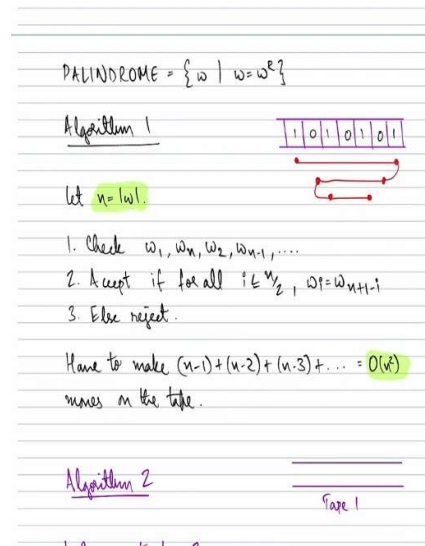


**Theory of Computation**  
**Professor Subrahmanyam Kalyanasundaaram**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**  
**Time Complexity - Part 2**

(Refer Slide Time: 00:16)



PALINDROME =  $\{w \mid w = w^R\}$

Algorithm 1

let  $n = |w|$ .

1. Check  $w_1, w_n, w_2, w_{n-1}, \dots$

2. Accept if for all  $i \in \{1, 2, \dots, \lfloor n/2 \rfloor\}$ ,  $w_i = w_{n+1-i}$

3. Else reject.

Have to make  $(n-1) + (n-2) + (n-3) + \dots = O(n^2)$  moves on the tape.

Algorithm 2

Tape 1



Let me explain couple of approaches for recognizing the language or deciding the language Palindrome. So, Palindrome is the class of strings that read the same way from left to right, the word level for instance, the word noon, it reads the same way back and forth. The word madam, for instance. So, these are palindromes.

However, some other word like friend is not a palindrome. Local is not a palindrome. So, how would we check if a given string is a palindrome? So, suppose the string was written in the input tape, so, maybe I will just write something. So, let us say 1010101. Obviously, this is a palindrome. So, what you would do is, suppose there are  $n$  symbols, let us say the length of  $w$  is  $n$ . Then what would you do?

You want to, so, to check that this is a palindrome, what you can do is the following, you will read the first symbol, and then go to the last symbol and see whether they are the same. If they are not the same, you can immediately reject. If they are the same, now, you can look at the second last symbol, and then come to the second symbol from the from the left point and check whether they are the same, if they are the same, then you continue.

If they are not the same, you reject. So, now, you check the third symbol, and the third last symbol, and then you keep doing this. So, basically you check first and last, second last and

the second, third last and the third and so on. If at any point there is a mismatch, you reject, otherwise you come to the end, and then you accept. That is what we are doing

So, we check  $w_1$  which is the first symbol,  $w_n$  which is the last,  $w_2$  is the second symbol,  $n - 1$  which is second last and so on. And you accept if

$$\text{for all } i \leq \frac{n}{2}, w_i = w_{n+1-i}$$

So, if the string is palindrome these two should be the same, otherwise, you complete and then you reject, otherwise you reject.

Even for one  $i$ ,  $w_i$  is not equal to  $w_{n+1-i}$  reject and how much time or how much time does it take? So, time again it is the number of steps. So, now, let us see what it takes. So, it starts here from the leftmost point and then it goes to the rightmost point. So, this  $n - 1$  steps then it comes back to the second point which is  $n - 2$  steps, then it goes to the third last which is  $n - 3$  steps and so on.

So, this takes if you add up all of this, this takes order  $n$  squared or this is something that adds up to asymptotically  $n$  squared. So, the Turing machine to do all this checking takes  $n$  squared number of steps. This means this procedure takes  $n$  squared time. So, now, this is going to be how we are going to approach things because we are going to closely measure how much time it takes.

(Refer Slide Time: 04:11)

2. Accept if for all  $i \leq \frac{n}{2}, w_i = w_{n+1-i}$   
 3. Else reject.  
 Have to make  $(n-1) + (n-2) + (n-3) + \dots = O(n^2)$  moves on the tape.

Algorithm 2

1. Copy  $w$  to tape 2.  $\rightarrow O(n)$   
 2. Move head of tape 2 to the right most end.  $\rightarrow O(n)$   
 3. As head of tape 1 moves from L to R, move head of tape 2 from R to L.  $\rightarrow O(n)$   
 4. Accept if  $w = w^R$ .



2. Move head of tape 2 to the right most end.  $\rightarrow O(n)$
3. As head of tape 1 moves from L to R, move head of tape 2 from R to L.  $\rightarrow O(n)$
4. Accept if  $w = w^R$ .



In this algorithm, tape heads move  $O(n)$  steps. But we need 2 tapes.

### PALINDROME P

Theorem 7.8: let  $t(n)$  be such that  $t(n) \geq n$ .

Then every multitape TM running in  $t(n)$  time has an equivalent single tape TM running in time  $O(t(n)^2)$ .



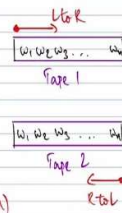
- let  $n = |w|$ .
1. Check  $w_1, w_n, w_2, w_{n-1}, \dots$
  2. Accept if for all  $i \in \frac{n}{2}, w_i = w_{n+1-i}$
  3. Else reject.



Have to make  $(n-1) + (n-2) + (n-3) + \dots = O(n^2)$  moves on the tape.

### Algorithm 2

1. Copy  $w$  to tape 2.  $\rightarrow O(n)$
2. Move head of tape 2 to the right most end.  $\rightarrow O(n)$



Now, let us look at another algorithm but we will assume that we have two tapes, maybe let us say  $w_1, w_2, w_3$  et cetera  $w_n$  is there in tape one. So, what we do is we copy this to tape 2 and then what we do is that, we scan the tape from left to right on tape 1 and we bring the tape head to the right moves in take 2 and we scan the tape right to left on tape 2.

So, it is like this, so, first we check is  $w_1$  the same as  $w_n$  because the tape 1 head will be reading  $w_1$  and tape 2 head will be reading  $w_n$ , are these two the same, if they are the same, we proceed, if they are not, we reject. And then if they are the same, the left head on tape 1 reads  $w_2$ , while that on tape 2 reads  $w_{n-1}$ .

And if that is also the same, then we go to the third symbol from the left on tape 1 and third symbol from the right on tape 2,  $w_3$  and  $w_{n-2}$  and so on. So essentially, we are checking the same things. But this approach is faster. Why is that? So, what did we do first, we first needed

to copy the content of tape 1 to tape 2, there are  $n$  symbols this takes  $n$  time, so this takes order  $n$  time. And then we move the head of the tape 2 to the rightmost end that is also order  $n$  time.

Now, we need to scan the input, the tape1 from left to right and tape2 from right to left that also takes order  $n$  time. So, every step is  $n$  time. So, the total time taken is  $O(n)$ . So, this algorithm it takes order  $n$  time or big O of  $n$  time whereas the previous one took big O of  $n$  squared time. But we needed two tapes here. So, this is more efficient, less time taken. But we needed two tapes.

So, we are actually, if you see the two tapes, correspond to using more space. So, there is some kind of a time space trade off we are using more space, but ending up using less time. So, suppose we have more space available, but we want to do it really quickly this may be better. So anyway, this also shows that palindrome is in P because  $n$  or  $n$  squared both are polynomial time.

(Refer Slide Time: 7:20)

PALINDROME

Theorem 7.8: let  $t(n)$  be such that  $t(n) \geq n$ .

Then every multitape TM running in  $t(n)$  time has an equivalent single tape TM running in time  $O((t(n))^2)$ .

Proof: (Theorem 3.13 from lecture 29)

We do a careful calculation of the approach followed in the proof of Theorem 3.13. let  $k$  be the number of tapes. We simulate the  $k$  tapes in a single tape as follows.

tape 1	#	tape 2	#	.....	#	tape k
--------	---	--------	---	-------	---	--------



Then every multtape TM running in  $t(n)$  time has an equivalent single tape TM running in time  $O(t(n)^2)$ .



Proof: (Theorem 3.13 from lecture 29)

We do a careful calculation of the approach followed in the proof of Theorem 3.13. Let  $k$  be the number of tapes. We simulate the  $k$  tapes in a single tape as follows.



- Put the tape in the above format. Initially tape 1 has the input, while



Let  $n = |w|$ .

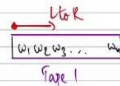


- Check  $w_1, w_n, w_2, w_{n-1}, \dots$
- Accept if for all  $i \in \mathbb{N}_2, w_i = w_{n+1-i}$
- Else reject.

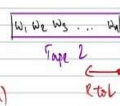
Have to make  $(n-1) + (n-2) + (n-3) + \dots = O(n^2)$  moves on the tape.



Algorithm 2



- Copy  $w$  to tape 2.  $\rightarrow O(n)$
- Move head of tape 2 to the right most end.  $\rightarrow O(n)$
- As head of tape 1 moves from L to R,

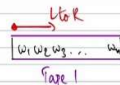


- Else reject.

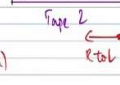
Have to make  $(n-1) + (n-2) + (n-3) + \dots = O(n^2)$  moves on the tape.



Algorithm 2



- Copy  $w$  to tape 2.  $\rightarrow O(n)$
- Move head of tape 2 to the right most end.  $\rightarrow O(n)$
- As head of tape 1 moves from L to R, move head of tape 2 from R to L.  $\rightarrow O(n)$
- Accept if  $w = w^r$ .



So, now let us see a general result. So, we saw that palindrome takes  $n$  time on a two tape machine, but  $n$  squared time on a single tape machine. Now, let us try to generalize this. So, suppose there is a Turing machine or a multitape Turing Machine, multitape deterministic Turing machine that runs in time  $t(n)$ .

But now, if you remember, when we studied multitape Turing machine and the equivalence to single tape Turing machine we saw that anything that a multitape Turing machine can do, can also be done by a single tape Turing machine. What this theorem states is that, the single tape Turing machine to decide the same language as a multitape Turing machine, takes time  $O((t(n))^2)$ .

So, if the multitape Turing machine takes  $t(n)$  time the single tape Turing machine takes only  $t(n)$  squared time, so, it takes more time, but the more time is not unbounded, it is bounded by the square of this. And so, for instance, if multitape Turing machine takes  $n$  time, which is what is happening here, the single tape Turing machine takes no more than  $n$  squared time or there is a single tape Turing machine that takes no more than  $n$  squared time.

So, let us see why that is the case. So, in fact, the proof is pretty much similar to what we saw when we saw that multitape Turing machines and single tape Turing machines are equivalent. This was theorem 3.13, which we covered in lecture 29 of our course. So, all that we are doing is it is exactly the same proof but there our goal was to just say that anything multitape Turing machine can do, single tape machine can also do.

But now that we are bothered about how many steps it takes, or the time it takes, we will have to analyze the steps it takes. So, let  $k$  be the number of tapes that the multitape Turing machine has. Now we want to simulate it in a single tape Turing machine. So, the you may recall this but nevertheless I will just briefly explain, basically the  $k$  tape contents we included in a single tape and we use this hash these delimiters, which we denote by the hash symbol.

And basically, we are virtually dividing the single tape into  $k$  tapes. And then we also have this marker for the heads. So, we will have some symbols and every virtual tape will contain one place where this marker is there, which indicates where the head is in the multitape machine. So, I am not getting into the details because if you if you forget, if you forgot what the full proof was, you can refer back to lecture 29. So, where we explain this in detail.

(Refer Slide Time: 10:43)

followed in the proof of Theorem 5.13. Let  $k$  be the number of tapes. We simulate the  $k$  tapes in a single tape as follows.



1. Put the tape in the above format. Initially tape 1 has the input, while the other tapes are empty.
2. Make 1 pass for the head locations.
3. Simulate the transition of the  $k$ -tape machine.
4. Make another pass to update.



we as a consequence of our approach followed in the proof of Theorem 5.13. Let  $k$  be the number of tapes. We simulate the  $k$  tapes in a single tape as follows.



1. Put the tape in the above format. Initially tape 1 has the input, while the other tapes are empty.
2. Make 1 pass for the head locations.
3. Simulate the transition of the  $k$ -tape machine.
4. Make another pass to update.



we as a consequence of our approach followed in the proof of Theorem 5.13. Let  $k$  be the number of tapes. We simulate the  $k$  tapes in a single tape as follows.



1. Put the tape in the above format. Initially tape 1 has the input, while the other tapes are empty.
2. Make 1 pass for the head locations.
3. Simulate the transition of the  $k$ -tape machine.
4. Make another pass to update.



So, what the single tape simulation tries to do is to, now go scan the tape from left to right. And when it scans the tape from left to it remembers where the head positions are. So, it knows where the  $k$  or it will try to see where the  $k$  head positions are. And then once it knows the  $k$  head positions, it knows what is the next step that the multitape Turing machine has to take.

So, now it has to reflect that change in the single tape, which is kind of virtually encoding the multiple tapes. So, now, to make the change, it makes another pass from left to right. So, if it needs to move the head position one place, if it needs to overwrite all that it will do in that second pass. So, basically two passes, one to scan. And then it figures out what is the next step, and then one another pass to implement the change.

And that is the rough algorithm. So, just one more point so, initially, we have the input in tape 1, followed by the other tapes being blank, which is how it should be and all the tape heads are in the left most and the tape heads will move. And the tapes will content will be written on the tapes as we go along. So, how much time does this take? So, the algorithm is exactly the same as what I had set in when we when we saw chapter 3.

But now the question is how much time does it take? So, let us see. So, for that, what is the biggest time consuming factor here. So, for each step of the multitape Turing machine, the single tape Turing machine has to do one pass to read and understand what the head positions are, and then another pass to implement the changes. So, two passes.

But two passes, how long? There are  $k$  tapes. However long these  $k$  tapes are, how long can the tape  $k$  tapes have any content? So, if you think about it, these are tapes corresponding to the multitape machine. So, if the multitape machine runs in  $t$  time, how much can the tape get filled up. So, if there is a Turing machine, and this is the multitape machine, let us say.

So, this tape, let us say every step, it moves one-one step to the right. And then in  $t$  steps it can take it can fill up to  $t$  spots. So, which means each, but cannot do any more than that in  $t$  steps can only move, it can only move  $t$  steps. So, which means each of these, each of these virtual tapes can be up to  $t(n)$  long in the worst case, it may not even be that much because if it moves left, then it is not going to fill up  $t$ , but worst case is, each of these tapes can be  $t$  cells long.

But then we have all these hash symbols and they have  $k$  tapes. So, roughly, let us say this takes  $k * t(n)$  space because of the  $k$  tapes. So, I am just ignoring the hash because it is not really going to contribute a significant amount. So, the tape is  $k * t(n)$  long. So, maybe I will just write that and to make two passes, it takes  $2 * k * t(n)$



(Refer Slide Time: 14:37)

3. Simulate the transition of the  $k$ -tape machine.  
4. Make another pass to update.

For each step of the  $k$ -tape machine, the single tape machine needs to make 2 passes - this takes  $2 * k * t(n)$  time. The  $k$ -tape machine needs to make  $t(n)$  steps.

Total time needed for the single tape machine  $\left. \begin{array}{l} \\ \end{array} \right\} = 2 * k * t(n) * t(n) = O((t(n))^2)$

Class P:  $n^k$  for some constant  $k$ .



Theorem 7.8: let  $t(n)$  be such that  $t(n) \geq n$ .  
Then every multitape TM running in  $t(n)$  time has an equivalent single tape TM running in time  $O((t(n))^2)$ .

Proof: (Theorem 3.13 from lecture 29)  
We do a careful calculation of the approach followed in the proof of Theorem 3.13. let  $k$  be the number of tapes. We simulate the  $k$  tapes in a single tape as follows.



So, that it to make two passes. And all of this is to make just one pass of the single or one step of the multitape Turing machine. So, when the multitape machine takes one step, the single tape Turing machine has to make  $2 * k * t(n)$  steps. But then how many steps does the multitape Turing Machine take?

So, by the assumption multitape Turing machine runs in time  $t(n)$  by assumption, this is the time it takes. To simulate  $t(n)$  steps of the computation of the multitape Turing machine, we need to multiply  $t(n)$  with the time it takes to accomplish or execute each step which is  $2 * k * t(n)$ . So,  $t(n)$  number of steps of the multitape Turing machine and the time it takes to execute each step is  $2 * k * t(n)$ .

So, altogether multiplying we get  $2 * k * (t(n))^2$ . 2 and k are constants, 2 is a constant, k is the number of tapes which is also a constant. So, the total running time is  $O((t(n))^2)$  which is what we claim here. So, all that says that if the running time of the multitape Turing machine is  $t(n)$ , then the single tape simulation takes at most  $(t(n))^2$  time asymptotically which kind of matches what happened in the case of palindrome?

In a way, it is good to know but it is not like the multitape Turing machine has k tapes, it is not  $(t(n))^k$  for instance. So, no matter how many tapes it takes, this is still squared, this is not bigger exponent. Of course, k features here as a constant in the O notation, but that is absorbed by the O notation, but if it featured here it would have been worse, but this dependence is something that is slightly less.

(Refer Slide Time: 16:43)

tape machine needs to make 2 passes - this takes  $2 * k t(n)$  time. The k-tape machine needs to make  $t(n)$  steps.

Total time needed for the single tape machine } =  $2 k t(n) + t(n)$   
 $= O((t(n))^2)$

Class P:  $n^k$  for some constant k.

Polynomial :  $O(n^k)$   
 Exponential :  $O(k^n)$

Examples:



Anyway, just to remind again, class P is a class of problems whose running time is  $n^k$  for some constant k in a deterministic Turing machine and always polynomial running time is considered to be more efficient than an exponential running time. Meaning whenever we have a choice between  $n^k$  and  $k^n$ ,  $n^k$  is much slower growing so, much less time than exponential.

(Refer Slide Time: 17:18)


Examples

(1) CONNECTED: Given a graph  $G$ , is it connected? EP.

Do BFS/DFS on the graph.

(2) 3-COLORABLE: Given a graph  $G$ , can we color it using 3 colors? (with each adjacent pair of vertices receiving distinct colors)

Not known to be in P




(3) RELPRIME: Given integers  $x, y$ , are they relatively prime?



Do BFS/DFS on the graph.

(2) 3-COLORABLE: Given a graph  $G$ , can we color it using 3 colors? (with each adjacent pair of vertices receiving distinct colors)

Not known to be in P



(3) RELPRIME: Given integers  $x, y$ , are they relatively prime? EP.  
 $\text{Is } \text{gcd}(x, y) = 1?$

(4) PRIME: Given integer  $N$ , is it prime? EP  
 $\sim \sqrt{N}$

(5)  $A \times B$  for two  $n \times n$  matrices.

(6) TSP: Travelling Salesman Problem.



So, now, I want to just go through some problems, I will not explain them to great detail, but some problems and say how much time it takes for each of them. So, one problem is given a graph is it connected. So, the answer is this is polynomial time this is in P you can do BFS or DFS. So, do DFS or BFS. So, this is in P. Because given the graph, breadth first search or depth first search traversal algorithm is something known and that tells you whether the graph is connected or not.

The other problem is whether the graph is three colorable, three colorable means can a graph be colored using three colors such that no two neighboring vertices get the same color. So, this small graph that I have drawn cannot be colored using three colors, whereas this graph that I am drawing, it can be colored using three colors.

So, given a graph, can this be colored using three colors and this is as of now not known to be in P. Of course, you can try all possible three colorings, but that takes exponential time. If the graph has  $n$  vertices, to try out all possible colorings using three colors, it takes  $3^n$  time, which is not polynomial. There is no known polynomial time algorithm for this problem. Relatively prime, given two integers  $x$  and  $y$  are they relatively prime.

So, meaning is the highest common factor or the highest common divisor great or sometimes known as GCD. So, in other words, is it like we are asking is GCD of  $x, y$  equal to 1. This will happen only when they are relatively prime, is there any common factors. In fact, this is known to be in P. There is something called Euclidean algorithm that tells you how to do this, basically divide the bigger number with the smaller so if  $x$  is bigger and  $y$  is smaller, you divide  $x$  by  $y$ .

And then, if  $y$  is a divisor of  $x$ , then they are not related with the prime because, assuming  $y$  is greater than 1, if  $y$  does not cleanly divide  $x$ , then you take the remainder and then repeat, so I do not want to get into details, but there is a clean algorithm for this. So, next question is given a number  $N$ , is  $N$  prime. This is also a polynomial time problem, there is a polynomial time algorithm for it.

However, this was some problem that kind of confounded computer scientists for a long time. This was only discovered in 2002, maybe around 20 years back. So, one may wonder what is the big deal here, like, given a number you want to test whether it is prime, can you just not try out all possible, let us say you are given some numbers, let us say 241, you try out factors at 241 is not a multiple of 2, is it a multiple of 3.

Then you see it is not a multiple of 3, then it is not a multiple of 5, you try out all possible prime factors and then you have to rule out everything till  $\sqrt{N}$  or something. However, this turns out to be something like roughly, it is enough to do up to the factors upto  $\sqrt{N}$ .

Because it has to have some non-trivial factor, which is at most  $\sqrt{N}$ , because two factor above  $\sqrt{N}$  will give some number bigger than  $N$ . But this particular approach that I suggested is not polynomial time.

(Refer Slide Time: 21:47)

(7) SUBSET-SUM: Given a set  $S$  of integers, and a target sum  $t$ , is there a subset of  $S$  that sums to  $t$ .

For PRIME, trying out all prime factors upto  $\sqrt{N}$  is NOT a polynomial approach. This is because  $\sqrt{N}$  is not polynomial in the length of the input  $\approx \log N = n$ .

$$\text{Then } \sqrt{N} = \sqrt{2^n} = 2^{n/2}$$

↑  
Exponential in  $n$ ,  
which is the length  
of the input.

is  $\text{gcd}(x, y) = 1$ !

(4) PRIME: Given integer  $N$ , is it prime? EP  
 $\sim \sqrt{N}$

(5)  $A \times B$  for two  $n \times n$  matrices. EP

(6) TSP: Travelling Salesman Problem.

(7) SUBSET-SUM: Given a set  $S$  of integers,

and a target sum  $t$ , is there a subset of  $S$  that sums to  $t$ .

For PRIME, trying out all prime factors upto  $\sqrt{N}$  is NOT a polynomial approach. This is because  $\sqrt{N}$  is not polynomial in the length of the input  $\approx \log N = n$ .

$$\text{Then } \sqrt{N} = \sqrt{2^n} = 2^{n/2}$$



Maybe I will just write here, for prime trying out all prime factors up to  $\sqrt{N}$  is not a polynomial time approach. This is because so, the problem here is that one may think that  $\sqrt{N}$  is polynomial in capital  $N$ , which is correct. The thing is that the running time should be polynomial in the length of the input, so,  $\sqrt{N}$  is not polynomial in the length of the input which is like  $\log(N)$ .

So, if you have a number, let us say 100, it takes seven bits to write down that number. So, the length of the input is not 100, the input is 100, but the length is just log of 100. So, in terms of the length of the input,  $\sqrt{N}$  is actually exponential time.

So, let us say  $\log(N)$  is equal to small  $n$ , then  $\sqrt{N}$  is like  $\sqrt{2^n}$ , which is like  $2^{\frac{n}{2}}$  or something, which is exponential  $n$ , which is the length of the input. So, when I say polynomial time every time it is in terms of the length of the input.

So this the straightforward algorithm that we are talking about that is just trying out all possible time prime factors up to square root of that number, it is a correct algorithm, but this is not a polynomial time algorithm. This is because this takes time that is exponential in the input length. But there is another approach which actually is polynomial time which I will not say, I will not explain but, it is also non trivial. After longtime people, there were many years people were looking at it.

So, the problem prime is in polynomial time. But this is this approach does not work. The next one is a computation, given two  $n \times n$  matrices, A and B, you want to multiply them, this is in polynomial time. There is a standard  $n$  cubed algorithm, but there are slightly better algorithms also.

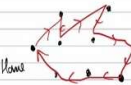
(Refer Slide Time: 25:39)

For PRIME, trying out all prime factors upto  $\sqrt{N}$  is NOT a poly-time approach. This is because  $\sqrt{N}$  is not polynomial in the length of the input  $\approx \log N = n$ .

$$\text{Then } \sqrt{N} = \sqrt{2^n} = 2^{\frac{n}{2}}$$

Exponential in  $n$ , which is the length of the input.

How




TSP



$\sim \sqrt{N}$   
 (5)  $A \times B$  for two  $n \times n$  matrices. EP  
 (6) TSP: Travelling Salesman Problem.  
 Not known to be in P  
 (7) SUBSET-SUM: Given a set  $S$  of integers,  
 and a target sum  $t$ , is there a subset  $A$   
 of  $S$  that sums to  $t$ . Not known to be in P.  
 $S = \{1, 3, 7, 10, 16\}$   
 Is there a subset that sums to  $t=12$ ?  
 For PRIME, trying out all prime factors  
 upto  $\sqrt{N}$  is NOT a polytime approach. This is  
 because  $\sqrt{N}$  is not polynomial in the length of



connected? EP.  
 Do BFS/DFS on the graph.  
 (2) 3-COLORABLE: Given a graph  $G$ ,  
 can we colour it using 3 colours? (with  
 each adjacent pair of vertices receiving distinct  
 colors)  
 Not known to be in P.   
 (3) RELPRIME: Given integers  $x, y$ , are  
 they relatively prime? EP.  
 Is  $\gcd(x, y) = 1$ ?  
 (4) PRIME: Given integer  $N$ , is it prime? EP  
 $\sim \sqrt{N}$   
 (5)  $A \times B$  for two  $n \times n$  matrices. EP  
 (6) TSP: Travelling Salesman Problem.



Finally, we have two more problems. One is TSP, traveling salesman problem. So, traveling salesman problem, maybe I will just briefly describe, it is like this, given a map and let us say you are a salesman and you are starting from some node, let us say this one, this is like home your goal is to visit all these cities. So, these dots are cities.

So, this is the traveling salesman problem, sometimes called TSP, your goal is to visit all the cities and then come back to the home. So, you have to have some tour something like this. So, this is a tour. Now, I want to do this in the most efficient manner in the shortest possible distance. So, that is a traveling salesman problem, given all the cities and the layout and the distances, what is the shortest possible way.

In fact, this is not known to be in P. Seems like a simple problem. But this is not known to be in P. And finally, one more language that I will just briefly describe. This is called SUBSET-

SUM. So, given a set, let us say, it is  $\{1, 3, 7, 10, 16\}$ . Now, I am asking, is there a subset that sums to 12? I think the answer is no.

But the question is, a subset that sums to 12. So, maybe I will just say  $t$  equal to 12. So, given a set  $S$  and a target  $t$ , is there a subset of  $S$  that sums to  $t$ ? That is a question. And surprisingly, this also is not known to be in  $P$ . So, this is just to give some of these, I will explain why we, of course, for those that are in  $P$  the algorithm is known, for those that are not in  $P$ , it is just that we do not know.

And for some of these problems let us say traveling salesman problem, subset sum, et cetera there is some evidence as to why it may be unlikely that we may never get to know whether or it is going to be very hard to get to know whether it is in  $P$  or not. And even so forth three colorability. These are some example problems, and I am just explaining what is known about them.

(Refer Slide Time: 28:56)

Complexity Theory

- What we will see next in this course is an introduction to the area of complexity theory.
- There are several more models of computation where resources come into play. Most important resources are time, space, randomness, circuits, parallelism, interaction etc.
- The goal of complexity theory is to understand computational problems in terms of resources needed. We will see "complexity classes" which are classes based on computational problems.
- In this course, we will see two resources.
  - Time Complexity





the resources used in the computation.



Time Complexity

no. of steps

Def 7.1: Running time or time complexity of a Turing machine  $M$  is the function  $f: \mathbb{N} \rightarrow \mathbb{N}$  where  $f(n)$  is the maximum (time) taken by  $M$  to accept/reject an input of length  $n$ .

Def 7.2: Landau's  $O$ -notation.

$f = 3n^2 + 2n + 5$   
 $f = O(n^2)$   
 $f = o(n^3)$

If  $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$ , we say that

$f(n) = O(g(n))$  if  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq c$  for some constant  $c$ .

$f(n) = o(g(n))$  if  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$   
 synonyms.



Def 7.7: DTIME ( $t(n)$ ) (TIME ( $t(n)$ ) in the book) is the set of languages that are decided by a DTM in time  $O(t(n))$ . (time = no. of steps)

TM is deterministic, and may be multitape. Usually we care if  $t(n)$  is a polynomial, or not.

$n, n^2, n^3, \dots$

Def 7.12:  $P = \bigcup_{k=1}^{\infty} \text{DTIME}(n^k)$

Why should we study the class  $P$ ?

- Stands for all efficient/practical algorithms.
- A robust class, independent of model.



TM is deterministic, and may be multitape. Usually we care if  $t(n)$  is a polynomial, or not.

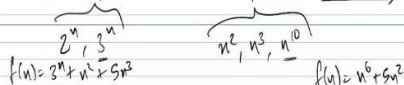
$n, n^2, n^3, \dots$

Def 7.12:  $P = \bigcup_{k=1}^{\infty} \text{DTIME}(n^k)$

Why should we study the class  $P$ ?

- Stands for all efficient/practical algorithms.
- A robust class, independent of most models of computation.

→ Exponential vs. Polynomial



Now, let me just quickly recap. So, we started the complexity theory, we said that, we are going to be considering only decidable languages and studying them based on how much resources they use. So, this chapter is about time complexity. So, where time means the number of steps taken by Turing machine to decide on a certain input.

So, we first define running time complexity, the O notation, we define  $DTIME(t(n))$ , which is a class of all computational problems that can be decided in  $O(t(n))$  time. We defined P to be the set of all computational problems that can be decided in polynomial time.

(Refer Slide Time: 29:35)

Theorem 7.8: let  $t(n)$  be such that  $t(n) \geq n$ .

Then every multitape TM running in  $t(n)$  time has an equivalent single tape TM running in time  $O(t(n)^2)$ .

Proof: (Theorem 5.13 from lecture 29)

We do a careful calculation of the approach followed in the proof of Theorem 5.13. let  $k$  be the number of tapes. We simulate the  $k$  tapes in a single tape as follows.

Examples:

(1) CONNECTED: Given a graph  $G$ , is it connected?  $\in P$ .  
Do BFS/DFS on the graph.

(2) 3-COLORABLE: Given a graph  $G$ , can we color it using 3 colors? (with each adjacent pair of vertices receiving distinct colors)  
Not known to be in P.

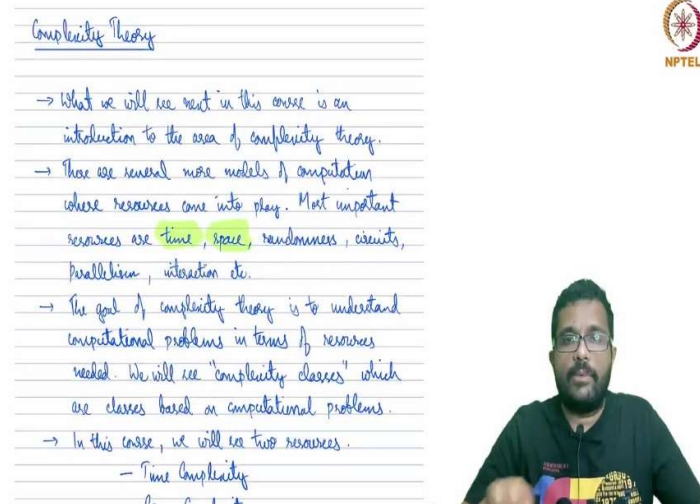
(3) RELPRIME: Given integers  $x, y$ , are they relatively prime?  $\in P$ .  
Is  $\gcd(x, y) = 1$ ?



And we explained this result that if a multitape Turing machine runs in time  $t(n)$  there is an equivalent single tape Turing machine that runs in  $(t(n))^2$ , followed by a bunch of problems

and we saw how much how we can solve them and in some cases, we explained what is known about this, whether this is known to be in P, not known to be in P and so on.

(Refer Slide Time: 30:01)



The image shows a slide with handwritten notes on a lined background. The title is "Complexity Theory". The notes are as follows:

- What we will see next in this course is an introduction to the area of complexity theory.
- There are several more models of computation where resources come into play. Most important resources are time, space, randomness, circuits, parallelism, interaction etc.
- The goal of complexity theory is to understand computational problems in terms of resources needed. We will see "complexity classes" which are classes based on computational problems.
- In this course, we will see two resources.
  - Time Complexity

There is a video inset of a man with glasses and a beard, wearing a dark shirt, speaking. In the top right corner of the slide, there is the NPTEL logo, which consists of a circular emblem with a star and the text "NPTEL" below it.

And that pretty much sums up what I wanted to say in this lecture, lecture number 45. So, in the next lecture we will see more about complexity theory. We will see more, we will see other types of other aspects of time complexity. Thank you.