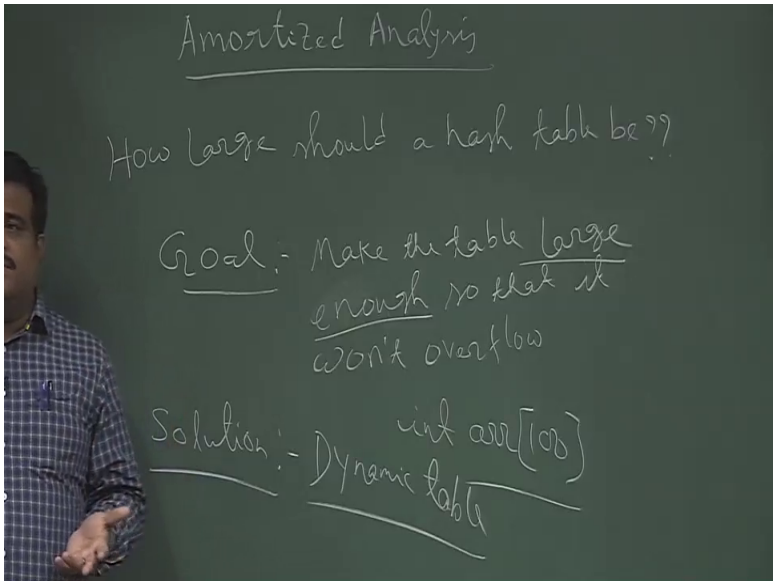
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 33
Amortized Analysis

So we talk about amortized analysis, amortized algorithm. So, what do you mean by amortized analysis of an algorithm? So, for this let us start with the first question which we want to discuss. The question is how large should a hash table be?

(Refer Slide Time: 00:36)

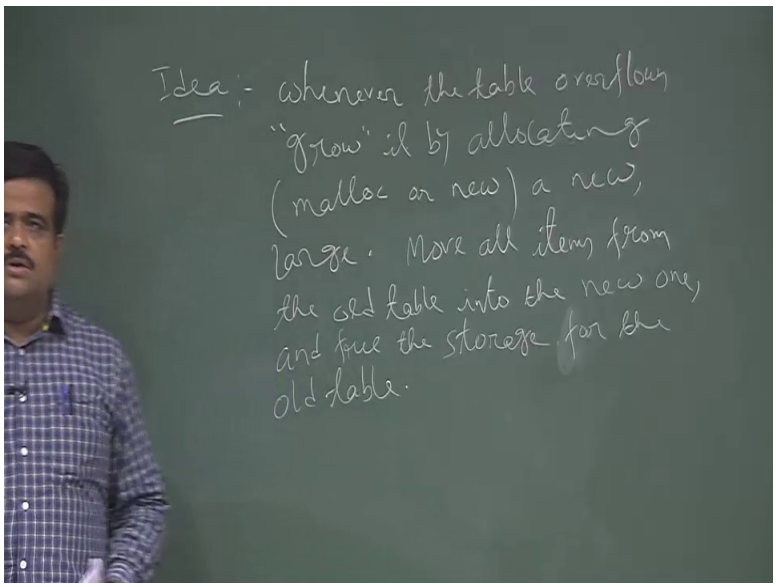


So, this is the question how large should a hash table be? So, I mean how large you should make our table. So, what is the idea? What is the goal? So, what is the idea? So, we want to make the hash table as big as we can. So, that there will be no overflow. So, the idea is, make the hash table large enough so that it will not overflow, otherwise it will be inefficient. And it should be try to make it as small as possible, but the question is how large? That is the question make it hash table large enough. So, this is a what we want. So, how large? Whether suppose in C language we sometimes we do not know the size of the array. Now if you want to allot some statically then sometimes, we do like this int, array, so 100. So, in C language sometimes we declare the our array like this.

We declare up to 100 because 100 is we thought we think 100 will be the maximum size we are going to be give the input, but that is the large enough, but suppose we are going

to input say only 5. So, that is the wastage and moreover suppose we are going to input say more than 100, at the runtime we decided our data size is more than 100, soon there is a issue. So, that is the question I mean. So, statically this has no solution I mean we cannot make it so small because in that case overflow and we cannot make it so large I mean large enough so that it will not overflow because in that case we are just wasting the memory. So, this it has no static solution, but is a dynamic solution, the solution is the dynamic table. So, solution of this problem is dynamic table. So that means, whenever we need, whenever there is a overflow we allot a new memory. So, that can be done if we you see that can be done by using Malloc or Kellogg. So, the solution is dynamic table dynamic allocation.

(Refer Slide Time: 04:03)



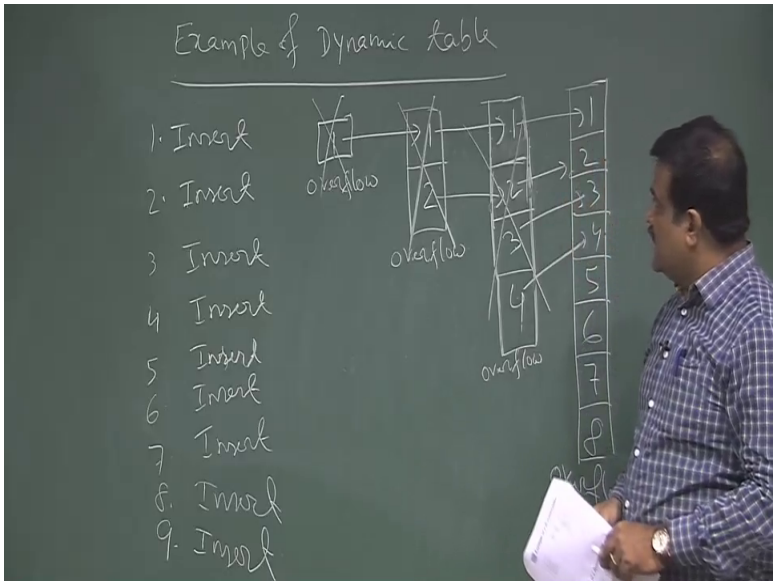
So, the idea is, whenever table overflow, we just grow the table. We grow it by allocating a new larger table, by allocating and this allocation we can do using say in C we know the Malloc or Kellogg in java I think new yeah. So, Malloc call or in java new is the call. So, we are allocating new larger table.

So, whenever that our table is having overflow then we allocate a new larger table which is larger than the current table. And then we shift all the current content to that table. This way we continue and then again we can start inserting and again if we have a overflow then again we will allot a larger table than the current table and again we move all the content to the this table like this; this way we continue. So, this is a dynamic allocation

dynamic table. So, to a new larger table and then what we do? We move all the items from old table to new table. From the old table into the new table new one and we free the storage of the old one and free the storage of storage of the old one. We have to free it otherwise it will be gangling difference. So, we have to free that storage of storage for the old table.

So, this is the idea, this is the dynamic allocation dynamic table. We want to analysis this, we want to analyze this. So, what is our insert insertion time or so let us take an example. So, let us take an example of dynamic table. So, idea is like this. So, we start with a smaller table, I mean reasonable size table then we start inserting the item or inserting the element and then when we found the overflow then we have to allot a new table which is larger than the old current one and then we shift all old item to this new table and still new table has more space to accommodate more items then we keep on insert the new items in the new table. And then again if we have a overflow in this new table we again allot a another table which is larger than the current table like this we continue.

(Refer Slide Time: 07:53)



So, let us take an example of this dynamic table strategy. So, let us take an example of dynamic table. Suppose we have a table with size 1 and suppose we insert a item. So, this is a first item and then suppose, we insert the second item. Then this is overflow? Because there is only one space, then what is our suggestion our dynamic strategy? Then

we have to allot a new table which is larger than this. So, suppose we allot a new table of size 2 now and now we have to shift the content of the old table to new table. So, we shift this 1to here and then what we do? Then we insert this item number 2 here. So, this way we continue. Now again suppose we are going to insert another item. So, again there is overflow here. So, since there is overflow then we have to allot a another table. So, another table must be larger than this table. So, our strategy is we will just have a double of this. So, this table is 1. So, double this table is 2. So now, we are going to have a table of size 4. So, this is a table of 1, 2, 3, 4. So, this is a table of so 1, 2, 3, 4. So, this is the new table we allot. So, this can be done in if we are using C language we will just use Malloc or Kellogg and if we are using java we will have to use the new to have this space.

Then what we do? Then you have to shift the old table content here and you have to free this. So, this is, this old table need to be free so that this memory space can be used later on. So, this is the, now suppose this insert we have to do. So, this is 3. So now, we are going to insert another item, insert now we can put it here. So, we are thinking that every time we have to have a new table, it is not. So now, we have a space to insert this item, but now if we are going to insert 5, number 5 item then we have the problem insert s e r t, then we have a problem because now this will be a basically overflow, this will be a overflow. So, if this is a overflow what is our strategy? Our strategy is to allot a, create a new table with the table size double of that so; that means, the table size will be just 8. So, this will be just 8; so 1, 2, 3, 4, 5, 6, 7, 8 maybe. So, let us check 1, 2, 3, 4, 5, 6, 7, 8. So, we have a new table with table size 8. Now what is our next job? Our next job is to move the content of the old table to the new table. So, we just move this content 1 here, 2 here, 3 here, 4 here and now we are going to insert this new item 5, fifth number item here, there is no issue.

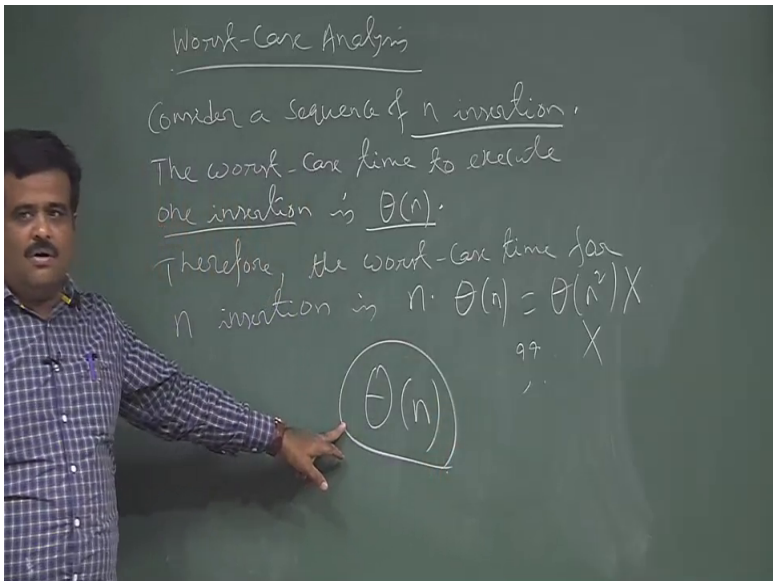
Now, we will keep on inserting the item up to say tables item number 6 we can insert. So, 6 will be inserted here even item number 7 can be insert. So, here we can keep on insert. So, up to 8 we can insert in this bigger table, but again at the ninth stage there will be overflow and once there will be overflow then we need to have a table of size we need to allot that. So, this has to be free after creating this after shifting this item. So, this has to be free, I mean free means we have to free this space.

So, that we can use this space for other purpose, so now, this is the strategy. So, after 8, 8 will be here, 8 insert is no issue there will be no overflow; so 8. So, what is the insert time? Insert time is, so for 6 just 1, for 7 if you are inserting that number 7 item 1, number 8 item 1, but what about number 9 item? If you are going to insert the number 9 item, when we are going to insert number 9 item then there will be overflow. So, once there will be overflow then, we have to have another table of double size. So, there has to be a table of size 16 and then we have to copy all these item there. So, that will that will increase our time complexity, but that is not for all the elements all the items, that is for few items. Most of the items are we are just inserting, we are not creating the table for most of the item, when n is equal to. So, we will come to that analysis for most of the item we are just having only one order of operation, but few item are creating that is for 9.

So, we have to create another table and we have to move each item there from the existing table. So, that will cost us if there are n items that will cost an order of n times.

Anyway let us analyze this method. So, what is the time complexity of this method? So, let us do a, so is this clear dynamic table? So, the idea is we just dynamically allow the, allocate the space memory. So, this is not a static allocation. So, we have to go for dynamic allocation because we do not know the size in hand, we do not know how big we should take the table size because we do not know the number of item in hand beforehand. So, static allocation is not possible in that case. So, in that case the suggestion is to go for dynamic allocation or dynamic table, but there we have a issue with the overflow. So, if the item size is more than the table size then that is called overflow. So, once there is a overflow then you have to create a larger table and once you have a larger table you have to move the all existing item on the old table to the largest table and then we have to free this old table.

(Refer Slide Time: 15:43)



So, let us talk about worst case analysis of this. So, worst case analysis of this dynamic table. So, consider a sequence of n insertion, suppose there are n items at some point of time. So, that means, we have n insertion. So, consider a sequence of n insertion. So, in that case, in the worst case for some of the, so what is the worst case time to execute? Worst case time to execute one insertion is order of n. Why? Because if there is a overflow, if suppose there is order of n items suppose n by 2 items are there. Now suppose there is a overflow then we have to shift all the n by 2 items into the new table. So, that will cost us order of n. So, that is why the worst case time for to execute one insertion will be order of n because in the worst case we have to copy or we have to create a table and you have to copy all the existing item to the new table. So, that will cost us order of n times if the size of that old table is order of n.

So, from here can you write this? Therefore, the worst case this is for one insertion, one insertion cost us this therefore, can we write this? Worst case time for n insertion because there are n insertion, is n into theta. This is going to be a theta of n square is this correct? This analysis. So, is the question is clear we are in the worst case we are assuming that our when we are going to insert an element we are assuming we want to have a new table. So, we are creating a new table. So, once you have a mew table then the old item has to be moved into the new table. So that means, all the old item, suppose there are order of n old item that has to be moved into the new table. So, that will cost us order of

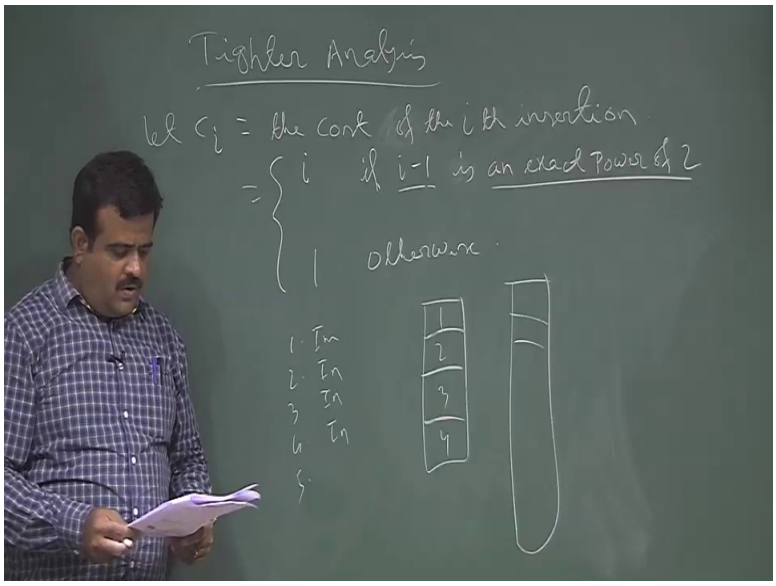
n times. So, there are n such insertion. So, that will cost us order of insert, this is it correct.

So, just simple I mean school level mathematics if we theta n for one execution then there are n execution so n into theta n. So, theta of n square; no, this is wrong. Why? Because these analysis, we are assuming that every insertion is sort of uniform every insertion is taking the worst case time. It is not because see some of the items are inserting directly, there is no overflow. Only we have to work more if there is a overflow otherwise, we are peaceful I mean we do not have to worry about that, we just insert the item there if there is no overflow. So, only thing was there is overflow then only we have to it, then only there will be more work otherwise other time there will be less work. Just order of 1, only the if there is a overflow then we have to create a new table and we have to shift the all the old item from this old table to the new table. So, that will cost us theta of n times, but that is not for all the items. Most of the items will be fitted by not creating the overflow, when the item will create overflow then only we have to have more work.

So, that analysis we have to do and that is called amortized analysis because it is average analysis; on an average, but there is no probability I mean it is a average analysis on an average. What is our insertion time? That we want to see and that will be going to the linear time, if there is n insertion then we can prove this will be this is not order of n square this will be order of n. And this analysis is called amotile, we will come to that. So, let us analysis this why it is order of n?

So, this is little tighter analysis. So, this is also a worst case analysis, but it is little tighter a little more careful way like we are not. So, when there is a overflow then only we have to do the more work otherwise here life is not that much tough.

(Refer Slide Time: 21:07)



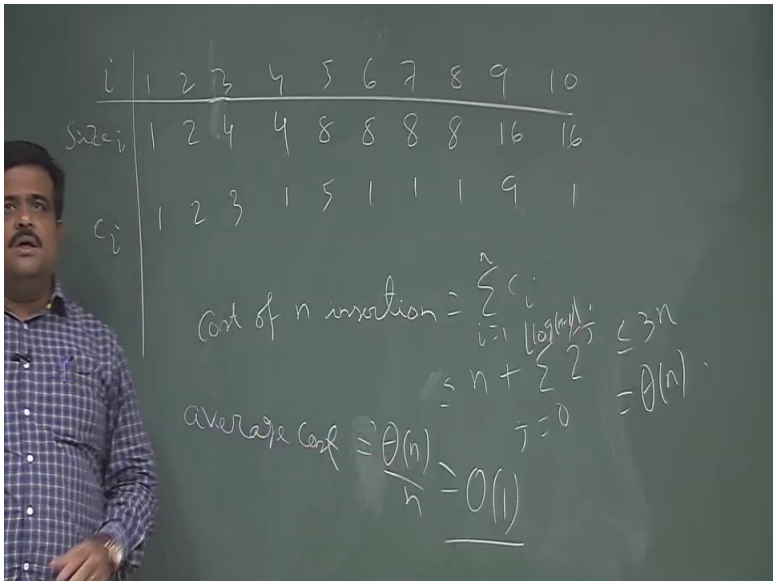
So, g h; tighter analysis, so if we denote the C i as the cost for the i'th insertion. Let C i is the cost of the i'th insertion then, C i is basically. So, for example, if i is 1 then, we have one insertion, if i is 2 then we have. So, C i is basically I, if i minus 1 is an exact power of two power of 2 otherwise, it is just 1. For example, if we just recall this table of size 4; suppose you have a table of size 4, we have 1 insert, we have 2 insert, we have 3 insert and we have 4 insert then, once you have 5 insert. So, up to 1 insert, 2 insert, 3 insert, 4 insert, there is no issue. So, for these are the insert if i is C i is basically just a 1, just for this 3, but now suppose the our table size is this now. If we have to insert 5 then, it is overflow once there is overflow then we have to work more. Then what is the strategy? Strategy is we have to make a table size double 8 and we have to shift all the elements over here. So, there are i elements over here. So, it is order of i basically it should be i minus 1.

So, 5 is basically i. So, 5 minus 1; 4 this is a exact power of 2 then, there is a overflow. So, similarly for 9, if we have a table of size 8 then we go. So, after 8 table we go up to 8 no problem then, once we reach to the nine then we have a issue. What is the issue? Then 9 is basically i is 9, so So, i minus 1 is power of 2, 2 to the power 3 basically, i minus 1 is 8 then there is a overflow. So, for those C i's for those i which is basically i minus 1 is a power of 2 then we have a overflow. Once you have overflow then we have to have a new table and then you have to copy this old content to the new table. So, that is will cost

us order of i whole item has to copy there including the new item. So, that will be the order of i. So, this is the cost for this is the exact cost for C I; the cost for the insertion.

So, if we just write it into the table. So, not for all i we are doing the we are copying. We are copying when the i is i minus 1 is the power when there is a overflow; that means, i minus 1 is the power of 2.

(Refer Slide Time: 24:36)



So, let us have the table. So, this is i. So, i is starting from 1, 2 sorry 3, 4, 5, 6, 7, 8, 9, 10 say then, this is the size of I, size of the table and this is the C i. So, if i is 1, table size is 1, if i is 2 table size is 3 then, if i is 3 there is overflow. So, you have to have a table size of double. So, it is 4; table size is 4 and similarly 4. Now once we come to the 5 then, there is overflow. So, table size will be 8. So, table size is 8, 8 table size is 8 up to this and then again you have overflow table size will be 16, 16 like this and C i will be 1, 2, 3 this is the time complexity basically the time for insert ith item, C i; 3, 1, 5, 1, 1, 1, 9, 1.

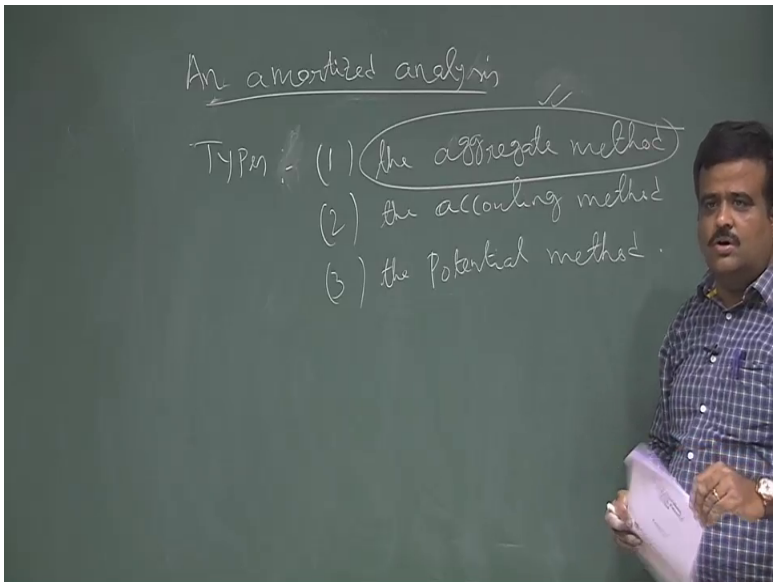
See in the most of the cases it is 1, but in the few cases where the power i minus 1 is a power of 2 then, we have to copy then we have to have a new table and we have to copy every item there. So, that is the basically. So, if we add this half. So, cost of insertion. So, if cost of n insertion, so it is not for theta n for all is basically summation of C i's, i is equal to 1 to n. Now some of the C i's are 1. So, it is basically less than equal n plus summation of 2 to the power J and J is varying from 0 to log of n minus 1 universally. So, basically why it is two to the power J? Because for this case we are just doing, so for

i is equal to 5, for i is equal to 5 we are just doing 4 times I mean 5 times. So, n is 1 time is going to n. So, this is the way, but these are from 0 to log of that not for all. For these values we are just having this huge cost otherwise every time here 1. So, that is why it is called it is not it is called amortize analysis because it is not uniform it is not same, for all the cases for some values for some i it is just a theta 1 that is why this, but there is some i for which we have more cost. So, we want to average out it. So, that is the amortized way we do. So, this is the, this thing and this if we calculate this will be less than this. So, this is basically theta of n.

So, this is basically theta of n, cost for n insertion so; that means, what is the cost for that the average cost? Average cost for one insertion, cost of average cost is basically theta of n by n. So, this is basically theta of 1. So, on an average cost is theta 1, although the n insertion cost is theta again. So, this is not probabilistic analysis there is no probability here. So, this is the exact analysis we did, but this is basically. So, the amortized analysis is a strategy for. So, let us write this. So, this is clear no? So, the average cost is. So, to insert one item average cost is theta 1 because most of the cases basically is with theta one, but some of the cases it is basically we need to spend more time because we have to copy the old item previous table to the new table. So, that will cost us more, but that is for few i's, not for all i. So, so that is the amortized analysis.

So, let us just write it. So, an amortized analysis amortized analysis.

(Refer Slide Time: 29:13)



Analysis is a strategy for analyzing a sequence of operation to show that average cost per operation is small even though the single operation within the sequence may have might have the expensive because some of the i's has the expensive cost. So, this is an amortized analysis. So, there are three types of an amortized analysis. Types of amortized analysis; one is aggregate method which we have discuss just now. Aggregate method and second one is accounting method accounting method and the third one is the potential method. So, the analysis we have discussed for our dynamic table is the method of aggregation. So, we are averaging out. So, this is slightly different from the normal type of analysis we usually discuss because here we are averaging where, in a sequence of execution we are averaging average cost is less although few items cost is expensive. So, this type of analysis is called amortized analysis.

Thank you.