VICTOR COSTAN: So I'm excited about today's recitation, because if I do this right and you guys get it, then I can mess up every other recitation after it. And you'll still get the gist of 6.006. So all I have to do is get this working. So most of the time in the real world you're probably not going to be coming up with new algorithms to do something, but rather you'll have some code and you want to make it faster. And the first step in making it faster is you realize, how does it do right now? How does it run, which lines are slow, which lines are fast, and where you can make improvements.

So in lecture we talked about the Python Cost Model which is what you use to look at the code and figure out how much time it takes to run. And we talked about document distance, which is a problem that we'll use to practice our analysis skills. And this entire recitation is all about looking at versions of document distance and analyzing them. So that's what we'll do, look at Python code, look at Python code, look at Python code. So you better have handouts, because I can't project.

OK, how many people remember the document distance problem? You guys said you went to lecture, right? OK, so very, very fast, document distance. I have two documents. The fox is in the hat. And the fox is outside. Document 1, document 2. What's the first thing I want to do? So there are three operations that Eric mentioned in lecture. Operation one, take each document, break it up into words. Right? This is a string. When I read it, then it becomes word one, word two, word three, word four, so on and so forth.

Operation two, build document vectors out of the two documents. So the documents are D1 and D2. A document vector is basically a list of the words in the documents with a count of how many times each word appears in the document. So let's build a document vector for document one. I'm not going to write it formally, so can anyone tell me what it should look like, and I'll sort of write it down as a list. So for all the words here, I want to list the words and how many times they show up. Somebody, please.

AUDIENCE: The is in there twice?

VICTOR COSTAN: OK. The, twice.

AUDIENCE: Fox, once.

VICTOR COSTAN: One.

AUDIENCE: Is, once.

VICTOR COSTAN: Is, one.

AUDIENCE: [INAUDIBLE] in once.

VICTOR COSTAN: In, one.

AUDIENCE: Hat once.

VICTOR COSTAN: Awesome. Thank you very much. Second one. Another volunteer. Yes, go for it.

AUDIENCE: The, once.

VICTOR COSTAN: The, one.

AUDIENCE: Fox, once.

VICTOR COSTAN: Fox, one

AUDIENCE: Is, one.

VICTOR COSTAN: Is, one.

AUDIENCE: Outside, one.

VICTOR COSTAN: Outside, one. OK, so this is a document vector. Notice two small details. Here, they is capitalized, here it's not, and yet I bundle them together. I know my grammar, so I put periods at the end of the sentences, and yet they don't show up anywhere here. So we got rid of the punctuation, and we made all words lowercase. These are details, but they are details that you'll see in the code, so if you're wondering why, this is why.

So step one, read the document, make it a list of words. Step two, compute the document vector. Step three, take the two document vectors, and compute the angle. What is the angle of two document vectors? Big ugly math formula.

The only thing that's relevant is that it takes these vectors and computes an inner product. So if we look at the code for angle vector, or vector angle, you'll see that because numerator denominator lines two and three, it calls inner product three times and then it does some math with it. We don't care about the math. We assume the math is order one. We only care about inner product. How does inner product work? Can anyone help me compute the inner product for these guys? Yes?

AUDIENCE: It's like the dot product?

VICTOR COSTAN: OK.

AUDIENCE: So, if we take the vectors and you multiply them, like, you're adding to the components, right? Because they're so thick--

VICTOR COSTAN: OK, this is too complicated, then. I'm seriously depressed, so give me some clear instructions step by step.

AUDIENCE: Like, I know you divide by the length of each of the vectors--

VICTOR COSTAN: Let's not worry about that. I have these vectors, and I want an inner product. I don't care about the angle, just the inner product.

AUDIENCE: OK, well do 2 times 1 for the right.

VICTOR COSTAN: OK. So I take the here, shows up twice. I take the here, shows up once. 2 times 1, right?

AUDIENCE: Mhm.

VICTOR COSTAN: OK. And then?

AUDIENCE: I would do the same for fox.

VICTOR COSTAN: OK, fox shows up here once, shows up here once, so what I do?

AUDIENCE: 1 times 1.

VICTOR COSTAN: OK.

AUDIENCE: And do the same for is.

VICTOR COSTAN: OK.

AUDIENCE: And in should be 0.

VICTOR COSTAN: OK.

AUDIENCE: [INAUDIBLE] in.

VICTOR COSTAN: OK.

AUDIENCE: And then outside would also be 0, and hat would also be 0.

VICTOR COSTAN: OK. So it turns out you don't have to go through both lists. It's sufficient to go through one of the vectors and look up the words in the other vector. Because if the words don't show up in any of the vectors, their contribution is going to be 0. So my algorithm is go through each of the elements here, look up each of the words there, look up at the word here. And if there's a word here and here, take out the number of times it shows up in each document, multiply them, and then add everything up.

So this is inner product. Everything else is good if you're writing a search engine or if you're using the scenario application, but we're not really concerned with it. OK, so now we have the three steps, read the document, break it up into words, compute document vectors, compute our inner product. So this is what we want to do. And document distance 1 does it in a painfully slow way, and we're probably not going to cover everything in recitation. But if you go all the way up to document distance 1, that's really, really fast. It's 1,000 times faster. So this is our job for the day.

Let's look at the code. Did anyone look at the code beforehand? Nope. OK, so when I look at a big piece of code, I like to look at it from top down. So that means I start to the main function, I see who is it calling, I see what everything is trying to do, and then I go into the sub functions and recurves and basically do the same thing. So I build a tree of who's calling what, and that helps me figure out what's going on.

So let's start with main. And let's look at main. Lines 1 through 6 look at the arguments. We don't really care. Line 7 and 8 call word frequencies for file. I am abbreviating liberally. And then line 9 calls vector angle. So line 7 and 8 read the two documents, do steps one and two, and then 9 does step three.

OK. Word frequencies for files. So the point of this is to read a file and to produce a word document vector out of it. And it does it in three steps. Reads the file, line two. Breaks up the file into words, so operation one, this is line 3, and then line 4, it takes up the list of words and computes a document vector out of it. I don't care about reading files because I'll assume this is somehow done for me. We care about the algorithms. So as far as I'm concerned, this function is calling get words from line list. Get words from line list, and count frequency.

And if we skip all the way to vector angle-- we already talked a little bit about how all it does is it calls inner product three times and then in does some fancy math of it. So this is how the code looks like big picture. OK, so to figure out the running time for main, we need to figure out the running time for these two functions and add them up, right? To figure out the running time for this, we need to figure out the running time for these functions and add them up, so on and so forth.

So as you go through each of the document distance versions, you want keep a scorecard of the implementation that shows you what the running time is, and this helps you follow what was improved in each implementation. So let's look at to get words from line lists. What does it seem like its doing? Without reading the get words from string, can anyone tell me what it seems like it's doing? If you just read lines 1 through 6.

AUDIENCE: [INAUDIBLE] through the list.

VICTOR COSTAN: OK. So it's getting an input list. And if you look at word frequencies for files at line 2, it names a variable line list. So it seems like what's happening is, reads a file into a list of lines. And then that list of lines goes to get words from line lists. So this is L in get words from line lists. So it takes a list of lines which is the entire document, and then?

AUDIENCE: Basically it removes the new lines. It sticks it into one giant list rather than a list of lines, is that right?

VICTOR COSTAN: Almost, so you seem to get words from string. Maybe we need to go through the function, but do see the get words from string function name? So I will assume that it does something with each of the words. And if the overall goal is to get a list of words, then I would assume that what that does is it takes a line and it breaks it up into words. Because this way, if you take up each line and break it up into words, then when we put all the words together we get the words that make up the document.

Do people follow? Any questions? I like that people are nodding, by the way, keep doing that. That helps me go at the right speed. If you're not nodding, I'll keep explaining the same thing over and over again. OK, so get words from string. Get words from string takes up a single line, that's a string, and produces a list of words. And we saw in the example there that it has to take care of a few details such as making all the letters lowercase and ignoring punctuation and skipping spaces.

So let's look at this code and figure out its running time. And the way we're going to do that is we're going to look at each line, and we're going to see what's the cost for that line and how many times does it run. And once we have those two numbers, we multiply them together and we see how much time does the program spend on that line in total.

So I'm going to write down line numbers here. 9, 10, 11, 12, 13, 14, 15. All the way to 23. Too low. 20, 21, 22, 23. OK, so let's start with something easy, lines 9 and 10 How many times are they run?

AUDIENCE: Once.

VICTOR COSTAN: OK.

AUDIENCE: [INAUDIBLE] Once in this method?

VICTOR COSTAN: Yep. So I'm only looking at this method. So assuming that the method gets one line, and the line has, I don't know, say, one line in characters, and we need another variable which we're going to figure out later. But for now, one line in characters. So how many times does line 9 run?

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: OK. Runs once. How about line 10?

AUDIENCE: Once.

AUDIENCE: Once.

VICTOR COSTAN: OK. What do they do? Create new lists and assign them to variables. What's the cause for that?

AUDIENCE: Constant [INAUDIBLE]

VICTOR COSTAN: Constant, excellent. So I'll be skipping the order of so that I don't have to write it 23 times. So 1, 1. OK, line 11. It's iterates over all the characters in a line. So how many times is it going to run?

AUDIENCE: Like, the line?

VICTOR COSTAN: OK, which is?

AUDIENCE: Line end characters.

VICTOR COSTAN: Awesome. And just the fact of iterating takes constant time. I'm not sure we covered that. So for each character, test if it's an alphanumeric character. Does anyone know what alphanumeric means?

AUDIENCE: It's a letter and a number.

VICTOR COSTAN: OK, so fancy word for letter or number, A through Z, 0 through 9. So how much time does it take to test if a character is alphanumeric? Guesses?

AUDIENCE: Constant.

VICTOR COSTAN: OK, so constant time. You compare it to the range A, Z and 0, 9. How many times am I doing it?

AUDIENCE: [INAUDIBLE]

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: Thank you guys, this is going much faster than the last recitation. You guys are active, I like it. So, now for line 13. That only gets executed when the character is an alphanumeric character. So we're going to have to make some assumptions about the document. And to make my life easier, we're going to make the following assumption.

If this is a natural language like, say, English, words are going to be a constant size, right? How many 500-character words do you see in English? So let's say 5 to 10 characters per word. And since the difference is so small, I'm going to say all the words have the same size W. And if you want to be more formal, you can replace word length with average length, and the math works out. So each line has a number of words, and the words are separated by exactly one space, and the word has W characters. So how many words do I have, by the way?

AUDIENCE: N divided by W.

VICTOR COSTAN: OK, good. Someone's paying close attention. N divided by W plus 1. And the reason that is, is a line would look like this, word, space, word, space, word, space. So W, characters, one space, W, characters, one space, W, characters, one space. That's why you have W plus 1 there. When we look at asymptotics it turns out that it doesn't really matter because W's a constant, W plus 1 is a constant, so order and words. But for now, let's keep track of W's to seem a bit more formal. So line 13. How many times is it going to run?

AUDIENCE: W times 10 over W plus one.

VICTOR COSTAN: Excellent. Let me pull them out. How much time does it take to run [INAUDIBLE].

AUDIENCE: Constant?

VICTOR COSTAN: Constant time, append, covered in lecture, constant time. So this is a bit tricky because if you have an array implementation that's naive, it's not constant time. But Python does some magic called table doubling, which we'll cover later in the course. And this is why you can say that append takes constant time. OK. Else, so if the character is not alphanumeric, than what's going on here? Can anyone see what's happening there?

AUDIENCE: If its like, [INAUDIBLE].

VICTOR COSTAN: OK, so let's say if it's a space.

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: Yeah, this the harder part. I think you need to run this on an example to figure out what's going on. I have to run it on an example in my head. So let's take this small example here, the fox is outside. And this is a single line, right? Nice and handy. So this can be the input for get words from string. And let's see what happens. So first I start with word list which is empty list, character, lists, empty list.

Take the first character, it's alphanumeric, gets appended here, the second character, alphanumeric, appended here, third character, alphanumeric, gets appended here. Fourth character, not alphanumeric, so I get to run lines 15 through

18. OK, I did the easy part. Someone walk me through the hard part. What happens in lines 15 through 18? Yes.

AUDIENCE: First, it takes that list and joins it into a string. [INAUDIBLE]

VICTOR COSTAN: OK, so this is a list of characters. And join takes the list and makes a string out of it. So I'll have the string the. OK, excellent.

AUDIENCE: And it converts it all to lower case.

VICTOR COSTAN: OK.

AUDIENCE: End up [INAUDIBLE] that to the word list.

VICTOR COSTAN: The world list is up here, right? So this is going to have the.

AUDIENCE: And then it clears the character list, [INAUDIBLE].

VICTOR COSTAN: OK. So now as I go through the next word, I have F-O-X. Then this becomes the word, and it gets added here. So on and so forth for everything. Do people see how this method works now? I'm not getting that many nods, so questions. If I don't get nods, I'll stop and you guys have to ask what you're confused about.

AUDIENCE: I think it's a little tricky because instead of saying if it's not an alphanumeric character, it's just like well, if the length of the list is greater than 0, which threw me off initially, but then I realized it was just, like, omission.

VICTOR COSTAN: OK, so why does it do this? What is the point of the length of the character list?

AUDIENCE: So that there are two spaces.

VICTOR COSTAN: Excellent. So here I was nice and I had one space, one space, one space. But if I'm sloppy when I'm typing and I have two spaces here, then suppose this is space, space-- kind a small, but pretend. Go with me here. So we got here. We got the fox is. And then this list is empty because line 18 just made it empty.

If I run the code the lines 15 through 18, it's going to add an empty word up here.

And empty words aren't very useful. You'll see how many times the documents have too many spaces in them, so that doesn't really help.

AUDIENCE: I mean, isn't that not an issue, because you call if C is L1 before you actually get to that. So you'd run through it again, but you would still just skip over that. That would fail, I mean it would not do anything for that equation.

VICTOR COSTAN: So first space. C as L now fails. I run lines 15 through 18.

AUDIENCE: Yep.

VICTOR COSTAN: Right? I have is here. This becomes empty.

AUDIENCE: Yep.

AUDIENCE: Second space, C as L now fails again.

AUDIENCE: Yep.

VICTOR COSTAN: And if I wouldn't have the length check, it would run lines 15 through 18 again.

AUDIENCE: Oh, OK. [INAUDIBLE]

VICTOR COSTAN: OK, so this is what it's trying to prevent. So you can see that this code looks complicated, right? It's trying to do a lot of things, it's complicated, it's hard to analyze. Oh, well, let's go with it. Let's try to finish it up quickly. So now that we know what it does, let's try to figure out how many times each line runs and what's the cost? Yes.

AUDIENCE: So I think the total cost is N times 1 minus W over W plus 1.

VICTOR COSTAN: Wait, so here?

AUDIENCE: Yeah.

VICTOR COSTAN: OK, so you're saying N times 1 minus. OK. Why do you say that? I like it, but why? OK, it's because it's everything that is in the character, and the line above it was characters--

VICTOR COSTAN: OK.

AUDIENCE: --all alphanumeric, [INAUDIBLE]

VICTOR COSTAN: So basically spaces, right? If we have word, space, word, space, word, space, this happens for all the spaces. Cool. So this is good. I'm going to make it a bit simpler. Same thing, it's just that it's slightly less intimidating.

AUDIENCE: Oh, yeah.

VICTOR COSTAN: Cool, thank you. Very brave, come up first. What's the running time for line 14? So, cost for running it once.

AUDIENCE: Constant. Excellent.

VICTOR COSTAN: I like you guys. Nice. Line 15, how much time does it to take to take characters and put them into a list?

AUDIENCE: N?

VICTOR COSTAN: N--

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: --where N is the size of the list, right?

AUDIENCE: Yeah.

VICTOR COSTAN: OK. So what's the size of the list now?

AUDIENCE: [INAUDIBLE]

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: Yep. OK, so when you're using more than one letter, the problem is you have to pay attention to which one you're using. Because when we teach algorithms, we say oh, this is N, this is N squared, so on and so forth. You have to replace it to the right letter. And I get confused about this all the time, so--

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: --a serious problem.

AUDIENCE: --columns? What are the two columns?

VICTOR COSTAN: So this is the cost of running a line once, and this is how many times it's run.

AUDIENCE: Oh, OK.

VICTOR COSTAN: Thanks for the question. I should have said that in the beginning. Thank you. OK, let's make this a little bit faster and notice that lines 15 through 18 all run the same number of times, right? They're in the if, and there's nothing else that's changes the control flow there.

So lines 15 through 18 are O and divided by W plus 1. All right, line 16. Take a word. So take a string and make another string where each character is the lowercase version.

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: OK, cool. Why W, intuitively?

AUDIENCE: Because [INAUDIBLE] has to check to make sure [INAUDIBLE]

VICTOR COSTAN: Yep.

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: Yeah, so if you have a 10,000 character string you, have to go through 10,000 characters. Very good. Append 917.

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: Sweet. And line 18, we said the character list of length list.

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: [INAUDIBLE] OK, how many times do lines 19 through 23 run?

AUDIENCE: Once.

VICTOR COSTAN: At most, once.

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: Can anyone figure out what's the point of them?

AUDIENCE: Catch any trailing [INAUDIBLE]

VICTOR COSTAN: Good. If you ended on the last letter of a word, you want to make sure you catch that word.

VICTOR COSTAN: All right.

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: Very good. So I find it here. Then after I'm done with the loop at line 19 what the word list would have, the fox is. And then the character list would have the characters for outside. If I return the word list, woops, I just missed a word. So lines 20 through 22 are a copy of lines 15 through 17, and they take care of the last word. So line 19 is an if, and it takes the length of a list and compares it to the number. What's the cost of that?

AUDIENCE: Constant.

VICTOR COSTAN: OK, very good. Checking list length in Python is constant time. We did that in lecture. How about lines 20 through 22? I just gave it away, guys, come on. Someone--

AUDIENCE: The same as 15 through 17.

VICTOR COSTAN: OK, same as 15 through 17. W, W, 1. Line 23, return constant time. OK, so now we know how much it takes to run a line once, how many times each line runs. So we're going to do a dot product of these guys. See, dot products are useful. And if we do a dot product of these guys, we're going to get the total running time for the function. So let's compute the partial terms.

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: I'm not going to write them down. Let's just go through them and figure out what they are. So you guys say them.

AUDIENCE: 1, 1, N, N, weird equation--

VICTOR COSTAN: OK, weird equation, what was the important part?

[INTERPOSING VOICES]

VICTOR COSTAN: Yeah, the important part. The important part is N, right? This is some constant times N, so N.

AUDIENCE: N, N, N, N, N, N, 1, 1.

VICTOR COSTAN: Pay attention.

AUDIENCE: 1, N.

VICTOR COSTAN: Pay attention. It's not N, it's not 1.

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: OK, actually is 1 I guess, if you think that W is a constant. Sorry.

AUDIENCE: You're testing us.

VICTOR COSTAN: OK. 1, 1.

VICTOR COSTAN: So I heard two numbers, N and 1, right? So this is 0 of N plus 1, which is order N, because as N goes to infinity, 1 becomes really tiny. OK, so this is how you analyze a function. Big functions are horribly painful to analyze because you have to look at each line and do this kind of reasoning. And it's not even a top level function here, so I don't even get to write anything here yet. So get words from string takes order and time where N is the length of a line. Let's look at get words from line list.

AUDIENCE: I have a question.

VICTOR COSTAN: Yes.

AUDIENCE: So [INAUDIBLE] is W characters long? Like, does it matter if the [INAUDIBLE]

VICTOR COSTAN: Does it matter--

AUDIENCE: [INAUDIBLE] make that assumption of that?

VICTOR COSTAN: So that I can reason for lines 15 and 16. I can reason through them easily if I have a content length. It turns out that if you have an average length, the results are going to be the same. Like overall, if you look at the running time as a sum of what's the running time for converting all the words to lowercase and then appending them to the list. The sum of those is still going to be n N, but that takes a bit more time to reason through so I took a shortcut.

Are you a math major, by the way? You're very rigorous. OK. So this is good, it's always good to try to keep this in the back of your head to make sure you don't fall for a trap. So get words from string order N, and we're trying to figure out get words from line list.

Any more questions before I do that? Or does anyone want to tell me I'm wrong? OK, good. So get words from line list. Lines 2 through 6. 2 3, 4, 5, 6. Line 2.

AUDIENCE: 1.

VICTOR COSTAN: OK, cost 1, how many times does it run?

AUDIENCE: Once.

VICTOR COSTAN: Cool. Line 3. We need a new number, right? We need the number of lines in a document. Let's say we have Z lines. So line 3 runs Z times, and 4 and 5 are in a loop so they also run Z times What's the cost for line 4?

AUDIENCE: 1.

VICTOR COSTAN: Excellent. What's the cost for line 3?

AUDIENCE: 1.

VICTOR COSTAN: 1. And what is the cost for line 5?

AUDIENCE: Looks constant.

VICTOR COSTAN: Looks constant, OK.

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: Does anyone else think it looks constant? Yeah.

AUDIENCE: It's a trap.

VICTOR COSTAN: It's a trap. It's a trap.

[INTERPOSING VOICES]

AUDIENCE: --length of the two lists.

VICTOR COSTAN: OK. Good. You paid attention in lecture, right?

AUDIENCE: I try.

VICTOR COSTAN: Nice. OK, so we have plus as an operator, and suppose we work with two lists. The first list is 1, 2, 3, all the way through 1,000. And the second list is 1, 2, 3. So when you code plus to combine them, if you say something like C equals A plus B, you would expect that-- if this is A, by the way and this is B-- you would expect that after you call this A is still this, B is still this, and C is a list that contains everything.

So because of that, what plus has to do is make a new list, append all the elements here, append all the elements here. So the cost of this if this list is 1,000 and this list is 3 is 1,003. Or if you have two lists of length, L1 and L2 the cost is order of L1 plus L2. Now there's another Python method called extend, which does what I think you would expect plus to do in terms of efficiency. So what extend does is you call it a 1 or A on one list, give it the other list, and it's going to take each element in the second list and append it to the first list. So for each element here, it calls append on this list. So what's the running time for extend?

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: OK, there are too many directions and--

AUDIENCE: Length of the second list.

VICTOR COSTAN: Length of the second list, excellent. So two lists, L1, L2, order of L2. So it doesn't matter this is 1,000 elements are a million elements, appending three elements is going to take time proportional to three. OK now, let's see what's going on here. So we have Z lines and characters in a line. I think I want a nicer constant. No, let's go with this for now.

AUDIENCE: [INAUDIBLE] lines.

VICTOR COSTAN: So this is the length of a word. Let's see, how many words will I have in a line? Let's say I have K words in a line, which is N divided by W. So I know that to get words from string returns a list of size K. So if that is the case, then the first time line 5 runs, word list is empty.

And it's going to get K elements. The second time it runs, word list has K elements and gets K more. Third time, it has 2K elements, it gets K more. So the running time for this looks like this. K plus 2K plus 3K plus 4K all the way until when I'm at the last line, if I have Z lines.

I had Z minus 1 times K elements in the list, because I have Z minus 1 lines and I put all the words in the list, and I'm adding K more words. So total, Z times K running time. So this is the total running time for this guy. And this is not constant, so it's complicated. What is the sum come down to, asymptotically?

AUDIENCE: Z plus 1K times Z over 2.

VICTOR COSTAN: Ok. Z plus 1K, ZK over 2. Slow because I care about asymptotics, this is order of Z squared times K, right? So now any one more natural number to work with would be the number of words in a document. And the number of words in a document is W, which is Z times K. So Z is W divided by K.

And if I substitute this, I get that this is equal to 0 of W squared over K. Now in a reasonable document that I see, there tends to be a limited number of words per line because the document has to fit on a page. So K's pretty much a constant. So this comes down to order of W squared.

So if I go down here and look at get word from line list, this is W squared, where W is how many words I have in a document. How many of you guys are still with me? Half. OK. Does anyone else want to ask questions, so that you can get back on track? Yes, no?

AUDIENCE: It makes sense so far.

VICTOR COSTAN: Thank you.

AUDIENCE: I think I didn't understand the part of [INAUDIBLE]

VICTOR COSTAN: OK. Thank you. So let's see what's going on lines 2 through 5. So I have a word list, which at the beginning is empty. Then in line 4, words in line gets K words. And those K words in line five are added to word list. So after that, word list has K words. Then I run through the loop again. Get the words from string gives me K new words.

They get added to the list, which now has 2K words. Next time I get K more words, they get that added to the list, which has 3K. So on and so forth until the end. I have ugly numbers. Z minus 1 times K words and I add the last K words. I'm getting confused here. And I get Z times K words.

So the word list is eventually going to have Z times K words, and it gets them K at a time. The thing that does this addition is the plus operator. And the running time for the plus operator is the size of the two lists, so it's this plus this. So that's why the running time is first K, then 2K, then 3K, then-- make sense now?

AUDIENCE: Yes.

VICTOR COSTAN: OK. So this is a subtle bug because if you change plus to extend, you get [? bug ?] disk two, which runs a lot faster. OK. So for everything else, we want to be able to do this sort of analysis, but we want to do it faster. So you guys should look through [? bug list ?] one through eight and do the same analysis for all the functions. And we're going to post recitation notes where we tell you this is the function that changed, and this is the total running time. And you should go through the lines and convince yourself that this is the right running time.

And you should do that until it becomes second nature, because when you're writing Python code, you want to have this in your head. You don't want to have to write it down, because if you have to write it down, you're going to be lazy and you're not going to do it, and you're going to use plus instead of extend, and your code is going to be horribly slow. So practice until this gets in your head, and then you'll be able to see the running time for things really quickly.

OK, do we have time for once more let me see. OK. Let's look at the running time for inner products, because this is nice and easy. 2, 3, 4, 5, 6, 7. 2 is 1, 1, very nice and easy. 3 looks at the first document list and iterates through it. Iteration is constant time, but if the first document vector has L1 elements, it's going to run L1 times. How about line 4, words 2 count 2 in L2. This is iteration again, so it's constant time to run it once, but how many times will it run?

AUDIENCE: L2 times L1 times.

VICTOR COSTAN: L2 times the 1, excellent. So these two loops are nested inside each other so that means that lines 4 through 6 are going to run once every time line 3 iterates. So sorry, actually line 4 is going to run once every time line 3 iterates. And then everything inside the second 4 is going to run L1 times L2 times. So lines 5 and 6 are also going to run L1, L2 times. L1, L2, L1, L2.

How much time does it take to do that if check there?

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: Why does it take a constant time?

AUDIENCE: I was going to say, it wasn't constant, so you don't have to pair each character with no word.

VICTOR COSTAN: OK, good. So we have two words, and equal, equal tells me are the words equal or not, right? So the way you do that, is you have words like the and fox. You go through each character, and you stop whenever you see different characters. But if you have something like, if you have a fake word F-O-I and fox, then go through the first character, they're equal, second character, they're equal, third character, they're different. So if you have length W words that are different only in the last character, this is going to be order W, right? So the real--

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: --yep, equals, equals 4 strings not constant. It takes W time where W is the length of a word. Now here we said that the length of a word is constant because we're dealing with English. So you could tell me it is constant because of that. But I would like to hear the argument before I take it. How about line 6?

AUDIENCE: Well, if the plus equals is going to be the same thing before when we were, every new time your plus equals, so it's going to be like how the word list before we were adding it, where we have to create that object, and then add it to the length. I mean, its going to be length of sum. Sorry. And then you add in the new one. So every time its going to be increasing, correct?

VICTOR COSTAN: Almost. It's a trap again.

[INTERPOSING VOICES]

VICTOR COSTAN: Yep. Yeah, so this time they're not lists. So if you look at what's going on inside there, you have count one and count two are these numbers in the document vector, so they're numbers. And then some starts out at 0, and then it keeps getting numbers. So sum is going to be a number. And multiplying numbers is constant time, adding numbers is constant time, so plus for numbers is order 1 indeed.

AUDIENCE: You're reassigning sum every time?

VICTOR COSTAN: Which is also constant.

AUDIENCE: OK.

VICTOR COSTAN: Because you're copying a number over. So as long as you're copying one element over, that's constant time. If you're adding two elements together-- two elements, not two lists-- that's constant time. So this is constant. And the last line is returned. So what's the running time for this?

AUDIENCE: L2 times L1.

VICTOR COSTAN: Excellent. So I assume this is a constant. So this lets me say this is 1, and then if we do the partial products we get 1L, 1L, 1, and L2. L1, L2, L1, L2. And if you add them up, you get L1 and L2. So this is going to be L1, L2. Vector angle calls inner product three times, right? So what's it's running time?

AUDIENCE: L1, L2.

VICTOR COSTAN: Excellent. Count frequency. You're going to have to take my word for it that this is order of W squared. And if that's the case, what's the running time for a word frequency for file?

AUDIENCE: W squared?

VICTOR COSTAN: Cool. So. What's the running time for main now? Last trick.

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: Yep, If you just add them up, except there is one last trick there.

AUDIENCE: If W is constant, [INAUDIBLE]

VICTOR COSTAN: No.

AUDIENCE: [INAUDIBLE] W's constant, right?

VICTOR COSTAN: No. So W is the number of words in a document.

AUDIENCE: Oh.

VICTOR COSTAN: So it's huge. If that's constant, then the whole problem should run in order one time, and we're done. We're going home.

AUDIENCE: W squared because it beats out L1 and L2.

VICTOR COSTAN: OK, so--

AUDIENCE: L1--

VICTOR COSTAN: --you're going faster than me. You're going too fast, but you're right. So word frequency for file is called twice. The first document is going to have W1 words. The second document is going to have W2 words. So you can just copy W because this is called twice for different files. So this is order of W1 squared plus W2 squared, different documents. And then I have plus L1, L2.

And you said that W1 and W2 dominate L1 and L2, right? Because W's the total number of words in a document, whereas L the is the number of unique words, because it the length of the vector. So that is true, but I'm not sure how to reduce this here to make use of that. However, I made use of what you said already when I wrote this. You see why? Can anyone else see why?

So let's look at the vector angle again, lines 2 and 3. So line 2, it calls inner product with L1 and L2. But if you look at line 3, it calls inner product with L1, L1 and then L2, L2 So the total running time for vector angle is actually L1, L2 plus L1 squared plus L2 squared. So if the first document has 1,000 words and the second document as one word, computing the inner product between L1 and L1 is going to take a lot more time than computing the inner product between L1 and L2. So I can't leave out these terms. They have to be here.

However, when I add them up here-- if I would write W1 squared plus W2 squared plus L1 squared plus L2 squared plus this-- in that case, I can use the fact that W1 is bigger than L1, and it cancels it out. Does this make sense? Did I lose people? Ask questions, please.

AUDIENCE: But you can't get rid of L1 and L2 and not an [INAUDIBLE].

VICTOR COSTAN: You can't--

AUDIENCE: [INAUDIBLE]

VICTOR COSTAN: Oh, so I can't get rid of this term--

AUDIENCE: --those, right? So this should be the sum of this and this, right?

AUDIENCE: Right.

VICTOR COSTAN: So it should be W1 squared plus W2 squared plus L1 squared plus L2 squared plus L1, L2.

AUDIENCE: Right. L1 is strictly smaller than W1.

AUDIENCE: Yeah. Goes away, L2 smaller than W2 goes away, and I get this. Correct. So L1L2 isn't smaller than W [INAUDIBLE] squared?

VICTOR COSTAN: Is it? If you know more math than me, you might be able to prove that it is, but I don't, so I'm just leaving it in there.

AUDIENCE: Ok.

VICTOR COSTAN: Yeah. I think there is some relation, but I really don't remember what it this, so let's leave it like that for now. Yeah, I think it should be the case that these are bigger than this, but I'm not sure. OK, yes.

AUDIENCE: How do you get the line for vector angle?

VICTOR COSTAN: How do I get the running time for it? So vector angle gets two vectors, right? The vector for document one and the vector for document two. The length of the first vector is L1. The length of the second vector is L2. Now, line, where is it? Line 2, for numerator calls inner product with L1 and L2. So we know that the running time is L1, L2 up here. Now the next line, line 3 in vector angle, calls inner product with L1 and L1. So the running time is L1 times L1 which is L1 squared. OK.

AUDIENCE: Can we say that because there's a bounded number of words in the English language, L1's bounded? And as the length of the document gets really, really big, that [INAUDIBLE] constant?

VICTOR COSTAN: Yeah, you might be able to do that. Yes, I think for the cases that we give you, that is true. Yeah, I never thought of that, that's cool.

AUDIENCE: It doesn't work if it's not a language, right? If you just have gibberish?

VICTOR COSTAN: Yes, also, to say that its constant is useful when the number of words in English is much smaller than your input size. So if, say, English has 50,000 words and your input is 3,000 words, then the input is much smaller. But if you're input is a million words, which I think is what we use, then yeah, it comes down to constant. So yeah, that's a good insight. That's really nice.

Anything else? OK, so you get to go through document distance 3 to 8. We'll tell you what's changed, and we'll give you a chance to help you analyze it. But you have to analyze it, then update the scorecard for each algorithm to see how things improve. Thanks.