

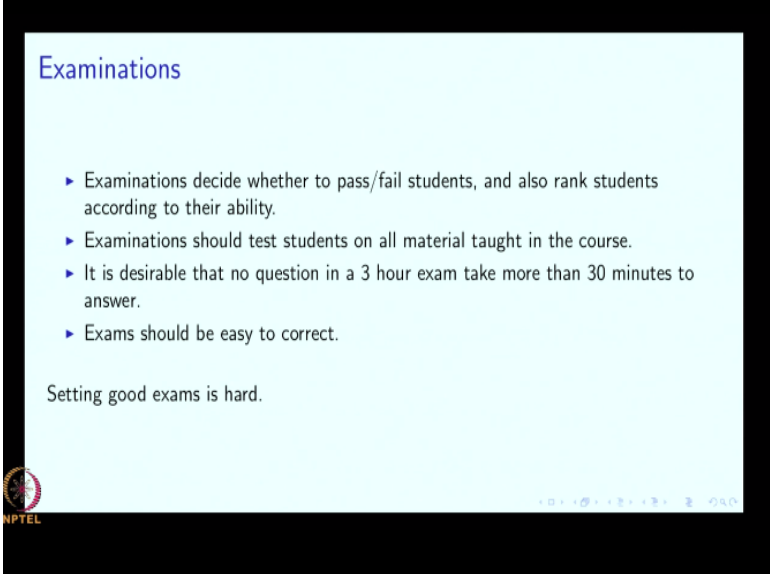
Design and Pedagogy of The Introductory Programming Course
Prof. Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology – Bombay

Lecture – 21

In Class Questions, Assignments, Examinations.1: Examinations

Hello welcome back. The topic for this sub-lecture is examinations.

(Refer Slide Time: 00:25)



The slide is titled "Examinations" in blue text. It contains four bullet points: "Examinations decide whether to pass/fail students, and also rank students according to their ability.", "Examinations should test students on all material taught in the course.", "It is desirable that no question in a 3 hour exam take more than 30 minutes to answer.", and "Exams should be easy to correct." Below the bullet points, it says "Setting good exams is hard." The slide has a light blue background and a black border. In the bottom left corner, there is a small NPTEL logo. In the bottom right corner, there are small navigation icons.

So examinations decide whether to pass/fail students and also ranks students according to their ability. Examinations are expected to test students on all material taught in the course. It is desirable that no question in a 3-hour exam take more than 30 minutes okay because a long one-hour question is really, really frightening to most students and if they get that right they have lost one third of the credit.

So that is probably not a good idea. Exams should be easy to correct and as you can see setting good exams is hard. So we do need to think about this.

(Refer Slide Time: 01:12)

Types of examination questions

- ▶ Program design
 - ▶ Write a program given a high level statement. *Stress that program*
 - ▶ Write a program given a high level statement + an invariant given as a hint.
 - ▶ Fill in the blanks in the program given below. *Easier to grade.*
- ▶ Language comprehension
 - ▶ What does the following code do? explain in English what high level function of the inputs it prints. *Tricky unless the code is very nicely written.*
 - ▶ What does the following code print?
 - ▶ "Name the looping constructs of C++". *"Rote learning". Not recommended.*

Program design questions are most important because that is finally needed.
 Language comprehension, at least "what is printed", are easier.
Should be asked; someone might not know design but may know language.

NPTEL

You can have many types of examination questions, so the most important question I guess is program design. Write a program to do whatever, whatever, whatever, so you give a high level statement and you ask the student to write a program. The high level statement is preferably in plain English or maybe depending upon the specific problem but you should try to use as few technical words as possible.

Because you really also want to test whether students can understand problem statement in high level English. Now when the students give the solution, they should not just give code, they should also explain why their code is correct. So they should write comments which include their plan and the plan should be quite precise. So the plan should state what values the variables in the program are expected to take in each iteration something like that.

So stress very, very continuously that it is not enough to write just a program but the program has to be explained. In fact, you could also go further and say that if you do not have the time to write the program but if you can just explain what each iteration is going to do, you will get some marks okay. Another possibility is to say write a program and I am giving you high level statement.

But I am also giving you a hint, so I am telling you that look you should use this invariant. Then, you can have fill in the blanks like questions where you are giving a program and then you say okay this program is supposed to do this fill in the blanks. Now sometimes a fill in the blanks question is easier to grade because the students do not have that much freedom and so we can see at a glance very quickly you can see whether the answer is correct or not.

Otherwise if you ask the student to write the entire program, it can be a long program, different students may use different ways of solving the problem and then your grading time increases. Besides program design, you should also have some language comprehension questions. The first kind of questions is what does the following code do? Explain in English what high level function of the inputs it prints.

This is a very tricky question because it has to guess what the programmer had in mind, was the programmer trying to find say the GCD of the input numbers or was the programmer trying to find the smallest number in an array or the second smallest, so all that is a little difficult. The easier kind of question as far as language comprehension is concerned is what does the following code print?

This is very straightforward. This test whether the student understands how a computer executes. So the first statement is executed, what are the new values, if it is a repeat which statement is executed next and so on. You could also ask questions like name the looping constructs of C++. Now I would discourage such questions because they are memory based questions.

They are what is called rote learning and I would not recommend such questions. Program design questions are the most important because that is what is finally needed. Language comprehension at least what is printed are easier and you should be asking them so someone might not know design and might not be able to write a complete program but if they can figure out what a program prints, if they can if they know enough to tell you how a computer executes a program then they should get credit for it.

So definitely ask both kinds of questions, so on program design questions here are some easy problems that could be asked.

(Refer Slide Time: 05:38)

Program design questions: some easy problems

- ▶ Prepare a bill for items purchased.
- ▶ Add up marks of students and find ranks..
- ▶ Reasonably simple evolution over time
 - ▶ Bank account with deposits and withdrawals specified in time order, interest and fine calculations as per rules.
 - ▶ Hydraulics systems consisting of tanks with inlets and outlets which open and shut at given times.

Should have at least one such problem.



So you are purchasing some items, so the program reads in what items are purchased what their prices are and prepares a bill or adds up marks of students and maybe find the ranks and maybe there is some physical system which has some reasonably simple evolution okay. So maybe there is a bank account with deposits and withdrawals and interest and fine calculations as per the rules and the program has to print out say the final balance.

Or maybe there is a system of tanks and inlets and program has to printout what is the amount of water in each tank at the end something like that okay. So here the manual algorithm should be completely obvious and I think you should have at least one such problem, some are really simple problem which can be stated in English and for which there is a simple program that a student can write.

(Refer Slide Time: 06:37)

Program design questions: harder problems?

For any problem you ask you should have in mind a discovery path that the student can take to the solution.

- ▶ Can student make progress using only "common sense"?
What is common sense to teacher may not be for students..
- ▶ Have you/they solved similar problems in the class/assignments?
Would it be fair to give a hint?
- ▶ Do they know the domain, e.g. if your problem is about mechanics should you expect they will remember the equations?
Have you solved problems using those equations in class?
Would it be better to supply the equations in the paper?

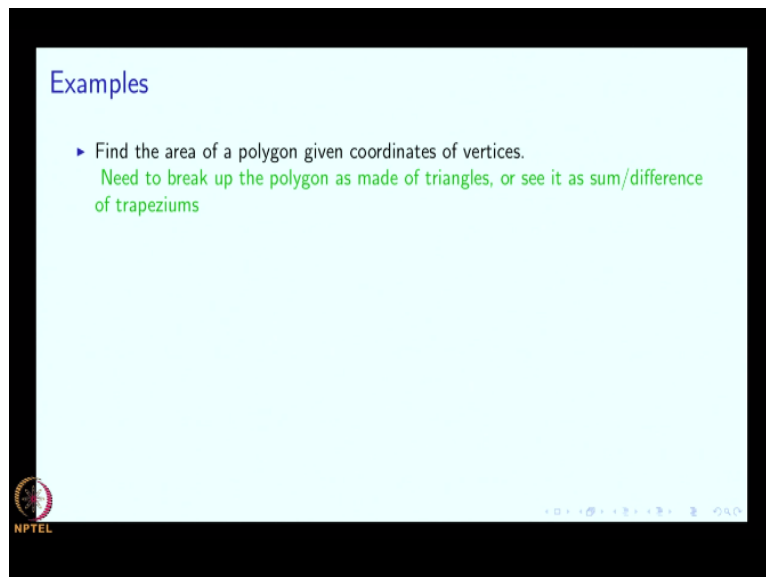


You could have some harder problems but then you have to be careful. For any problem you ask, you should have in mind a discovery path that a student can take to the solution. It should not involve flashes of insight because flashes of insight in a 3-hour exam or in a 2 hour or a 1-hour exam are really difficult okay. So it should be something which is related to what you have taught okay.

It should be based on common sense okay but if it is based on common sense you should say that it is the common sense of the student not the common sense of the teacher. Your common sense as teachers may be much more refined. So try to think like a student here. You should ask have they solved similar problems in the class, so then they know they have a hint as to how to proceed or if they have not you should ask yourself would not be fair to give a hint.

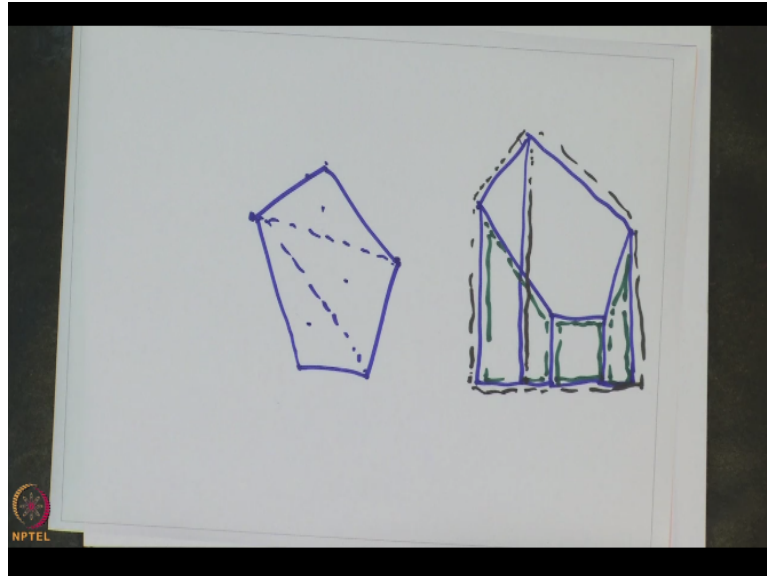
Then, do they know the domain? If the problem is about mechanics, should you expect they will remember the equations or should you just give the equations? So even if you have solved problems using the equations in class, I would say think very, very carefully about whether you want to give the equations also okay.

(Refer Slide Time: 08:00)



So some examples, find the area of a polygon given coordinates of the vertices? So what is needed over here? Well, the polygon should be broken up into triangles or somehow they should see the polygon as sum and differences of trapeziums. So let me take a picture.

(Refer Slide Time: 08:29)



So if this is my polygon, then I can break it up into triangles in this manner and often students know how to find the area of a triangle given the lengths of the sides. So if you know the coordinates, this coordinate, this coordinate, this coordinate you know the side lengths, you can calculate the side lengths so you can get the area of this triangle. Then, you can get this triangle, then you can get the area of this triangle.

So you can add up the areas and give the answer but you can also do it in a different way. So if this is our same polygon because see that look these are the areas which are below the polygon and these are the areas okay so let me use different colored pens to tell you what I mean. So this dotted trapezium this black trapezium area and this trapezium area okay-the areas of these green trapeziums okay is the area of the polygon.

So you can say this you can say that breakup the polygon into triangles or see it as sum or difference of trapeziums because after all you want to test whether we know the logic okay. If somebody gets stuck about whether they should break it up maybe you should help them out. On the other hand, if you have done, if you have stressed enough times that look you should break up you should see if we can solve smaller things and break the problem into smaller things then this is perfectly fine.

(Refer Slide Time: 10:55)

Examples

- ▶ Find the area of a polygon given coordinates of vertices.
Need to break up the polygon as made of triangles, or see it as sum/difference of trapeziums
Give hint about above?
- ▶ Given two arrays each representing coefficients of a polynomial, find the quotient.
More introspection to get index calculation
- ▶ Given a sequence of pairs (i, j) where $0 \leq i \leq 100$ is an individual and $0 \leq j \leq 10$ where i represents a person and j a club, determine if there exist some two clubs having at least 5 common members.
Deal with a lot of information
Easier if matrix used to represent membership

NPTEL

What I mean is then not giving a hint is perfectly fine, so another possibility given two arrays each representing coefficients of a polynomial find quotient polynomial. Now they know how to do this by hand but the introspection needed to figure out what kinds of array index saying which coefficients contribute where is a little tricky but this is a reasonable problem because this really just requires them to introspect, they know everything.

Here is another problem given a sequence of pairs i, j where $0 < i < 100$ is an individual and $0 < j < 10$ okay where i represents a person and j a club okay. Determine if there exist some two clubs having at least 5 common members okay. Now here the students have to deal with the lot of information but that is okay that is something that you could be testing and what is needed over here is that somehow they should represent this membership.

So if they use some kind of a matrix to represent the membership, so say put 1 in matrix element i, j if you know that you have read a pair i, j and then the question of whether some two clubs have at least 5 common members is just checking whether as you vary the j index which means if you take the sum of a row whether that row adds up to at least 5, so something like that.

But you do not have to explain it. I think students should be able to get something like this. It will require a little bit of thought and that is perfectly fine.

(Refer Slide Time: 12:35)

Example

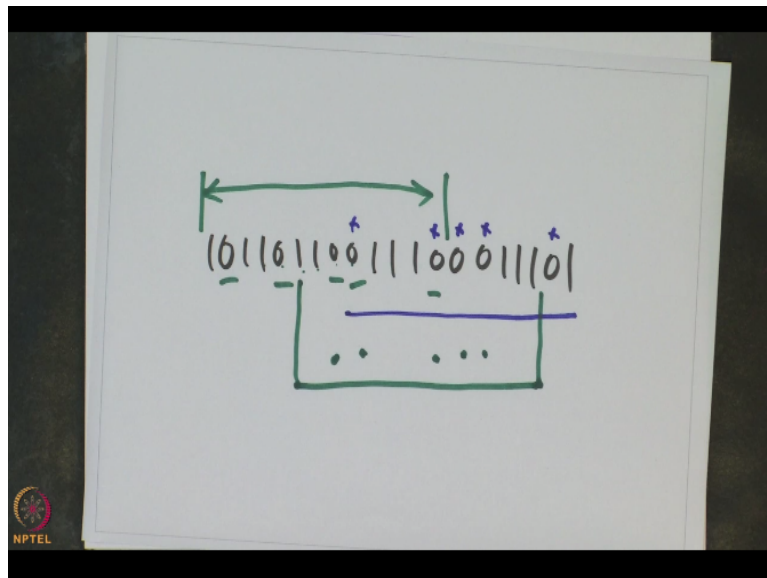
"Given an array of 0s and 1s, state how long a sequence of consecutive 1s you can make if you are allowed to change at most L 0s to 1s."



Navigation icons: back, forward, search, etc.

Here is another example, given an array of 0s and 1s, state how long a sequence of consecutive 1s can you make if you are allowed to change at most L 0s to 1s so perhaps is given example.

(Refer Slide Time: 12:56)



So here is my sequence of 0s and 1s and I am allowed to change some 0s to 1s, so say maybe I am allowed to change 5 0s to 1s okay. So maybe I change these 0s. So 4 and this 1 I change these 5 0s. Now that gives me this as a contiguous sequence of 1s, so what is the length of this sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13. So now I have a contiguous sequence of length 13 but suppose instead I change this 0 these 4 0s and this 1 0s, so what do I get?

I get this contiguous sequence of 1s okay or I could do better instead of changing this 0, I am going to change say this 0, this 0, this 0, this 0 and this 0. So now I will get a contiguous

sequence from here all the way going till here. How many 1s does this sequence have? So let us count, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 okay, so whatever so in this way you can count how many 0s each of those possible choices has and then you want to pick the one which gives you the longest consecutive sequence of 1s okay.

(Refer Slide Time: 14:41)

Example

"Given an array of 0s and 1s, state how long a sequence of consecutive 1s you can make if you are allowed to change at most L 0s to 1s."

Required thought process:

- ▶ Key decision: which 0s should you change?
- ▶ **Definition:** Two 0s are **consecutive** if there are only 1s between them.

NPTEL

Now what is the thought process required to do this? So of course the key decision is which 0s should you be changing? So for this let us make a definition so that our discussion is easier. So we will say that 2 0s are consecutive if there are only 1s in between them.

(Refer Slide Time: 15:02)

0110100111000110

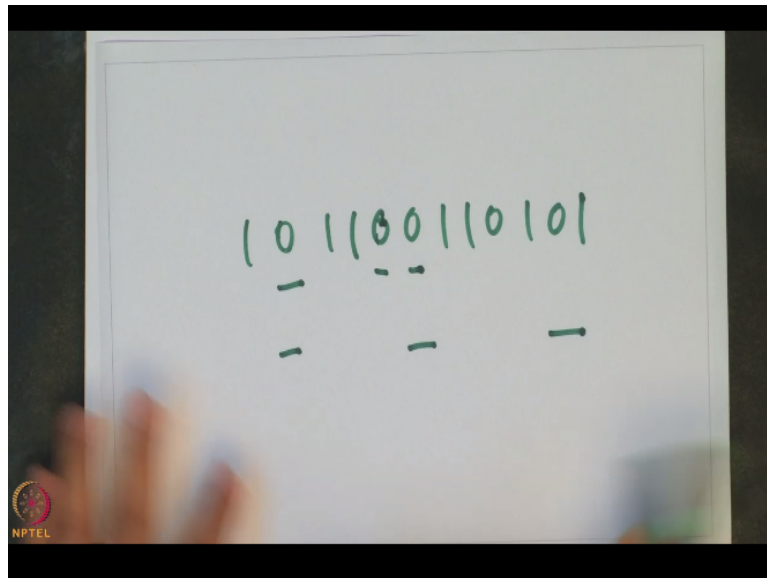
NPTEL

So for example, in this picture this 0 and this 0 are consecutive because there are only 1s in between them or this 0 and this 0 are also consecutive because there are only 1s between them but this 0 and this 0 are not consecutive because in between them there are 0s as well

okay alright. So the reason for having this consecutiveness is that whatever 0s you are changing there should be any 0s in between them that you will leave unchanged.

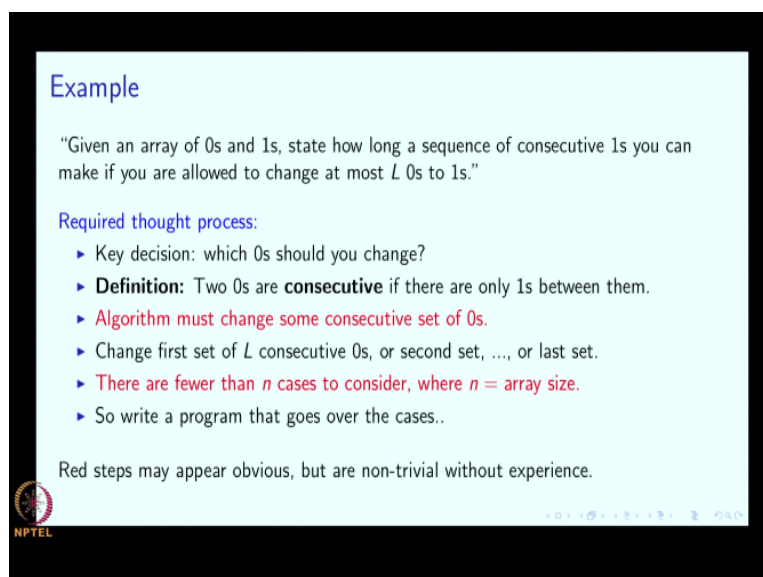
So you should really be changing sort of consecutive 0s, so what you should not do so this has become two cluttered so let me change the example.

(Refer Slide Time: 15:51)



So say my pattern is and say I am allowed to only change 2 0s or 3 0s I should be changing these 3 possibly but I should not do something like change this, this, and this because if I leave this 0 unchanged it does not really help me, either it does not help me to change this 0 or it does not help me to change this 0 okay. So somehow I should be changing 0s which are consecutive okay. So this is the first realization that you need to have okay.

(Refer Slide Time: 16:38)

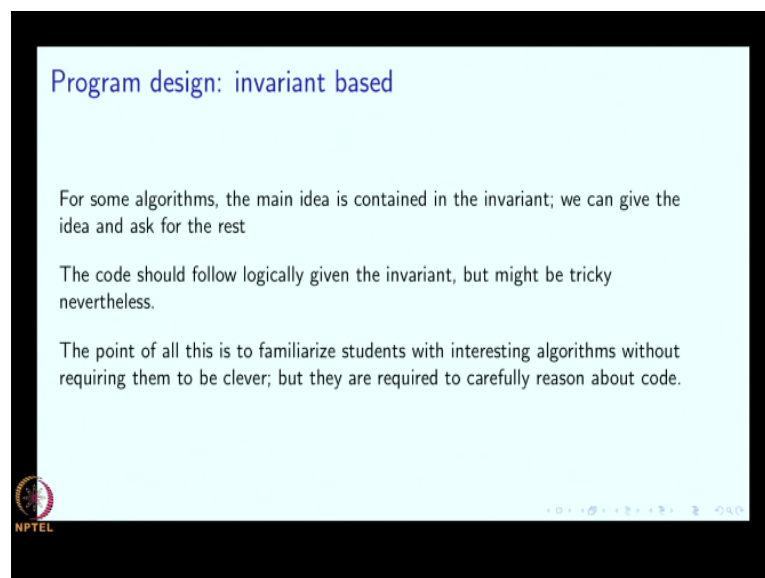
A slide with a light blue background. The title "Example" is in blue. The text reads: "Given an array of 0s and 1s, state how long a sequence of consecutive 1s you can make if you are allowed to change at most L 0s to 1s." Below this, it says "Required thought process:" followed by a list of points: "Key decision: which 0s should you change?", "Definition: Two 0s are consecutive if there are only 1s between them.", "Algorithm must change some consecutive set of 0s.", "Change first set of L consecutive 0s, or second set, ..., or last set.", "There are fewer than n cases to consider, where n = array size.", "So write a program that goes over the cases..". At the bottom, it says "Red steps may appear obvious, but are non-trivial without experience." There is an NPTEL logo in the bottom left corner and some navigation icons in the bottom right.

So now the next realization is that okay so we have some L consecutive 0s, we need to change some L consecutive 0s. So we can change the first L consecutive 0s or we can choose the second L consecutive 0s or the third L consecutive 0s and so on. So then you realize that oh there are only n cases to consider where n is the array size or at most n cases. So what we have to do is we have to write a program that goes over the cases.

Now I have seen or I know or I have been told I should say that this program was given as an exam problem but in my opinion this is a pretty difficult problem because the red steps are quite non-obvious. So unless you have students who you know are very hardworking and very sharp, I would not recommend giving a problem as harder as this even though with the few hints the students should be able to solve it.

But without hints certainly no and this is an example of where you should be careful.

(Refer Slide Time: 17:48)

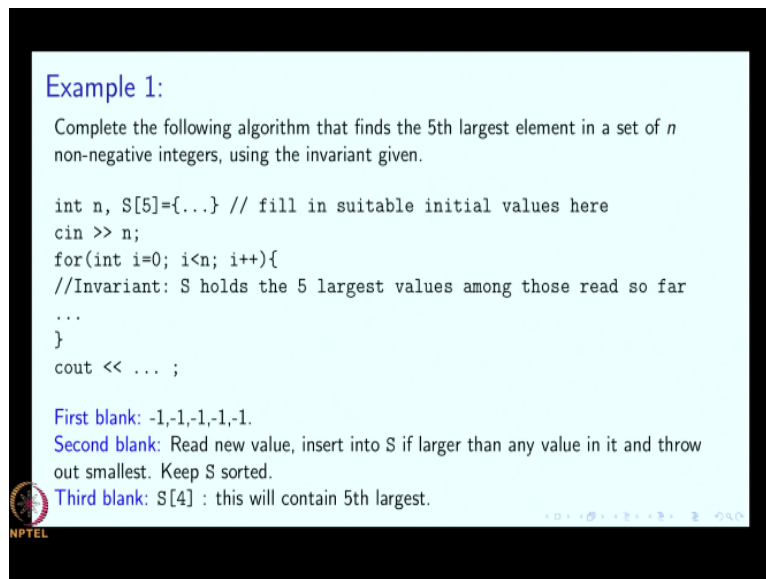


Another kind of program design question that you can ask is invariant based. So for some algorithms, the main idea is contained in the invariant and it could be an insightful idea and what we can do is we can give that idea and ask for the rest. Now the code really should follow logically given the invariant but it does not mean that writing the code is still going to be easy.

The code might still be tricky and especially in this case such problems make sense okay. So the point of giving such a question is that we can familiarize the students with interesting algorithms without requiring them to be clever but it is not still completely easy for them,

they still have to be careful and they have to sort of do work systematically to reason about what code is necessary.

(Refer Slide Time: 18:53)



Example 1:

Complete the following algorithm that finds the 5th largest element in a set of n non-negative integers, using the invariant given.

```
int n, S[5]={...} // fill in suitable initial values here
cin >> n;
for(int i=0; i<n; i++){
//Invariant: S holds the 5 largest values among those read so far
...
}
cout << ... ;
```

First blank: -1,-1,-1,-1,-1.
Second blank: Read new value, insert into S if larger than any value in it and throw out smallest. Keep S sorted.
Third blank: S[4] : this will contain 5th largest.

NPTEL

So an example, complete the following algorithm that finds the 5th largest element in a set of n non-negative integers which you are reading okay using the invariant that is given. So here is the code. So first you are going to read n which is the number of elements that you will see, you will employ an array S into which we will store 5 numbers. So some way you have to initialize that array.

So you read in that n and then you are going to read each number but you are going to somehow let S guide you, so that you will somehow keep the 5 largest values in S . So at the end you have to print something. So this is the fill in the blanks question as well as an invariant driven question. So what you are expected to do is you are supposed to fill in code where the dots are given to you.

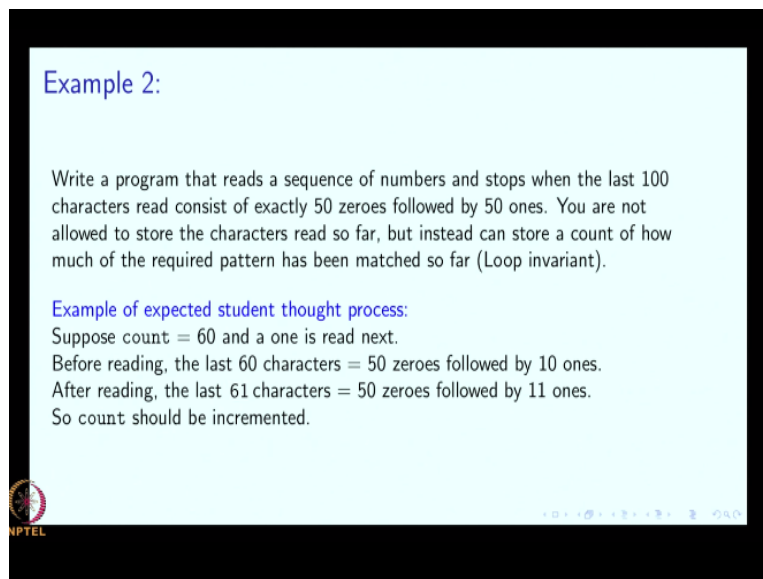
The first blank could be filled with negative numbers because then whatever the first 5 numbers are read will come in, so remember that you want that array to contain the 5 largest integers. So you have to initialize it somehow and initializing it with negative numbers makes sense. The second blank is the key so you will read in the new value and you will insert it into the set S if it is larger than any single value in it.

So you will read in and insert it and at that time throw out the smallest value from S and we will try to keep S sorted because that is how you can quickly check whether the new value

that you read is in fact larger than any of the values in that set S and finally you know that the set S will contain the 5 largest values and finally they must contain the 5 largest values in of all the numbers and therefore S of 4 can be printed out.

So as you can see we have told them the algorithm but we have not given them code, we have told them what invariant they need to satisfy. So they still need to think about exactly how to make the invariant work, so it is not the case that they just get to write code for free, this is not a completely easy question but it is not a terribly hard question either because the invariant is being told to them.

(Refer Slide Time: 21:33)



Example 2:

Write a program that reads a sequence of numbers and stops when the last 100 characters read consist of exactly 50 zeroes followed by 50 ones. You are not allowed to store the characters read so far, but instead can store a count of how much of the required pattern has been matched so far (Loop invariant).

Example of expected student thought process:
Suppose count = 60 and a one is read next.
Before reading, the last 60 characters = 50 zeroes followed by 10 ones.
After reading, the last 61 characters = 50 zeroes followed by 11 ones.
So count should be incremented.

NPTEL

Here is a slightly more difficult version of it. Write a program that reads a sequence of numbers and stops when the last 100 characters read consist of exactly 50 zeroes followed by 50 ones. You are not allowed to store the characters read so far but instead can store a count of how much of the require pattern has been matched so far and so this is what you have to use as a loop invariant.

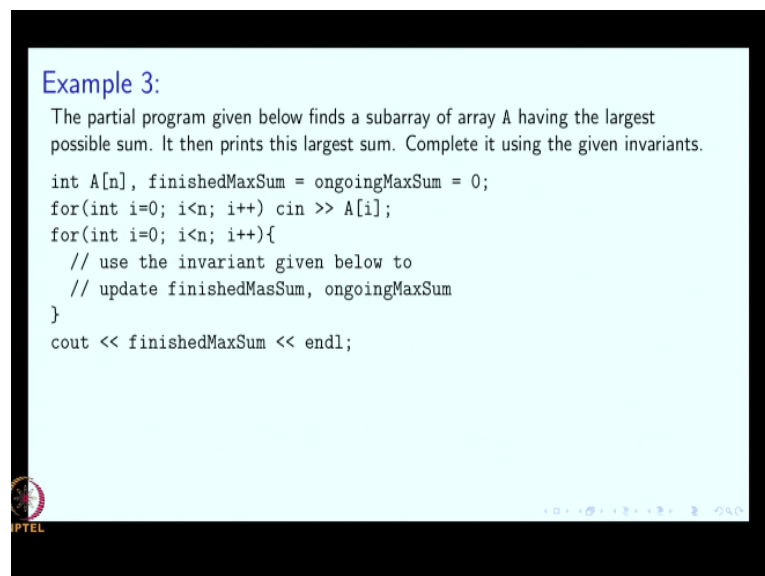
So you are allowed to store one variable called the count and the count should say how much of the pattern you have matched so far okay. So what is the thought process expected of the student? So let us say at some point the count is 60, so the student should take an example and suppose 1 is read next, so what can the student deduce at this point, well before reading because the count is 60 that means that the last 60 characters are what we want. So the last 60 characters must be 50 zeroes followed by 10 ones.

So now if you read of 1 what you know after reading, so after reading the last 61 characters should be 50 zeroes followed by 11 ones. So now 61 characters are matching the pattern, so count should be incremented. Of course if in this case a 0 was read what happens? Well the student can deduce that and that is where some amount of careful thinking is needed by the student.

Our last example is the most complicated. Here I am going to show you a problem for which there is an obvious n^2 time algorithm but there also exist a clever $O(n)$ time algorithm. Now the $O(n)$ time algorithm cannot be discovered by students because it requires an insight and it is very hard to discover especially in the tensed atmosphere of an examination.

So what you can do and which is what will happen over here is that we will give an invariant which contains the main idea. So here is our problem.

(Refer Slide Time: 23:51)



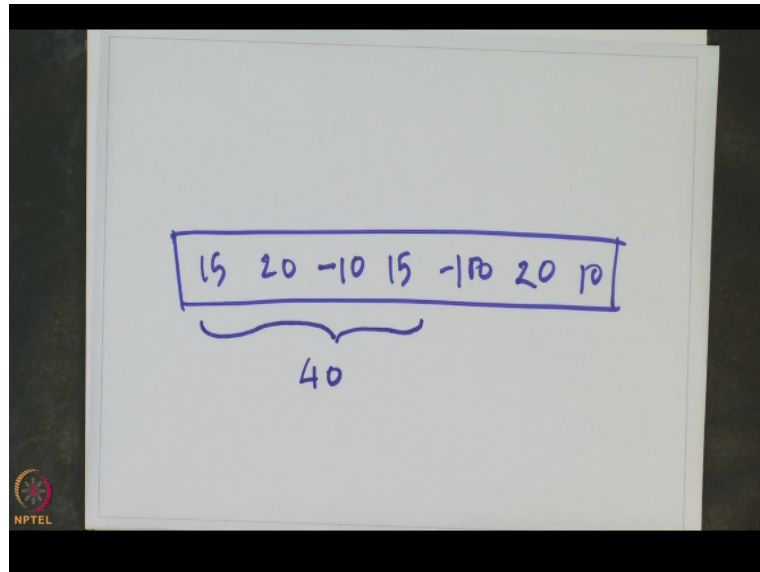
Example 3:

The partial program given below finds a subarray of array A having the largest possible sum. It then prints this largest sum. Complete it using the given invariants.

```
int A[n], finishedMaxSum = ongoingMaxSum = 0;
for(int i=0; i<n; i++) cin >> A[i];
for(int i=0; i<n; i++){
    // use the invariant given below to
    // update finishedMasSum, ongoingMaxSum
}
cout << finishedMaxSum << endl;
```

So the partial program given below finds the sub array of array A having a largest possible sum. It then prints this largest sum, complete the program using the given invariants. So before looking at a program let me tell you what the problem is. So the problem is something like this.

(Refer Slide Time: 24:15)



I have an array and that array might contain say the numbers 15, 20, -10, 15, -100, 20, 10. So I want a sub array of this such that its sum is the largest possible. So it was seen that if I take this portion including this -10 but until this point then I should get the largest possible sum. So what is the largest possible sum here? $15+15$ is $30+20$ $50-10$ is 40, so this should be the largest possible sum.

I should not include -100 because the subsequent stuff which is going to be included because of that is not enough to count at the effect of this -100. So in this case the answer should be 40 okay, so obvious algorithm to do this is to consider every possible sub array. So this array, this array, this array as well as this array, this array, this array and so on okay but this code which we have seems to give a linear time algorithm.

So let us take a look at what that code does, well it starts by defining two variables, finishedMaxSum and ongoingSum of course an addition to the array that we have. It first meets the array and then the code has a loop and you are supposed to fill in the value over here. So the value should be just updating finishedMaxSum and ongoingMaxSum. At the end, the finishedMaxSum should be printed out, so the invariants mentioned are these.

(Refer Slide Time: 25:57)

Example 3:

The partial program given below finds a subarray of array A having the largest possible sum. It then prints this largest sum. Complete it using the given invariants.

```
int A[n], finishedMaxSum = ongoingMaxSum = 0;
for(int i=0; i<n; i++) cin >> A[i];
for(int i=0; i<n; i++){
    // use the invariant given below to
    // update finishedMasSum, ongoingMaxSum
}
```

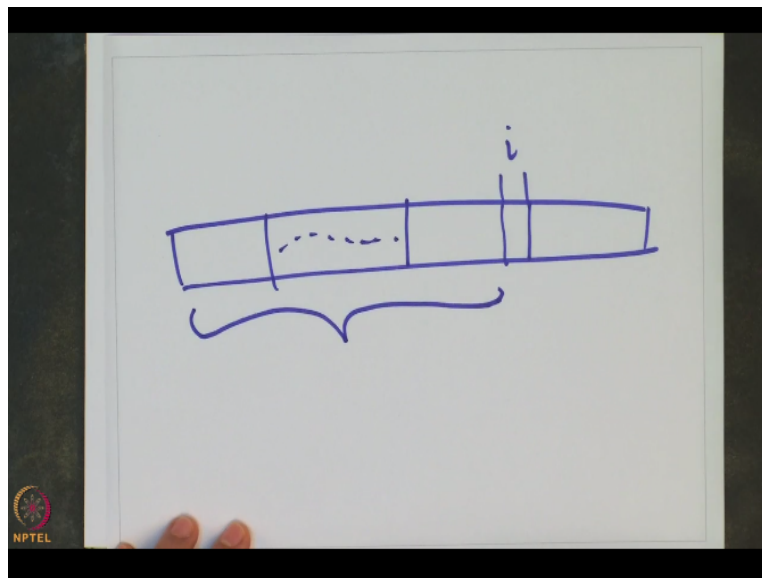
Invariant 1: finishedMaxSum = the largest sum for any subarray of A[0..i-1], starting and terminating anywhere.



Navigation icons

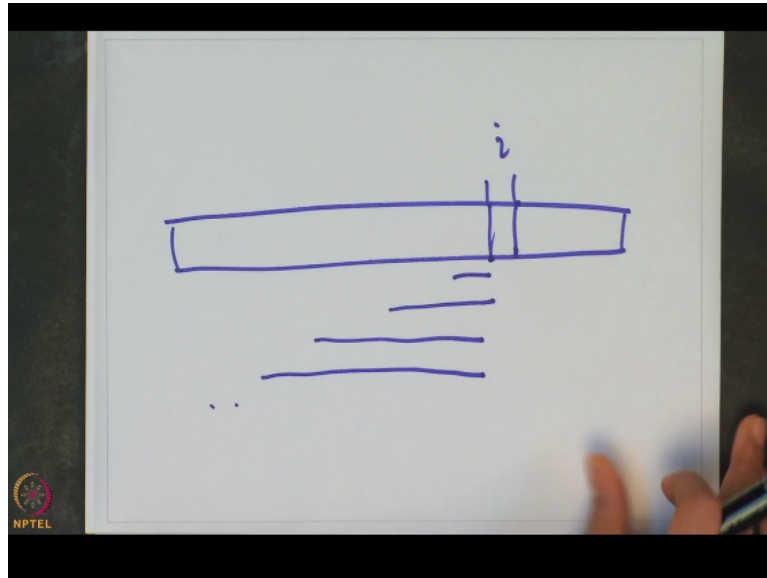
FinishedMaxSum should equal the largest sum of any subarray of the first i elements okay. They could start anywhere and terminate anywhere.

(Refer Slide Time: 26:12)



So if this is my array, this is the i index that I am currently look at, then I want a subarray of this portion, it could start anywhere end anywhere. So may be it is this such that the sum of the numbers in this region is the largest possible of all possible subarrays starting anywhere and ending anywhere. So that is as far as finishedMaxSum is concerned. OngoingMaxSum on the other hand is a little bit constrained.

(Refer Slide Time: 26:44)



So here again this is my array, this is the current value of i , i is pointing to this index, now I am looking for subarrays but I am looking for very specific kind of subarrays. So all the subarrays must terminate over here, so it could be this subarray, it could be this subarray, it could be this subarray, it could be this subarray and so on.

So they must terminate over here and of these I want the one which has the largest sum and that should be placed in `ongoingMaxSum` okay. Now at the end, we are printing `finishedMaxSum` but `ongoingMaxSum` is going to be useful in updating. So if you think about how to update this, you will see that `ongoingMaxSum` helps you in deciding what `finishedMaxSum` is going to be.

(Refer Slide Time: 27:40)

Example 3:

The partial program given below finds a subarray of array A having the largest possible sum. It then prints this largest sum. Complete it using the given invariants.

```
int A[n], finishedMaxSum = ongoingMaxSum = 0;
for(int i=0; i<n; i++) cin >> A[i];
for(int i=0; i<n; i++){
    // use the invariant given below to
    // update finishedMasSum, ongoingMaxSum
}
cout << finishedMaxSum << endl;
```

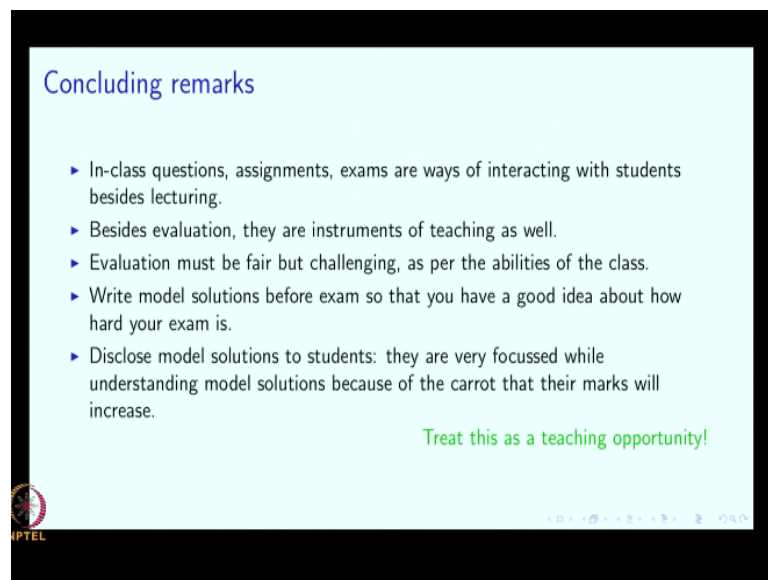
Invariant 1: `finishedMaxSum` = the largest sum for any subarray of $A[0..i-1]$, starting and terminating anywhere.

Invariant 2: `ongoingMaxSum` = the largest sum of any array starting anywhere but terminating exactly at $i-1$.

Clever algorithm: $O(n)$ time.

And at the end of course then you can print out finishedMaxSum. So as you can see this is a clever algorithm which requires $O(n)$ time because inside the loop we are only going to do two updates and there is no looping inside that. So although the student will not be able to discover this algorithm, the student should be able to write code which maintains this invariants okay.

(Refer Slide Time: 28:14)



The slide is titled "Concluding remarks" in blue text. It contains five bullet points in black text, each starting with a blue right-pointing triangle. The points discuss in-class questions, assignments, exams as teaching tools, the importance of fair but challenging evaluation, writing model solutions before exams, and the focus of students when they see model solutions. At the bottom right, there is a green text note: "Treat this as a teaching opportunity!". The slide also features a small circular logo in the bottom left corner and navigation icons in the bottom right corner.

- ▶ In-class questions, assignments, exams are ways of interacting with students besides lecturing.
- ▶ Besides evaluation, they are instruments of teaching as well.
- ▶ Evaluation must be fair but challenging, as per the abilities of the class.
- ▶ Write model solutions before exam so that you have a good idea about how hard your exam is.
- ▶ Disclose model solutions to students: they are very focussed while understanding model solutions because of the carrot that their marks will increase.

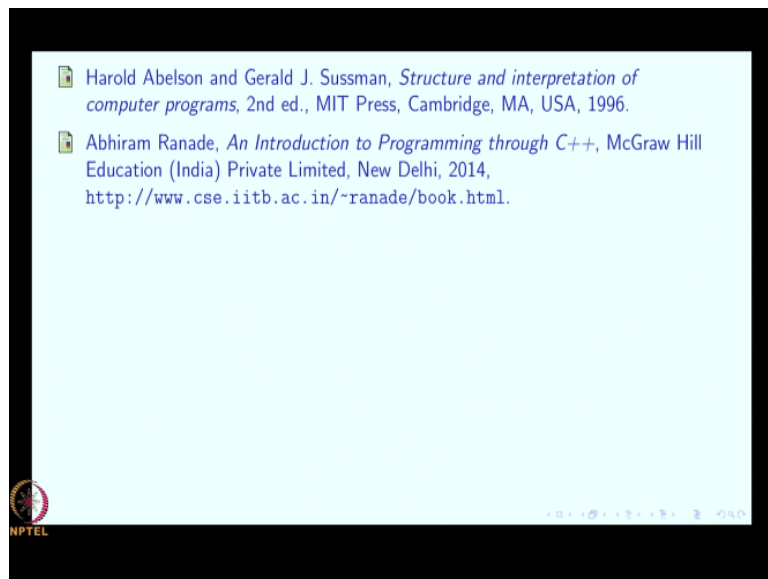
Treat this as a teaching opportunity!

So I want to now conclude the topic of examinations okay and in-class questions and assignments, the entire sequence of lectures about these topics. So in-class questions, assignments, exams are we are interacting that the students besides lecturing. Besides evaluation, they are instruments of teaching as well. Evaluations must be fair but challenging as per the abilities of the class.

You should write model solutions before exam so that you have a good idea about how hard your exam is. We should disclose model solutions to students. Note that students are going to be very focused while understanding model solutions because they want to increase their marks. So in order to increase their marks they will read the model solutions very carefully. In fact, you can say that quite possibly students will be most focused and most attentive and most eager to understand what you are telling them when they are reading the model answers.

So in fact you should treat this as a teaching opportunity, write the model answers carefully, tell them what it is that you expect them to write and tell them what their mistakes are because you may see this as a painful process of haggling for marks but it is actually also a learning process. So that concludes this topic of in-class questions, assignments.

(Refer Slide Time: 29:53)



And these are the references that we have used in this sequence of lectures.