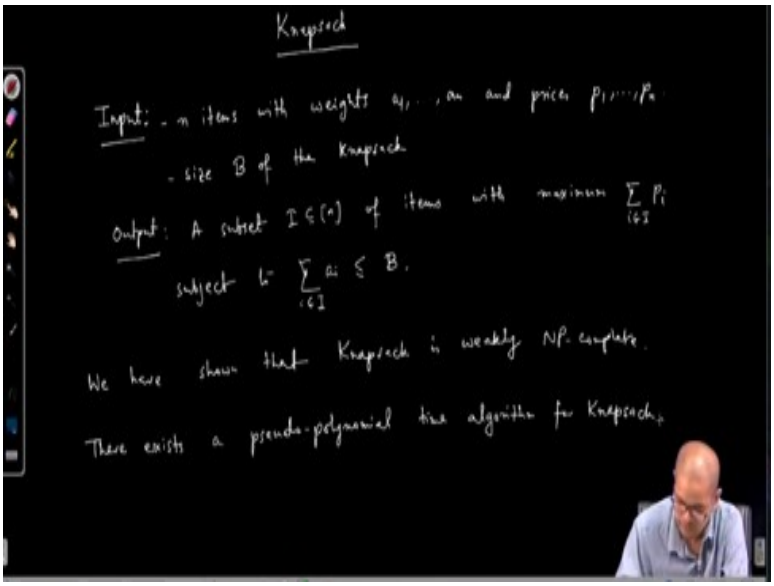
Selected Topics in Algorithm
Prof. Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 46
FPTAS for Knapsack

Welcome. So, in the last class we have seen a login factor approximation algorithm for the set cover problem. So, today you will see the knapsack problem and we will see what is called a fully polynomial time approximation scheme for knapsack.

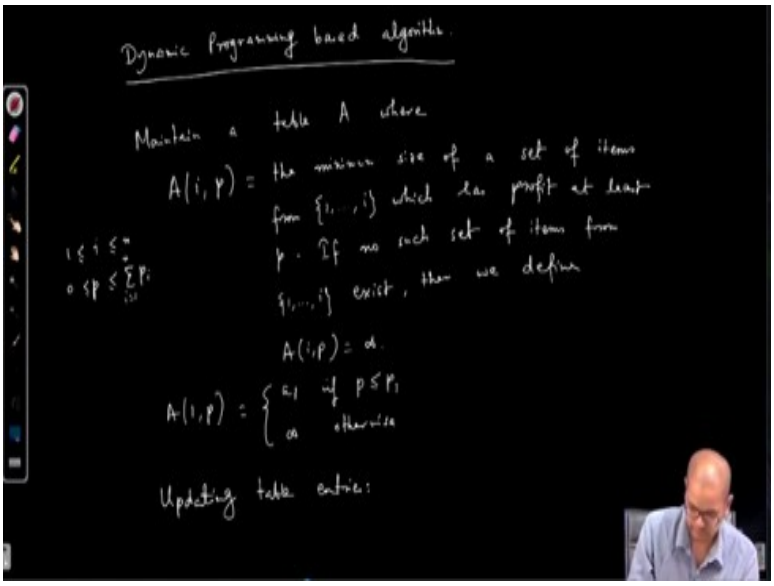
(Refer Slide Time: 00:44)



Today's problem is knapsack. Let us briefly recall the problem statement. Input n items with weights $a_1, \ldots, a_n$ and prices $p_1, \ldots, p_n$ size B of the knapsack. Output a subset I subset of n of elements of items with maximum sum of prices subject to the size the sum of the weights of all the items is at most B. So, we recall we have shown that it is NP complete and we have only shown that it is weekly NP complete.

So, we have seen that knapsack is weekly NP complete. So, there exists a pseudo polynomial time algorithm for knapsack. So, first let us see that pseudo polynomial time algorithm.

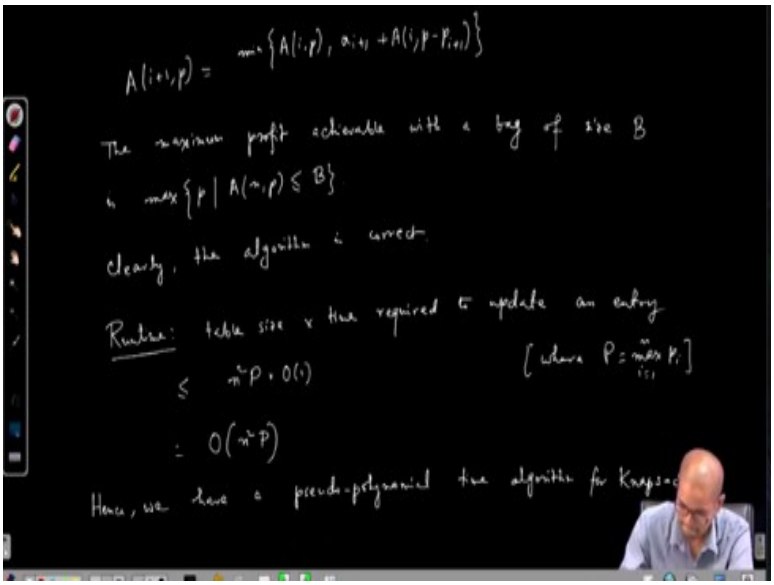
(Refer Slide Time: 04:43)



So, it is a nice dynamic programming algorithm for knapsack. So, what we do is that we maintain a table A where $A(i, p)$ is the so where i varies from 1 to n and p varies from you know 0 to sum of profits. $A(i, p)$ is the minimum size of set of items from 1 to i which has profit at least p. So, if no sub set of items from 1 to i exist that means for all subset of items 1 to i the price is less than p then we define $A(i, p) = \infty$.

So, we do the base cases for i = 1 and for all p so, $A(1, p)$ is the weight of the item a_1 if p is less than equal to p_1 and infinity otherwise this is how would the base case. Now how do we update dynamic programming table.

(Refer Slide Time: 09:17)



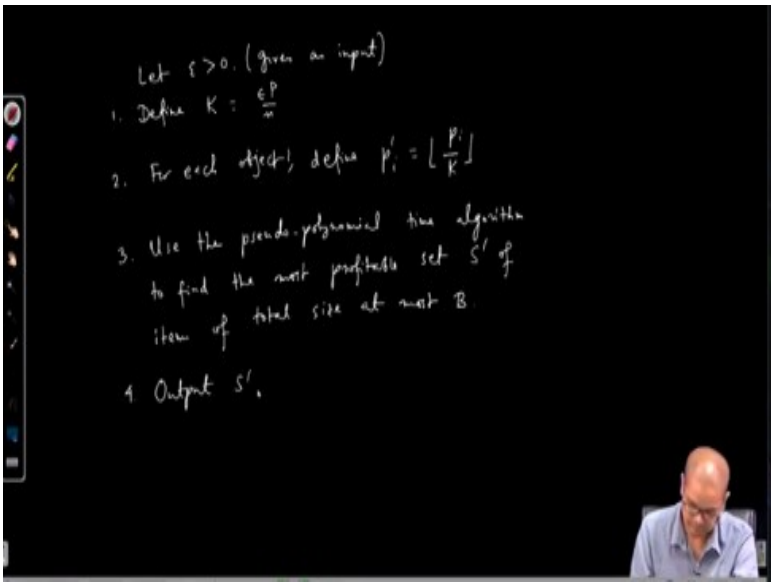
Updating table entries, $A(i+1, p)$ so this is the minimum size of the objects from 1 to i + 1 which has minimum profit the profit at least P. It has two options either so it has two options. This is minimum either it will pick the i + 1 of the item or it does not pick. So, if it picks then if it does not pick then $A(i+1, p)$ is $A(i, p)$ and this $A(i, p)$ if the i + 1 of the item is not pick but if the i + 1 of the item is picked then the weight is a_{i+1} that is the weight + a_i from the items 1 to i.

The profit that I have to achieve is $p - p_{i+1}$. So, this is the update rule here that means. For i = 1 for we do the base case for all p and from $A(2, p), A(3, p), \ldots, A(n, p)$ we update the tables and the output the maximum profit achievable with bag of size B is you look at all the you find the max p max over those profits such that $A(n, p)$ is less than equal to B. So, clearly the algorithm is correct. It follows from the correctness of base case and correctness of update rule.

Now what is the runtime? Runtime is table size time required to update an entry. So, tables are indexed by i and p, i varies from 1 to n and p varies from 0 to $\sum p_i$. Now if capital P is the maximum profit, then $\sum p_i$ is bounded by nP. So, this is less than equal to $n^2 P$ that is the table size and where $P = max_{i=1}^{n} p_i$ and updating our table just take comparing two table entries. So, it can be done in order one time.

So, run time is $O(n^2 P)$. Now if the prices are encoded in unary then this is a polynomial time algorithm hence it is a pseudo polynomial time algorithm. Because typically the numbers are incurred in binary and with respect to the size of the binary representation the value is exponential. So, it is an exponential time algorithm but it is a pseudo polynomial time algorithm. Hence, we have pseudo polynomial time algorithm for knapsack.

(Refer Slide Time: 15:31)

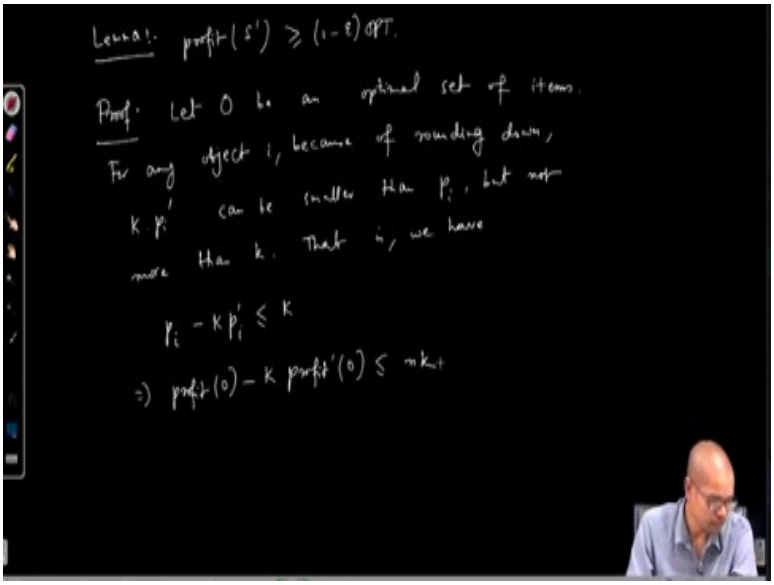


Next what we do is that using this we will build an approximation algorithm $1-\epsilon$ factor approximation algorithm using this algorithm. So, let us first explain that algorithm. So, let epsilon be any positive given part of input given as input define $k=\frac{\epsilon P}{n}$. Now what we do is that we will scale down the profit. The problem with pseudo polynomial time algorithm is that it is it can be very high.

So, what we do is that we will scale down and round up the profits and budget, that is the idea. So, that is the first step second step for each object scale down the profit define $p_i^{'}$ the profit of i. For each object i define $p_i^{'}=\frac{p_i}{k}$ and take its float. We are supposed to have integer profit not rational number. Number three use the pseudo polynomial time algorithm to find the most profitable set of items call it say $S^{'}$ of total size at most B.

So, we are not touching the size we are scaling down the profit because our algorithm has pseudo polynomial dependence on the profit and then we simply output S'. So, here is the claim.

(Refer Slide Time: 18:56)

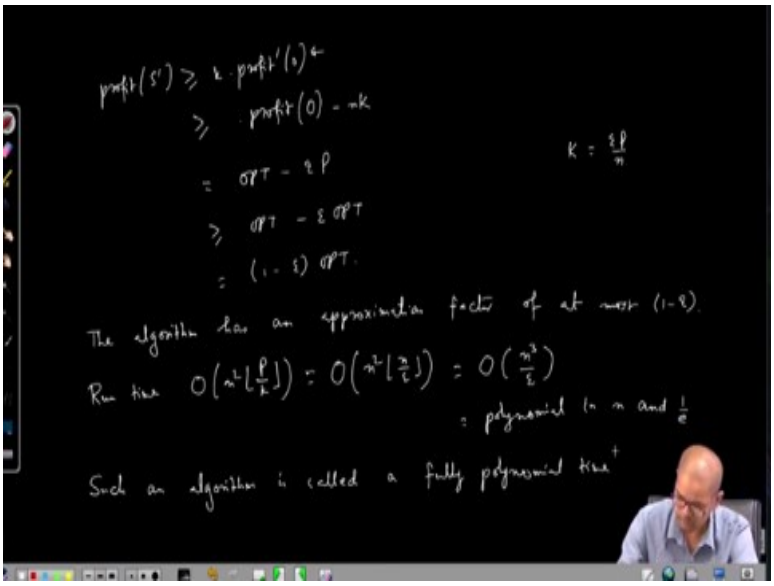


Lemma, you know profit of S' is greater than equal to $(1-\epsilon)OPT$. Proof, so let O be an optimal set of items. Now because each objects each object's profit is scaled down let us see what could be its profit in the modified instance in the scaled down instance. For any object i because of rounding down it may be possible that because of rounding down in the round down instance round down version O is no longer an optimal solution.

K times profit of O can be smaller than profit of K times profit of any object i so we have scaled down by K. So, K times profit of i so K times p_i be smaller than p_i but not more than K. In summary that is we have for each i item it can drop by at most K. So, profit of O that means $p_i - K\, p_i'$ is should can be is less than equal to K. This implies that profit of O - K times profit of o.

If I add this over the objects in O like K times profit prime of O is less than equal to O can have at most n objects is n times K.

(Refer Slide Time: 23:18)



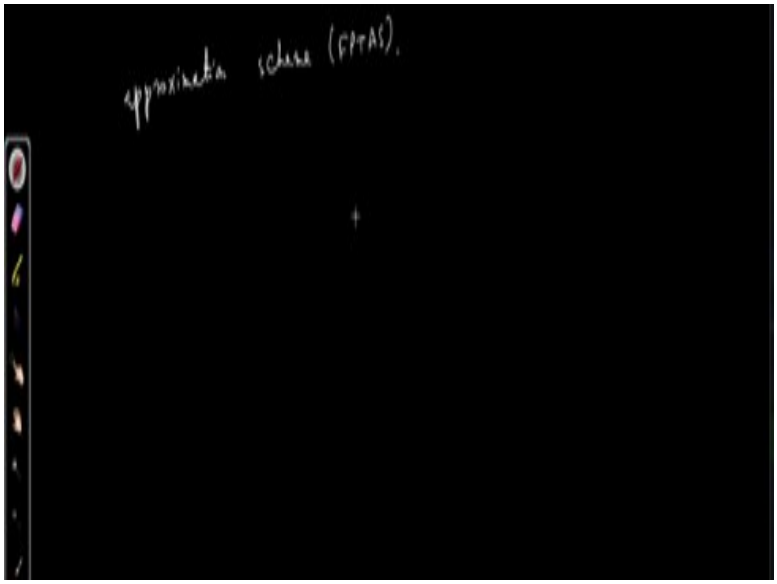
Now dynamic programming must find the best solution. So, what we have profit of S prime is greater than equal to K times profit prime of o. But this is greater than equal to K times profit of so profit of S prime in the original instance this is must be greater than equal to K times profit prime of o but this is K times profit of O - $\frac{n}{K}$. So, K times profit of O is at the profit of O in and that is the minimum profit of o in that scale down version.

So, profit of $S^{'}$ is this greater than equal to K times profit of O - $\frac{n}{K}$ but not K time. K times profit prime of O is greater than equal to profit of O so this is equal to OPT minus and $nk=\epsilon$. So, what was K? K was epsilon P by n so n k is epsilon P and P is the size of the maximum is the highest profit. So, OPT is this is greater than equal to $OPT-\epsilon\, OPT$ this is $(1-\epsilon)OPT$.

So, the algorithm has an approximation factor of at most $1-\epsilon$. The runtime is $O(n^2\, maximum\ profit^{'})$. The maximum profit is $\frac{P}{K}$. Now this floors and filling does not matter in big O notation. This is $O(n^2)$ now and of $\frac{P}{K}$, $K=\epsilon\frac{P}{n}$. So, $\frac{P}{K}$ is $\frac{n}{\epsilon}$ this is $O(\frac{n^3}{\epsilon})$. So, the runtime dependence on epsilon is polynomial so this is polynomial in n and $\frac{1}{\epsilon}$.

So, for every epsilon greater than 0 we have an algorithm whose approximation factor is $1-\epsilon$ and its run time is polynomial in n and $\frac{1}{\epsilon}$. So, such an algorithm is called fully polynomial time approximation scheme in short FPTAS.

(Refer Slide Time: 28:16)



Because it is not one algorithm, it is a we have an algorithm for every epsilon and the runtime dependence is polynomial in the input size and $\frac{1}{\epsilon}$. So, such a thing is called FPTAS. This sort of from approximation side for a NP complete problem this is the best is the holy grail that we can hope to achieve. So, we will stop here today.