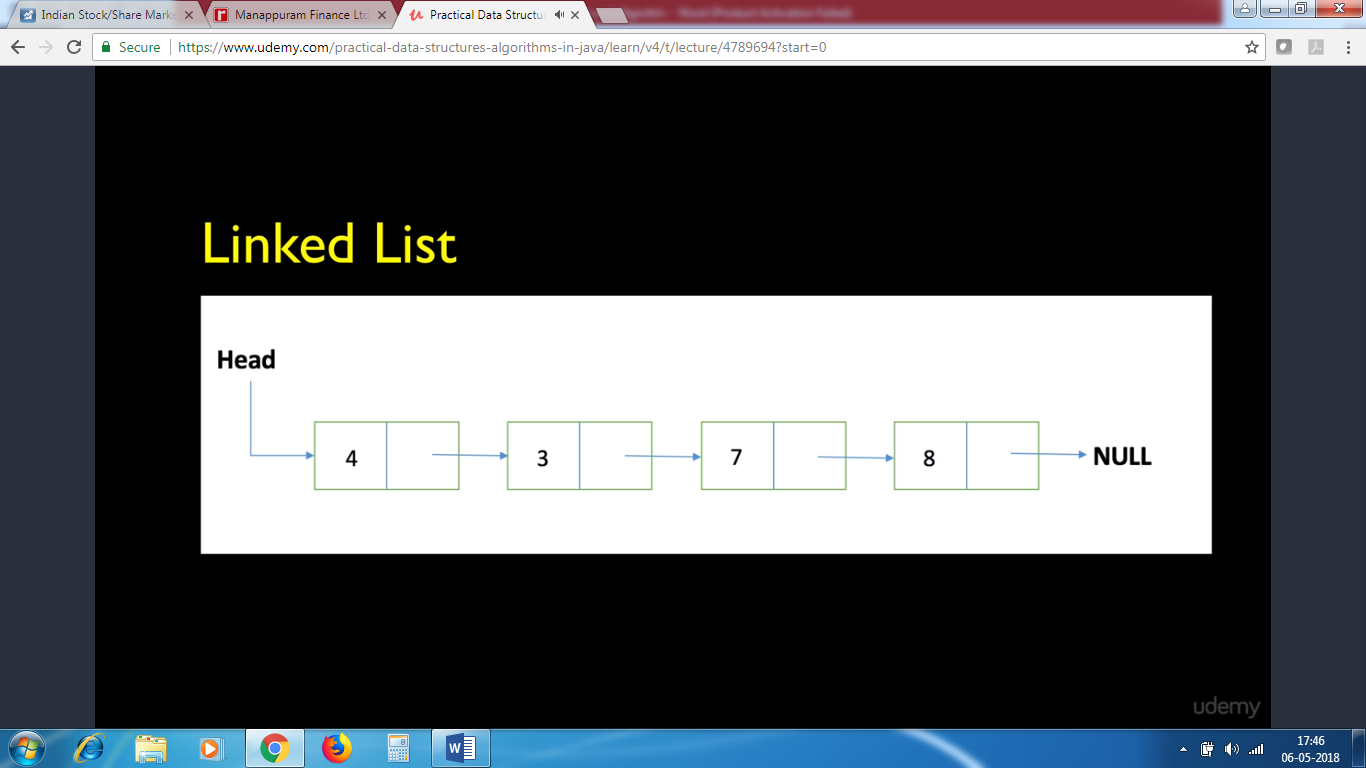
* Slots cannot be increased
* Int[]myArray=new int[100]

I cannot store 101th item . I cannot increase the size of the array.

* If u want to insert the data in the middle of the array , u cant tell all the items to move over and make some room . You need to iterate over each array and reassign the items .
* It is a fixed size
* These are overcome by linked list

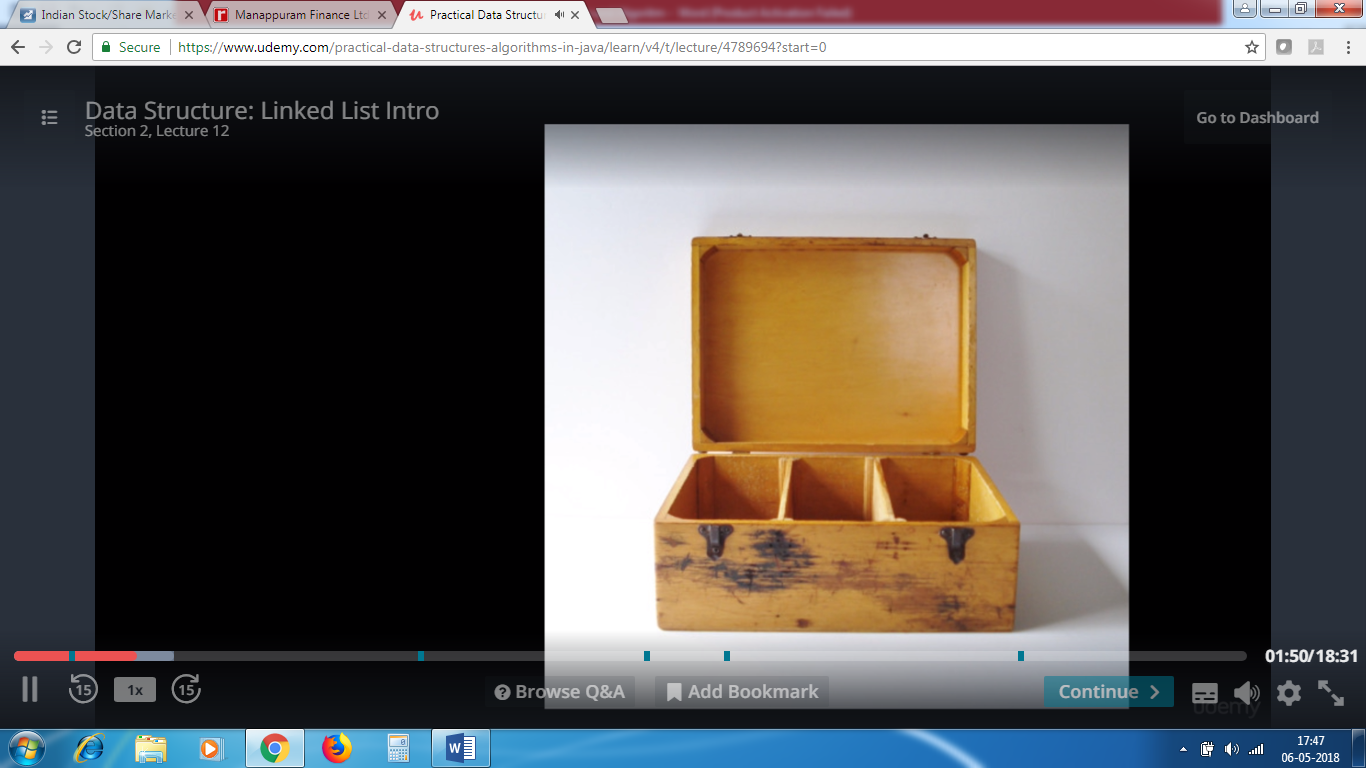
Linked list :

You can insert data at the beginning of the array, end of the array or in the middle(all the data are automatically pushed over). There is no fixed size



Pic of array

I cannot insert 4 items – we have to create a box for 4th element



Linked List:

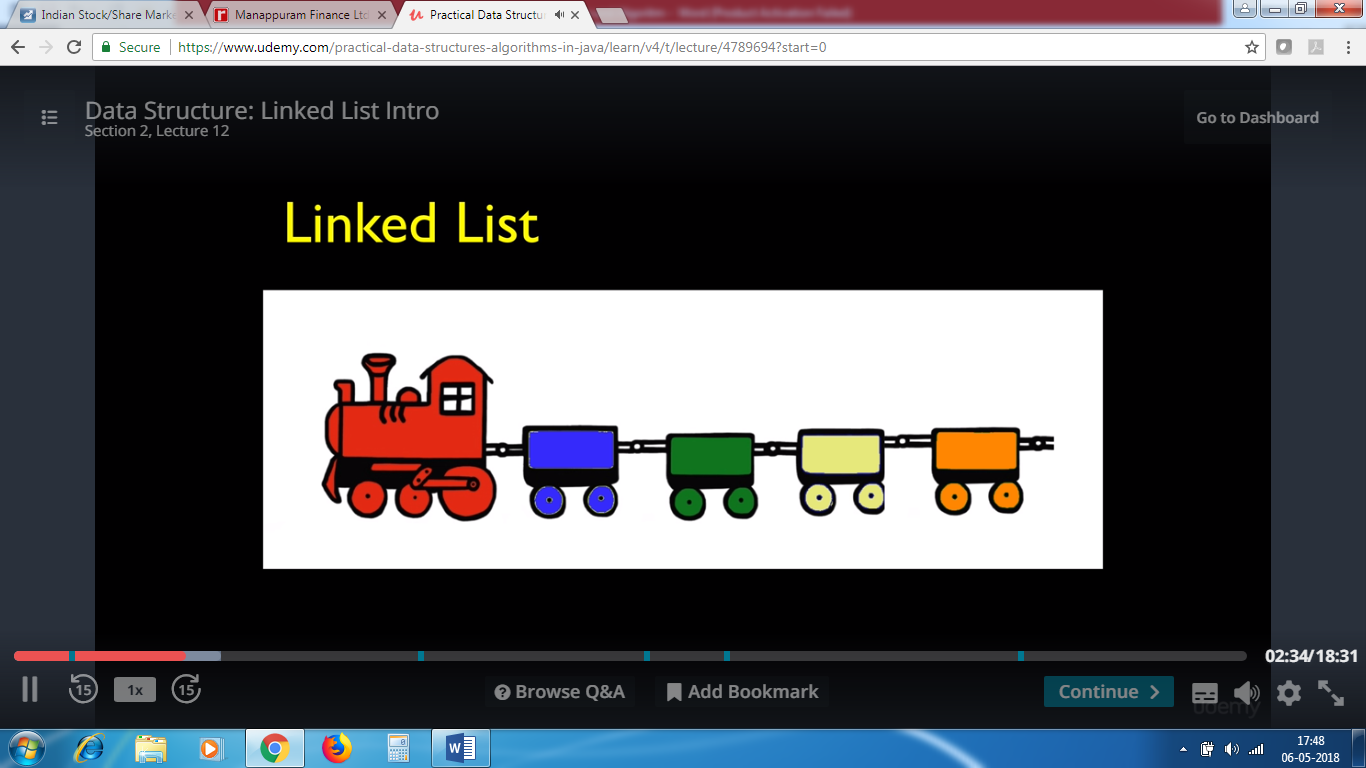
This is a variable data size

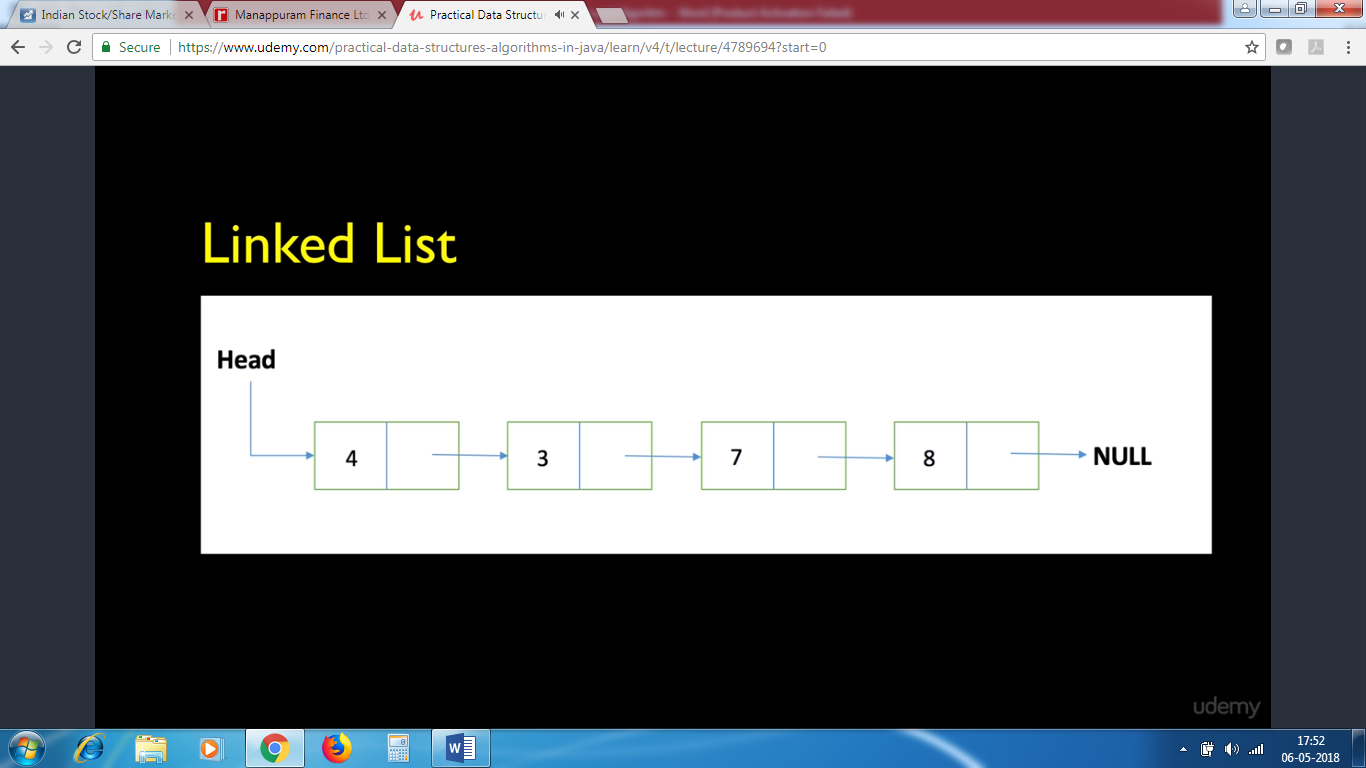
You can change the size of the data

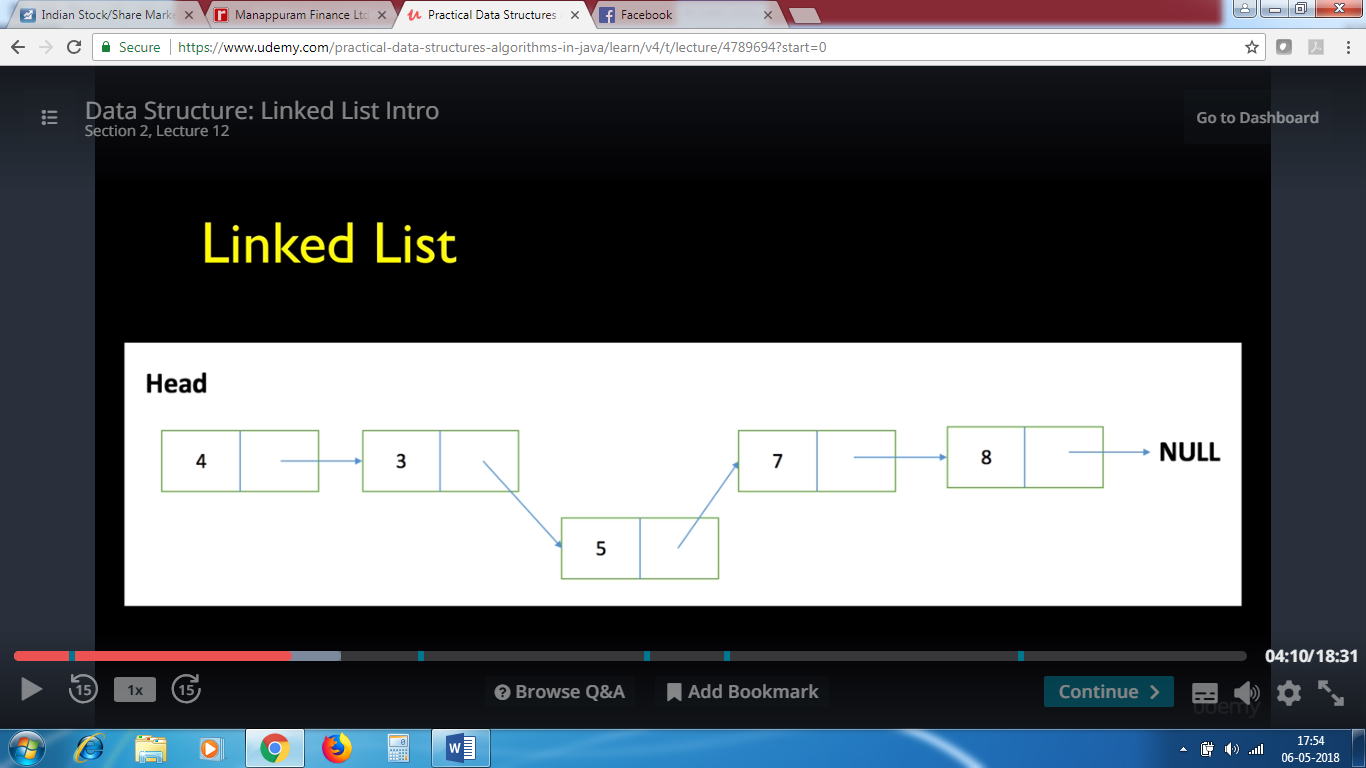
You can add another cart , you can interchange the cart -< green/yellow.

You can add the cart to the front of the linked list(Head) , or you can add a cart to the end of the linked list(Tail).

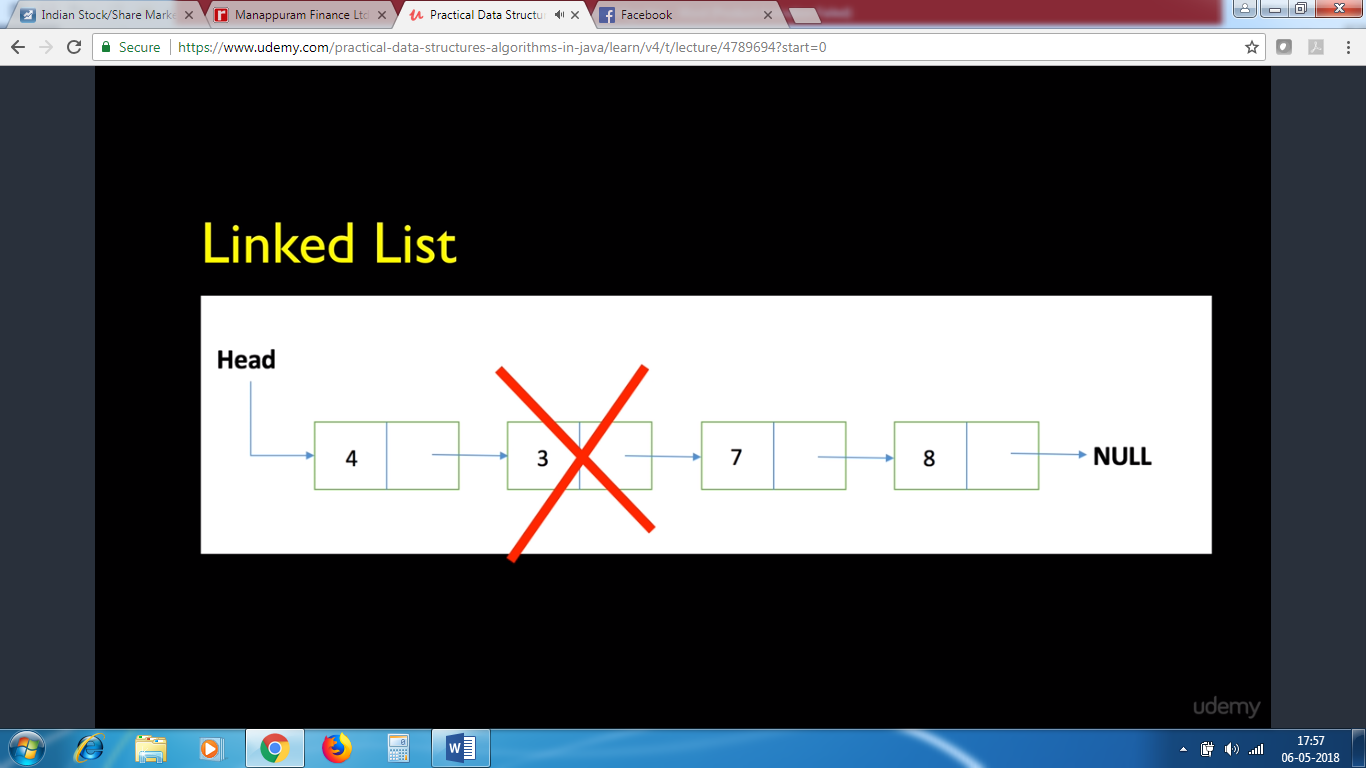
You can add the cart in the middle.



* Each cart is referred as node.
* Change the links to point to other node
* 



You can make node 3 to point node 5 and Node 5 to point 7

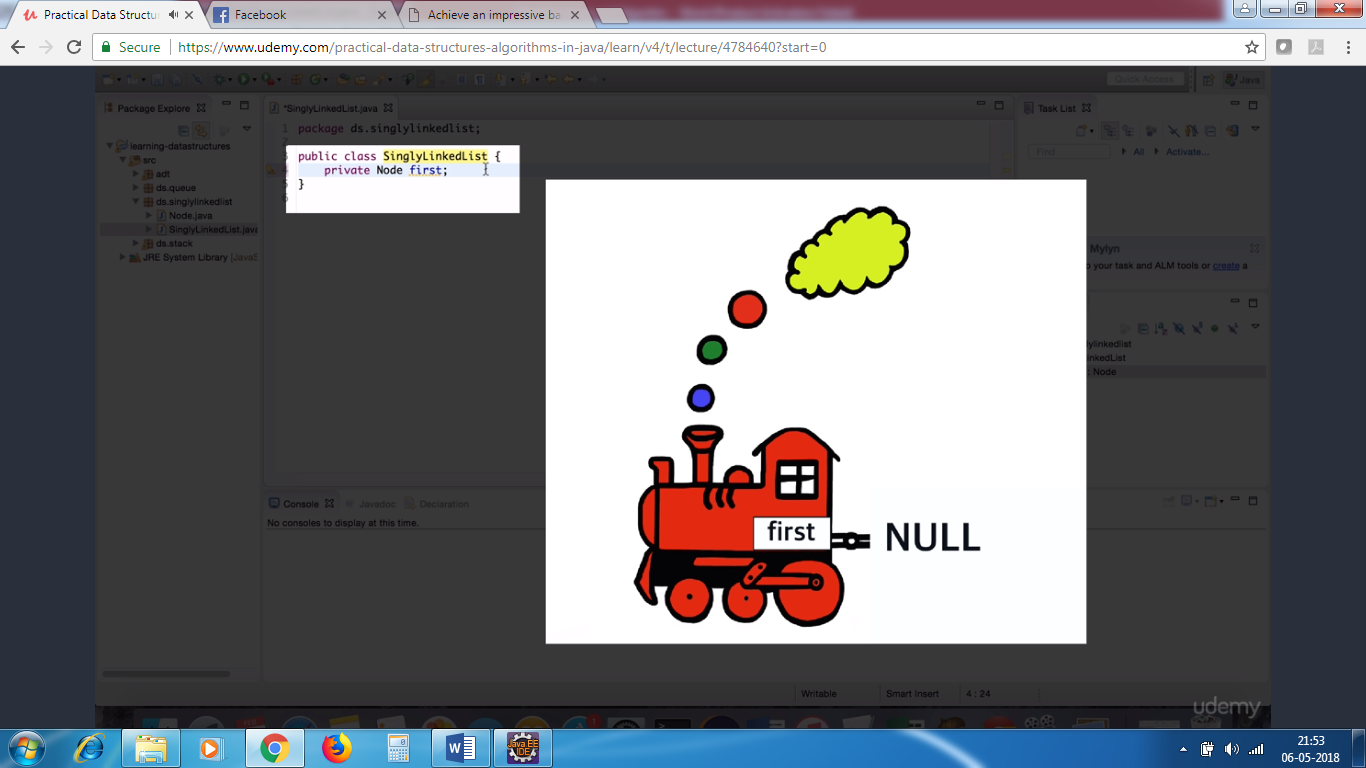


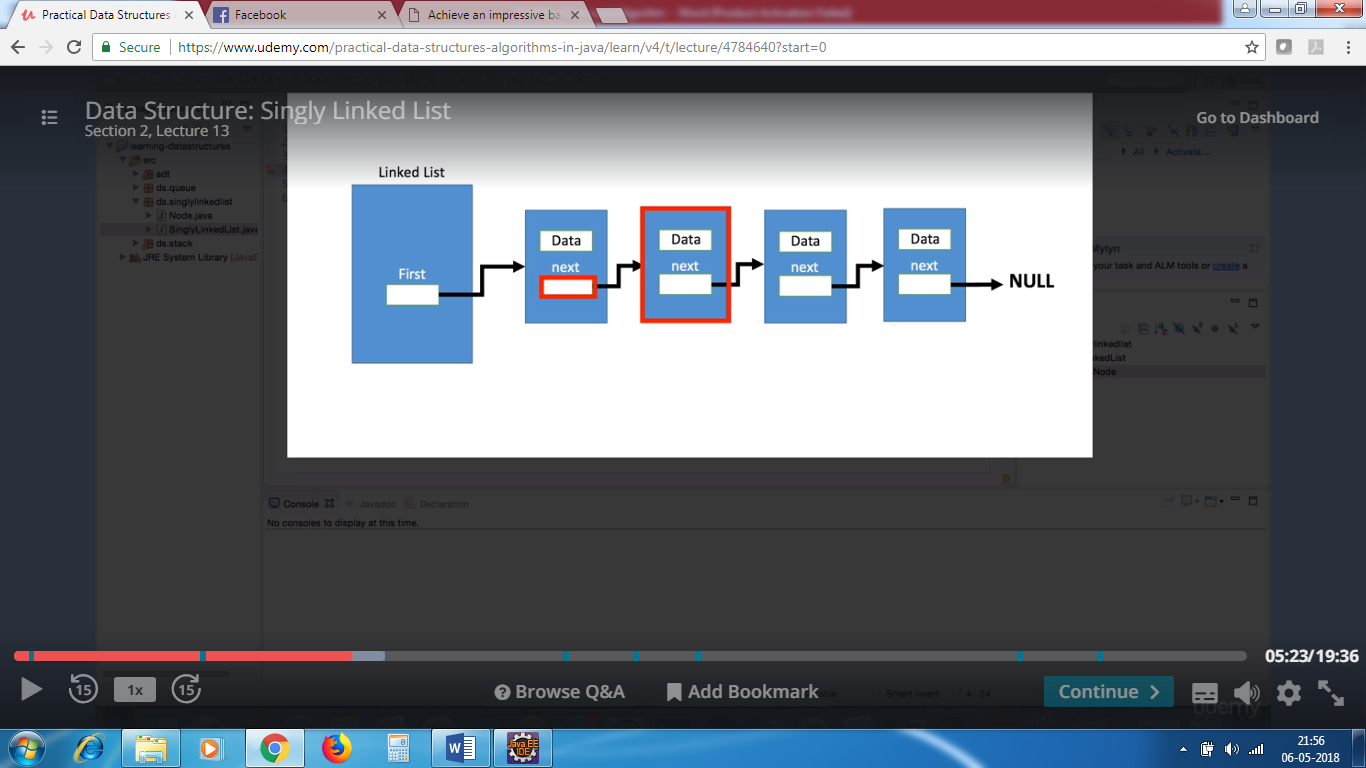
You can make node 4 point to node 7

The last node always points to null.

We will be using node to store the data. It is easy to manipulate the items when compared to array

Singly Linked List



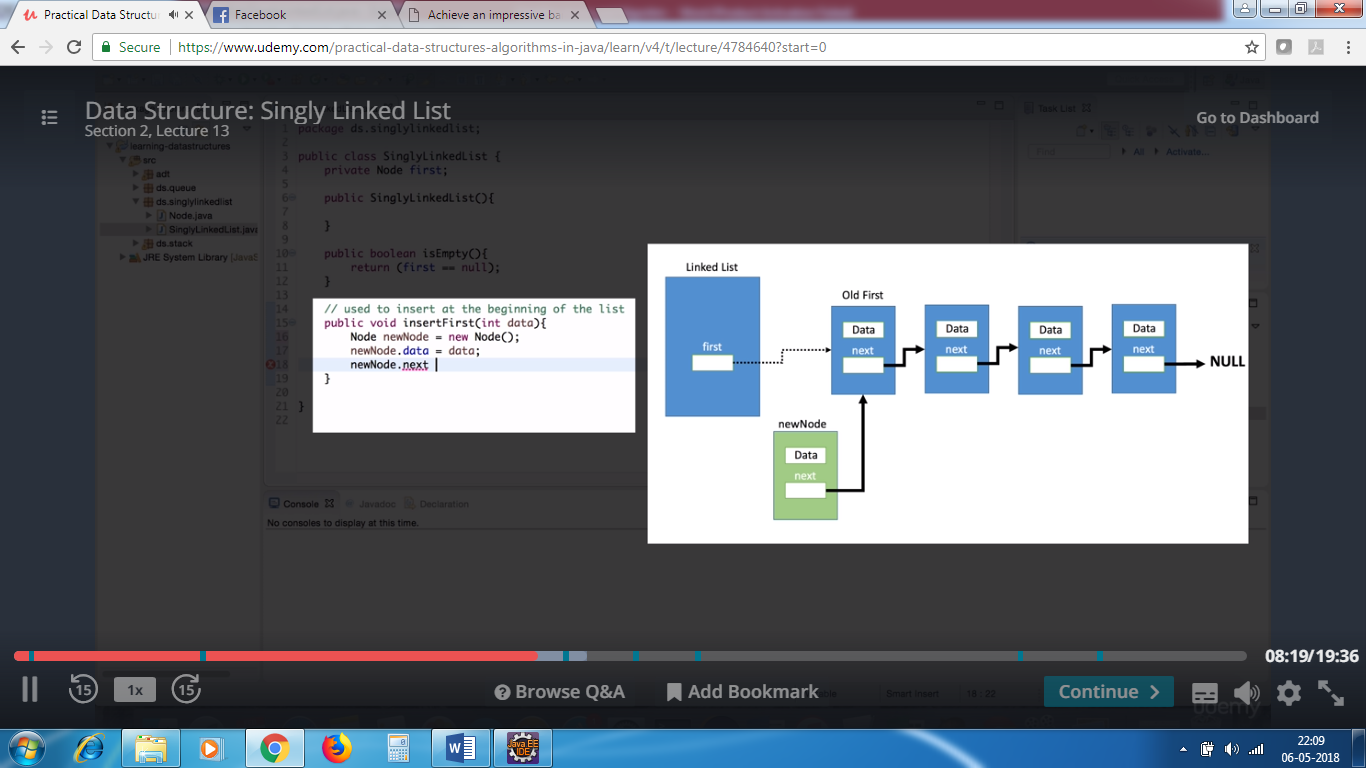


One node knows about the next node . (Node A knows about Node B , Node B knows about Node C..

But Node B doesn’t know about Node A..Node C doesn’t know about Node B.

We just have null method in the Node class..Thats y..Its one directional .So its called as Singly linked list.

Singly linked list has access only to the first node. We can do variety of operations.



Circular linked list:

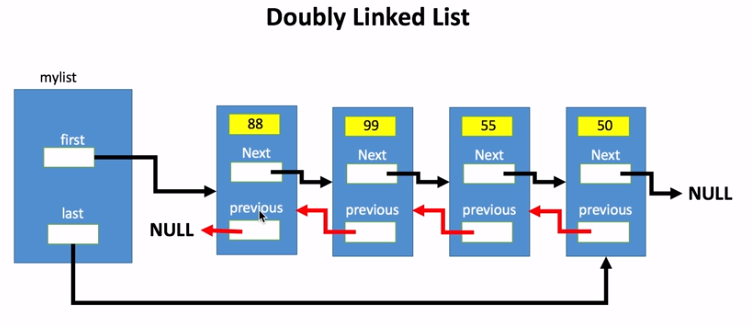
In the circular linked list , the beginning and end of the train are both accessible to engine(start node and end node).The linked list class knows about the first node and as well as the last node.The middle nodes only knows about the node that follows them.

Doubly linked list:

The first variable points to the first node

The last variable points to the last node.

The previous node knows about previous node. This allows to insert things in the middle, insert things at the end, and delete this from middle, end



Notice that first previous node points to null just like last next node points to null.

Linear search

Pseudo code : English way to communicate to other people how a particular algorithm operates.



Binary Search

* Data should be sorted
* Like books in shelves sorted by author name.
* If not sorted we use linear search
* I can go to middle of the of the shelf , and If I find book with author starting with U ,then I can eliminate past to u , I can previous to U.

ELIMINATE<--E | S | U🡪ELIMIMATE

We are narrowing in the problem by breaking the problem size in half in every iteration.

PSEUDOCODE

BINARY\_SEARCH(X,A)

X- value we are searching for

1. Array we want to search the value in

At any given point of time , we are only interested in subset

P

R

Q

P Q R – index positions of the array

P – 1st index position

Q- middle index position

R – last index position

A[P----R] – range we are interested in the slots or subset of the array.

We need to find index position of x.

1. P=0 R = A.length-1
2. While p<=r , do

a)set q=p+r/2 – gives the midpoint – we want whole no

b) if A[q]==x , return q – we have found the value

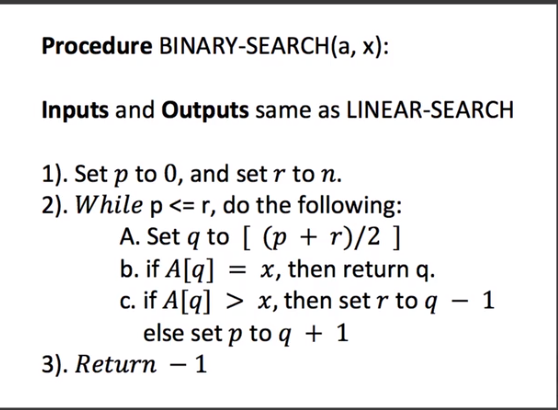
c) if A[q]>x , the range should be P to q-1 and q to r can be eliminated

so , set r=q-1

else

p to q should be eliminated , and we set p=q+1

1. Return -1 we were not able to find the value



Recursion

Method that invoke itself

Foo{

Foo() - invoking itself is called recursion

}

Stack is a memory area reserved to maintain method invocation. When the application starts up , the first method invoked in main method

Main(){

Bar();

}

Bar(){

doWork();

}

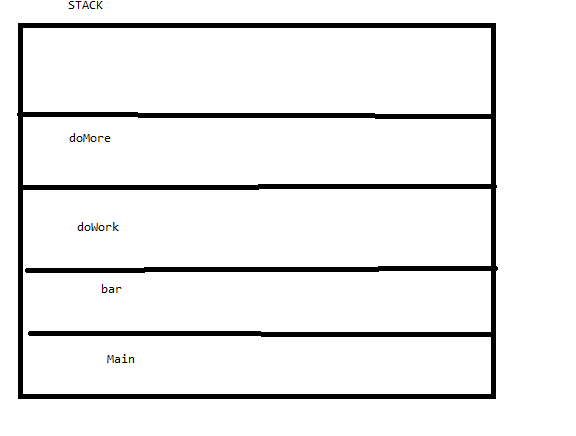
doWork(){

doMore()

}

Bar waits for do work to complete , do work waits for do more to complete .All methods are waiting for that they called to complete.

Once do More is complete it is eliminated from the stack , and when do work is complete, it is elimimated from the stack . When bar is complete , it is also eliminated from the stack.



Lets see when we have recursive method

Main(){

Foo();

}

Foo(){

Foo();

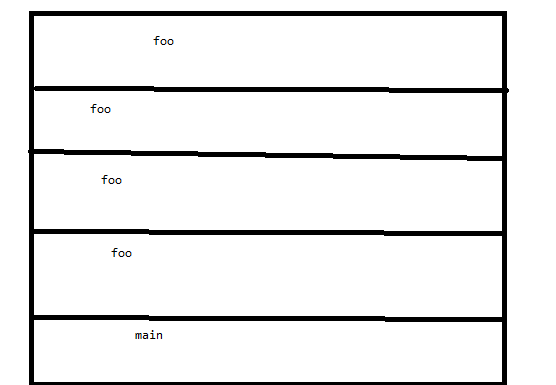
}

Main method – main method is created in stack.

Main method has foo function – the interpretor sees foo function and creates foo function in stack , foo has function foo inside it , so it creates another foo function in stack..

Foo calls itself itself infinite times and functions are created in stack ..and we get stack overflow error. We have run out of stack space.

We need to tell this method , hey stop calling urself if this condition is satisfied.



Main(){

Reducebyone(3)

}

Reducebyone(int n){

If(n>=0){

Reducebyone(n-1);

}

}

Reduceby3 is waiting for reduceby2 to finish and so on.. -1 is removed from stack first , 0 is removed from stack , 1 is removed from stack..and last main is removed.



**package** algo.recursion;

**public** **class** App {

**public** **static** **void** main(String[] args) {

*reducebyOne*(10);

}

**public** **static** **void** reducebyOne(**int** n){

**if**(n>=0){

*reducebyOne*(n-1);

}

System.***out***.println("completed call : "+n);

}

}

-1, 0 , 1 is printed first because 10 is waiting for 9,8,7…-1 to finish

completed call : -1

completed call : 0

completed call : 1

completed call : 2

completed call : 3

completed call : 4

completed call : 5

completed call : 6

completed call : 7

completed call : 8

completed call : 9

completed call : 10

