Lecture 14

**DECISION TREE:**

Revisiting the tree with weight vector [5,3,2] and value vector [9,7,8] with maximum allowed weight of 5 pounds.

Leaves of the tree are at the bottom whereas the roots are at the top.

Each node except the leaves, the solution of that node can be computed from the solution of its children. in a binary decision tree the two children denote either selecting or rejecting an item and at the top we have the solution from either the left branch or the right branch.

In a knapsack problem we will always have this kind of a tree but in different problems there is a possibility that this may not be so.

this denotes optimal substructure. we divide the problem into the sub problems and find the solution to the sub problem. and then combine it together. in this case the combining is simply selecting the best of the two.

So have we found the solution to solving an exponential problem in linear time? – NO

this algorithm’s complexity is: Ο(n S) where

* n is the number of items in the list
* S is the size of the knap sack(Roughly speaking)

And also it takes ‘n S’ space.

This is a rough approximation only. Actually the time and space requirements are dependent not only on the size of the problem but also on the size of the solution. That is, it also depends on the number of items that we would end up fitting into the knap sack.

an entry is made in the memo whenever an item and an available size pair is considered. and as available size becomes 0, we can’t include anymore. so the number of things to be remembered is the same as the number of the items that can be fitted into the knap sack. and the running time is almost same as the amount of items to be remembered.

and this is not how we like to talk about the complexity of an algorithm. as we don’t know the size of the solution without solving the problem.

what we have here is a pseudo polynomial algorithm that is it is a polynomial in the size of the solution.

\*\*NOT CRUCIAL\*\*

a pseudo polynomial algorithm has running time:

* polynomial in the numeric value of the input

Numeric algorithm – exponential in the number of digits in the number

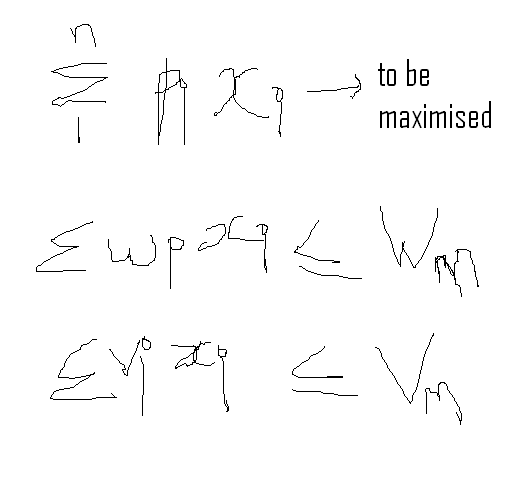
Ο: size of the coding of the input

that is the number of bits required to represent the number inside the computer.

When we start talking about complexity it can be very subtle.

**SLIGHT VARIATION OF THE KNAP SACK PROBLEMS:**

If there is a constraint on the volume and the weight of the number of objects:



to change the code just check the constraint of the weight and the volume.

generally pseudo polynomial algorithms are better than the brute force algorithm. and definitely in this case they would not be worse even if not better.

in dynamic programming:

* we are trading time for space
* don’t be intimidates by exponential problems
* dynamic programming is broadly useful
  + if it uses recursive approaches then always think of dynamic programming
* problem reduction – concept
  + if u can reduce the problem to something that has already solved then we can use the code

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**PYTHON AND PROGRAM ORGANISATION**

Module is a collection of related functions.

We will refer to the functions using dot ‘.’notation

import math

math.sqrt(11)

we have the dot notation to avoid name conflicts.

**CLASSES**

we are going to use classes in the context of OOP.

typically to create data abstractions or abstract data types.

these concepts are about 40 years old.

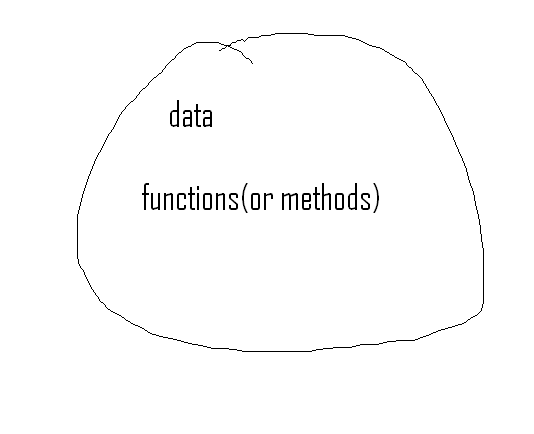
only after the arrival of JAVA, OOP got public attention.

**WHAT IS AN OBJECT?**

An object is a collection of data and functions. functions that operate on the data. key idea to bind the data and the functions together.

here we will create user defined types.

and the this binding is called encapsulation.



people normally talk about this in terms of a message passing metaphor. it is just a metaphor, just a way to think about it and nothing deep.

one object can pass a message to another object and the receiving object responds by executing one of its methods.

The notion of an instance:

we create instances of types.

a class is a collection of objects with characteristics in common. the in built methods can be thought of as methods of classes.