# ONLINE SUPPLEMENTARY MATERIALS

## Appendix 1. Transcript of the academic lecture

Some students find this course particularly challenging. So we will have extra help. We will post weekly office hours on the course website for the TAs.

And then as an experiment this term, we are going to offer homework labs for this class. So, what a homework lab is, is it's a place and a time you can go where other people in the course will go to do homework. And there will be typically two TAs who staff the lab. And so, as you're working on your homework, you can get help from the TAs if you need it. And it's generally a place, we're going to schedule those, and they will be on the course calendar for where it is and when it is that they will be held, but usually Sundays two to four pm, or else it will be some evening. I think the first one is an evening, right? Near to when the homework is due.

Your best bet is try to do the homework in advance of the homework lab.

But then, if you want extra help, if you want to talk over your solutions with people because as we will talk about problem sets you can solve in collaboration with other people in the class. Okay? In addition, there are several peer assistance programs. And those usually get. Also the office of Minority Education has an assistance program, and those usually get booked up pretty quickly.

If you're interested in those, good idea to make an appointment to get there and get help soon. Okay. So the homework labs, I hope a lot of people will try that out. We've never done this.

I don't know of any other course.

Do other people know of courses at MIT that have done this?

6.011 did it, OK

Good. And was it successful in that class? It never went,

OK. Good. We will see. If it's not paying off then we will just return to ordinary office hours for those TAs, but I think for some students that is a good opportunity.

If you wish to be registered in this course, you must sign up on the course Web page. So, that is requirement one. It must be done today. We will…You will find it difficult to pass the course if you are not in the class.

Okay, and you should notify your TA if you decide to drop so that we can get you off and stop the mailings, stop the spam.

Okay, and you should register today before seven pm

And then we're going to email your recitation assignment to you before noon tomorrow. Okay. And if you don't receive this information by Thursday noon, please send us an email. Okay. Saying to the course staff generally, not to me individually, saying that you didn't receive your recitation assignment. And so if you haven't received it by Thursday noon, you know, you want to. I think generally they are going to send them out tonight or at least by tomorrow morning.

Yeah.

OK. SMA students don't have to worry about this.

Problem sets.

We have nine problem sets that we project will be assigned during the semester. A couple things about problem sets. Homework won't generally be accepted, if you have extenuating circumstances, circumstances you should make prior arrangements with your recitation instructor.

In fact, almost all of the administrative stuff, you shouldn't come to me to ask and say can I hand in something late? You should be talking to your recitation instructor. You can read the other things about the form, but let me just mention that there are exercises that should be solved but not handed in as well to give you drill on the material. I highly recommend you doing the exercises. They both test your understanding of the material, and exercises have this way of finding themselves on quizzes.

You're often asked to describe algorithms. And here is a little outline of what you can use to describe an algorithm. Okay. The grading policy is something that somehow I cover. And always every term there are at least a couple of students who pretend like I never showed them this. Okay. So, if you skip problems, it has a nonlinear effect on your grade. Okay.

Nonlinear, OK?

If you don't skip any problems, no effect on your grade. Okay?

If you skip one problem, a hundredth of a letter grade, we can handle that. Okay. But two problems it's a tenth.

And, as you see, by the time you have skipped like five letter grades, it is already a third of five problems.

This is not problem sets, by the way.

This is problems, OK?

You're down a third of a letter grade.

Okay. And if you don't do nine or more, so that's typically about three to four problem sets. Okay you don't pass the class.

Okay. I always have some students coming at the end of the year saying oh, I didn't do any of my problems. Can you just pass me because I did OK on the exams?

Answer no. Okay. A very simple answer because we've said it upfront. Okay

So, the problem sets are an integral part of the course.

Collaboration policy. This is extremely important. So everybody pay attention. If you are asleep now wake up. Okay.

Like that's going to wake anybody up, right?

Okay. The goal of homework.

Professor Demaine and my philosophy is that the goal of homework is to help you learn the material.

And one way of helping to learn is not to just be stuck and unable to solve something because then you're in no better shape when the exam rolls around, which is where we're actually evaluating you. So, you're encouraged to collaborate. But there are some commonsense things about collaboration. If you go and you collaborate to the extent that all you're doing is getting the information from somebody else, you're not learning the material. And you're not going to do well on the exams.

So, in our experience, students who collaborate generally do better than students who work alone. Okay.

But you owe it to yourself, if you're going to work in a study group, to be prepared for your study group meeting. And specifically you should spend a half an hour to forty five minutes on each problem before you go to your group. So you're up to speed and you've tried out your ideas. Okay. And you may have solutions to some, you may be stuck on some other ones, but at least you applied yourself to it.

Okay. After thirty to forty five minutes, if you cannot get the problem, just sitting there and banging your head against, it makes no sense. It's not a productive use of your time. And I know most of you have issues with having time on your hands, right?

Like it's not there. So, don't go banging your head against problems that are too hard or where you don't understand what's going on or whatever. So, that's when the study group can help out. And, as I mentioned, we'll have homework labs which will give you an opportunity to go and do that and coordinate with other students rather than necessarily having to form your own group. And the TAs will be there.

So, if your group is unable to solve the problem then talk to other groups or ask your recitation instructors. And, that's how you go about solving them. Okay.

Writing up the problem sets, however, is your individual responsibility and should be done alone. You don't write up your problem solutions with other students. You write them up on your own. Okay.

And you should on your problem sets, because this is an academic place, we understand that the source of academic information is very important, if you collaborated on solutions you should write a list of the collaborators. Say I worked with so and so on this solution. Okay. It does not affect your grade. It's just a question of being scholarly. Okay.

It is a violation of this policy to submit a problem solution that you cannot orally explain to a member of the course staff. So, you say oh, well, my write up is similar to that other person's because you know because I didn't copy them. You know. We may ask you to orally explain your solution. If you are unable, according to this policy, the presumption is that you cheated. So, do not write up stuff that you don't understand. Okay. You should be able to write up the stuff that you understand. Understand why you're putting down what you're putting down. If it isn't obvious, no collaboration whatsoever is permitted on exams.

Exams is when we evaluate you. Okay. And now we're not interested in evaluating other people, we're interested in evaluating you. Okay. So, no collaboration on exams. We will have a take home exam for the second quiz. You should look at the schedule. If there are problems with the schedule of that, we want to know early. And we will give you more details about the collaboration in the lecture on Monday, November twenty eighth. Now, generally, the lectures here, they're mandatory and you have to know them, but I know that some people say gee, nine thirty is kind of early, especially on a Monday, you know, or whatever. It can be kind of early to get up. Okay. However, on Monday, November twenty eighth, you fail the exam if you do not show up to lecture on time. Okay. That one day you must show up.

Any questions about that? Okay? That one day you must show up here, even if you've been watching them on the Web.

Okay. And generally, if you think you have transgressed, the best is to come to us to talk about it. We can usually work something out. It's when we find somebody has transgressed from a third party or from obvious analyses that we do with homework, that's when things get messy. So, if you think, for some reason or other, oh, I may have done something wrong, please come and talk to us. Okay. We actually were students once, too, albeit many years ago. Okay? Any questions?

So, this class has great material. Fabulous material. Okay. And it's really fun, but you do have to work hard. Okay. So, let's talk content.

Okay. This is the topic of the first part of the course. The first part of the course is focused on analysis. The second part of the course is focused on design. Before you can do design, you have to master a bunch of techniques for analyzing algorithms. And then you'll be in a position to design algorithms that you can analyze and that which are efficient.

The analysis of algorithm is the theoretical study of computer program performance and resource usage. And a particular focus on performance. Okay. We're studying how to make things fast. In particular, computer programs. We also will discover and talk about other resources such as communication, such as memory, whether RAM memory or disk memory. Okay.

So, there are other resources that we may care about, but predominantly we focus on performance. Okay. Because this is a course about performance, I like to put things in perspective a little bit by starting out and asking, in programming, what is more important than performance? If you're in an engineering situation and writing code, writing software, what's more important than performance? Correctness.

Good. OK. What else?

Simplicity can be.

Very good. Yeah.

Maintainability often much more important than performance.

Cost. Okay. And what type of cost are you thinking? No, but I mean cost of what?

We're talking software here, right? So, what type of cost do you have in mind?

Okay. There are some costs that are involved when programming like programmer time. So, programmer time is another thing also that might be. Stability.

Robustness of the software. Does it break all the time?

What’s else?

Come on. We've got a bunch of engineers here. A lot of things.

How about features? Features can be more important.

Having a wider collection of features than your competitors.

Functionality.

Modularity.

Okay. Is it designed in a way where you can make changes in a local part of the code and you don't have to make changes across the code in order to affect a simple change in the functionality?

Okay. There is one big one which definitely, especially in the 90s, was like the big thing in computers.

The big thing. Well, security actually. Good. I don't even have that one down. Security is excellent. Okay. That's actually been more in the 2000. Okay. Security has been far more important often than performance.

Scalability has been important, although scalability, in some sense, is performance related. But, yes, scalability is good.

So, what was the big breakthrough and why do people use Macintosh rather than Windows, those people who are of that religion? User friendliness.

Wow. If you look at the number of cycles of computers that went into user friendliness in the 90s, it grew from almost nothing to where it's now the vast part of the computation goes into user friendly. So, all those things are more important than performance. So, this is a course on performance.

Then you can say OK, well, why do we bother and why study algorithms and performance if it's at the bottom of the heap? Almost always people would rather have these other things than performance.

You go off and you say to somebody, would I rather have performance or more user friendliness?

It's almost always, you know, more important than performance.

Why do we care then? Yeah?

That wasn't user friendly. Sometimes performance is correlated with user friendliness, absolutely. Okay. Nothing is more frustrating than sitting there waiting, right?

So, that's a good reason. What are some other reasons why? Sometimes they have real time constraints so they don't actually work unless they perform adequately. Yeah?

Hard to get, well, we don't usually quantify user friendliness so I'm not sure that, but I understand what you're saying. He said we don't get exponential performance improvements in user friendliness. We often don't get that in performance either, by the way. Okay. Sometimes we do, but that's good.

So, there are several reasons that I think are important.

One is that often performance measures the line between the feasible and the infeasible. Okay, we have heard some of these things. For example, when there are real time requirements, if it's not fast enough it's simply not functional.

Or, if it uses too much memory it's simply not going to work for you. Okay. And, as a consequence, what you find is algorithms are on the cutting edge of entrepreneurship. If you're talking about just reimplementing stuff that people did ten years ago, performance isn't that important at some level.

But, if you're talking about doing stuff that nobody has done before, one of the reasons often that they haven't done it is because it's too time consuming. Things don't scale and so forth.

So, that's one reason, is the feasible versus infeasible.

Another thing is that algorithms give you a language for talking about program behavior, and that turns out to be a language that has been pervasive through computer science, is that the theoretical language is what gets adopted by all the practitioners because it's a clean way of thinking about things. A good way I think about performance, and the reason it's on the bottom of the heap, is sort of like, performance is like money, it's like currency.

Okay, you say what good does a stack of hundred dollar bills do for you? Would you rather have food or water or shelter or whatever? And you're willing to pay those hundred dollar bills, if you have hundred dollar bills, for that commodity. Even though water is far more important to your living. Well, similarly, performance is what you use to pay for user friendliness. It's what you pay for security. And you hear people say, for example, that I want greater functionality, so people will program in Java, even though it's much slower than C, because they'll say it costs me maybe a factor of three or something in performance to program in Java. But Java is worth it because it's got all these object oriented features and so forth, exception mechanisms and so on.

And so people are willing to pay a factor of three in performance. So, that's why you want performance. Okay, because you can use it to pay for these other things that you want. And that's why, in some sense, it's on the bottom of the heap. Okay, because it's the universal thing that you quantify. Do you want to spend a factor of two on this or spend a factor of three on security, et cetera? Okay,

And, in addition, the lessons generalize to other resource measures like communication, like memory and so forth. And the last reason we study algorithm performance is it's tons of fun.

Speed is always fun, right?

Why do people drive fast cars, race horses, you know, you know, whatever? Rockets, et cetera, why do we do that? Because speed is fun.

Ski. Who likes to ski? I love to ski. I like going fast on those skis. It's fun.

Hockey, fast sports, right?

We all like the fast sports. Okay. Not all of us, I mean. Some people say he's not talking to me. OK, so let's move on.

That's sort of a little bit of a notion as to why we study this, is that it does, in some sense, form a common basis for all these other things we care about. And so we want to understand how can we generate money for ourselves in computation? Okay. So we're going to start out with a very simple problem.

It's one of the oldest problems that has been studied in algorithms, is the problem of sorting.

We're going to actually study this for several lectures because sorting contains many algorithmic techniques. The sorting problem is the following.

We have a sequence a one, a two up to a nof numbers as input. And our output is a permutation of those numbers. A permutation is a rearrangement of the numbers.

Every number appears exactly once in the rearrangement such that, I sometimes use a dollar sign to mean *such that*, okay, *such that* a one is less than or equal to a two prime. Such that they are monotonically increasing in size.

Take a bunch of numbers, put them in order. Okay. Here's an algorithm to do it. It's called insertion sort.

And we will write this algorithm in what we call pseudocode. It's sort of a programming language, except it's got English in there often. Okay. And it's just a shorthand for writing for being precise.

So this sorts a from one to n. And here is the code for it.

This is what we call pseudocode.

And if you don't understand the pseudocode then you should ask questions about any of the notations. You will start to get used to it as we go on.

One thing is that in the pseudocode we use indentation, where in most languages they have some kind of begin end delimiters like curly braces or something in Java or C, for example.

We just use indentation.

The whole idea of the pseudocode is to try to get the algorithms as short as possible while still understanding what the individual steps are. Okay. In practice, there actually have been languages that use indentation as a means of showing the nesting of things.

It's generally a bad idea, because if things go over one page to another, for example, you cannot tell what level of nesting it is. Whereas, with explicit braces it's much easier to tell.

So, there are reasons why this is a bad notation if you were doing software engineering. But it's a good one for us because it just keeps things short and makes fewer things to write down.

So, this is insertion sort.

Let's try to figure out a little bit what this does.

Okay. So, it basically takes an array *A* and at any point the thing to understand is, we're setting basically, we're running the outer loop from *j* is two to n, and the inner loop that starts at j minus one and then goes down until it's zero. Basically, if we look at any point in the algorithm, we essentially are looking at some element here j. a of j, the jth element.

And what we do essentially is we pull a value out here that we call the key. And at this point the important thing to understand, and we'll talk more about this in recitation on Friday, is that there is an invariant that is being maintained by this loop each time through.

And the invariant is that this part of the array is sorted.

And the goal each time through the loop is to increase, is to add one to the length of the things that are sorted.

And the way we do that is we pull out the key and we just copy values up like this. And keep copying up, okay, until we find the place where this key goes, and then we insert it in that place. And that's why it's called insertion sort. We just sort of move the things, copy the things up until we find where it goes, and then we put it into place. And now we have it from *a* from one to *j* is sorted, and now we can work on *j* plus one. Okay. So, let's give an example of that.

Imagine we are doing eight, two, four, nine, three, six.

We start out with two, *j* equals two. And we figure out that we want to insert it there. Now we have two, eight, four, nine, three, six. Okay, then we look at the four and say oh, well, that goes over here.

We get two, four, eight, nine, three, six after the second iteration of the outer loop. Then we look at nine and discover immediately it just goes right there. Very little work to do on that step.

Okay. So, we have exactly the same output after that iteration.

Then we look at three and that's going to be inserted over there. Two, three, four, eight, nine, six. Finally we look at six. That goes in there. Two, three, four, six, eight, nine. And at the point we’ve done.

Okay. Question?

Uhm. The array initially starts at one, yes. Okay, A one to n.

OK? So, this is the insertion sort algorithm. And it's the first algorithm that we're going to analyze. And we're going to pull out some tools that we have from our math background to help to analyze it. So, first of all, let's take a look at the issue of running time.

Okay. The running time depends, of this algorithm depends on a lot of things. Okay. So, one thing it depends on is the input itself. For example, if the input is already sorted, then insertion sort has very little work to do. Okay. Because every time through it's going to be like this case.

It doesn't have to shuffle too many guys over because they're already in place. Okay. Whereas, in some sense, what's the worst case for insertion sort?

If it is reverse sorted, then it's going to have to do a lot of work because it's going to have to shuffle everything over on each step of the outer loop. In addition to the actual input it depends, of course, on the input size.

Here, for example, we did six elements.

It's going to take longer if we, for example, do six times ten to the ninth elements.

Okay, if we were sorting a lot more stuff, it's going to take us a lot longer. Typically, the way we handle that is we are going to parameterize things in the input size. We are going to talk about time as a function of the size of things that we are sorting. So we can look at what is the behavior of that. Okay. And the last thing I want to say about running time is generally we want upper bounds on the running time. We want to know that the time is no more than a certain amount. And the reason is because that represents a guarantee to the user.

Okay. If I say it's not going to run, for example, if I tell you here's a program and it won't run more than three seconds. .Okay, that gives you real information about how you could use it, for example, in a real time setting.

Whereas, if I said here's a program and, you know, it goes at least three seconds, you don't know if it's going to go for three years. It doesn't give you that much guarantee if you are a user of it.

Generally we want upper bounds because it represents a guarantee to the user.

So, there are different kinds of analyses that people do.

The one we're mostly going to focus on is what's called worst case analysis. And this is what we do usually where we define *T* of *n* to be the maximum time on any input of size *n*. So, it's the maximum input, the maximum it could possibly cost us on an input of size *n*.

What that does is, if you look at the fact that sometimes the inputs are better and sometimes they're worse, we're looking at the worst case of those because that's the way we're going to be able to make a guarantee.

It always does something rather than just sometimes does something. So, we're looking at the maximum. Notice that if I didn't have maximum then T of n in some sense is a relation, not a function, because the time on an input of size *n* depends on which input of size *n*.

I could have many different times, but by putting the maximum at it, it turns that relation into a function, because there's only one maximum time that it will take. Okay. Sometimes we will talk about average case. Sometimes we will do this.

Here *T* of *n* is then the expected time over all inputs of size *n*. It's the expected time.

Now, if I talk about expected time, what else do I need to say here? What does that mean, expected time? I'm sorry.

Raise your hand. Expected inputs.

What does that mean, expected inputs?

I need more Maths. What do I need by expected time here, Maths? You have to take the time of every input and then average them, OK.

That's kind of what we mean by expected time.

Good. Not quite.

Okay. I mean, what you say is completely correct, except is not quite enough. Yeah?

It's the time of every input times the probability that it will be that input. It's a way of taking a weighted average, exactly right. How do I know what the probability of every input is? How do I know what the probability a particular input occurs is… in a given situation?

I don't. I have to make an assumption. What's that assumption called? What kind of assumption do I have to meet? I need an assumption of the statistical distribution of inputs.

Otherwise, expected time doesn't mean anything because I don't know what the probability of something is. In order to do probability, you need some assumptions. Okay, and you've got to state those assumptions clearly. One of the most common assumptions is that all inputs are equally likely. That's called the uniform distribution. Every input of size *n* is equally likely, that kind of thing.

But there are other ways that you could make that assumption, and they may not all be true.

This is much more complicated, as you can see. Fortunately, all of you have a strong probability background. And so we will not have any trouble addressing these probabilistic issues of dealing with expectations and such. Okay.

If you don't, time to go and say gee, maybe I should take that Probability class that is a prerequisite for this class. The last one I am going to mention is best case analysis. And this I claim is bogus.

Bogus. No good.

Why is best case analysis bogus?

Yeah? The best case probably doesn't ever happen. Actually, it's interesting because for the sorting problem, the most common things that get sorted are things that are already sorted interestingly, or at least almost sorted. For example, one of the most common things that are sorted is check numbers by banks. They tend to come in, in the same order that they are written. Okay. They're sorting things that are almost always sorted.

I mean, it's good. When upper bound, not lower bound? Yeah, you want to make a guarantee.

And so why is this not a guarantee? Okay. You're onto something there, but we need a little more precision here.

How can I cheat? Yeah?

Yeah, you can cheat. You cheat.

You take any slow algorithm that you want and just check for some particular input, and if it's that input, then you say immediately yeah, OK, here is the answer.

And then it's got a good best case.

But I didn't tell you anything about the vast majority of what is going on. Okay. So, you can cheat with a slow algorithm that works fast on some input.It doesn't really do much for you. Okay. So we normally don't worry about that. Okay. So, let's see.

What is insertion sort worst case time?

Now we get into some sort of funny issues.

First of all, it sort of depends on the computer you're running on. Whose computer, right? Is it a big supercomputer or is it your wristwatch? Okay. They have different computational abilities. And when we compare algorithms, we compare them typically for relative speed.

This is if you compared two algorithms on the same machine.

You could argue, well, it doesn't really matter what the machine is because I will just look at their relative speed. But, of course, I may also be interested in absolute speed.

Is one algorithm actually better no matter what machine it's run on?

And so this kind of gets sort of confusing as to how I can talk about the worst case time of an algorithm of a piece of software when I am not talking about the hardware. Okay, because, clearly, if I had run on a faster machine, my algorithms are going to go faster. So, this is where you get the big idea of algorithms.

Which is why algorithm is such a huge field, why it spawns companies like Google, like

Akamai, like Amazon. Okay. Why algorithmic analysis, throughout the history of computing, has been such a huge success, is our ability to master and to be able to take what is apparently a really messy, complicated situation and reduce it to being able to do some mathematics. Okay.

And that idea is called asymptotic analysis.

And the basic idea of asymptotic analysis is to ignore machine dependent constants and, instead of the actual running time, look at the growth of the running time.

So, we don't look at the actual running time. We look at the growth. Okay.

So, let's see what we mean by that. This is a huge idea. It's not a hard idea, otherwise I wouldn't be able to teach it in the first lecture, but it's a huge idea. We are going to spend a couple of lectures understanding the implications of that and will basically be doing it throughout the term.

And if you go on to be practicing engineers, you will be doing it all the time.

In order to do that, we adopt some notations that are going to help us. In particular, we will adopt asymptotic notation.

Okay, so most of you have seen some kind of asymptotic notation. Maybe a few of you haven't, but mostly you should have seen a little bit. The one we're going to be using in this class predominantly is theta notation.

Okay. And theta notation is pretty easy notation to master because all you do is you just from a formula, you just just drop low order terms and ignore leading constants.

Okay. For example, if I have a formula like three n cubed plus ninety n squared minus five n plus six oh four six, I say, well, what low order terms do I drop? Well, n cubed is a bigger term than n squared. I am going to drop all these terms and ignore the leading constant, so I say that's Theta of n cubed. That's pretty easy.

So, that's theta notation. Now, this is an engineering way of manipulating theta notation. There is actually a mathematical definition for this, which we are going to talk about next time, which is a definition in terms of sets of functions. And, you are going to be responsible, this is both a math and a computer science engineering class. So throughout the course you are going to be responsible both for mathematical rigor as if it were a math course and engineering commonsense because it's an engineering course. okay, We are going to be doing both.

okay, this is the engineering way of understanding what you do, so you're responsible for being able to do these manipulations. You're also going to be responsible for understanding the mathematical definition of theta notion and of its related O notation and omega notation. Okay.

**Source:**

<https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-1-administrivia-introduction-analysis-of-algorithms-insertion-sort-mergesort/>

## Appendix 2. Information of the target single words

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Target words | Frequency of words | Number of letters | Number of syllables | Part of Speech | Type of vocabulary | Verbal elaboration1 | Nonverbal elaboration2 |
| algorithm | 28 | 9 | 4 | noun | technical | explicit | yes |
| computation | 12 | 11 | 4 | noun | academic | no | no |
| notation | 12 | 8 | 3 | noun | academic | no | yes |
| insertion | 10 | 9 | 3 | noun | technical | no | yes |
| probability | 9 | 11 | 5 | noun | academic | no | no |
| assumption | 8 | 10 | 3 | noun | academic | no | yes |
| loop | 6 | 4 | 1 | noun | academic | no | yes |
| recitation | 6 | 10 | 4 | noun | administrative | no | no |
| theta | 6 | 5 | 2 | noun | technical | no | yes |
| pseudocode | 5 | 10 | 3 | noun | technical | explicit | yes |
| asymptotic | 4 | 10 | 4 | adjective | technical | no | yes |
| element | 4 | 7 | 3 | noun | academic | no | yes |
| factor | 4 | 6 | 2 | noun | academic | no | no |
| feasible | 4 | 8 | 3 | adjective | non-specialized | implicit | no |
| array | 3 | 5 | 2 | noun | technical | no | yes |
| bogus | 3 | 5 | 2 | adjective | non-specialized | no | yes |
| constant | 3 | 8 | 2 | adjective | academic | no | yes |
| cubed | 3 | 5 | 1 | adjective | academic | no | yes |
| whereas | 3 | 7 | 2 | conjunction | academic | no | no |
| bound | 2 | 5 | 2 | noun | academic | implicit | yes |
| distribution | 2 | 12 | 4 | noun | academic | no | yes |
| formula | 2 | 7 | 3 | noun | academic | no | yes |
| invariant | 2 | 9 | 3 | noun | academic | no | no |
| iteration | 2 | 9 | 4 | noun | technical | implicit | yes |
| manipulation | 2 | 12 | 5 | noun | academic | no | no |
| nonlinear | 2 | 9 | 3 | adjective | academic | implicit | no |
| notion | 2 | 6 | 2 | noun | academic | no | no |
| orally | 2 | 6 | 3 | adverb | administrative | no | no |
| precise | 2 | 7 | 2 | adjective | academic | no | no |
| quantify | 2 | 8 | 3 | verb | technical | implicit | no |
| squared | 2 | 7 | 1 | adjective | academic | no | yes |
| theoretical | 2 | 11 | 5 | adjective | academic | no | yes |
| commodity | 1 | 9 | 4 | noun | non-specialized | implicit | no |
| constraint | 1 | 10 | 2 | noun | administrative | implicit | no |
| coordinate | 1 | 10 | 4 | verb | academic | no | no |
| correlated | 1 | 10 | 3 | adjective | academic | no | no |
| explicit | 1 | 8 | 3 | adjective | academic | no | no |
| exponential | 1 | 11 | 4 | adjective | academic | no | no |
| integral | 1 | 8 | 3 | adjective | administrative | no | no |
| mandatory | 1 | 9 | 4 | adjective | administrative | implicit | no |
| mechanism | 1 | 9 | 4 | noun | academic | no | no |
| modularity | 1 | 10 | 5 | noun | academic | implicit | no |
| parameterize | 1 | 12 | 5 | verb | technical | no | yes |
| perspective | 1 | 11 | 3 | noun | academic | no | no |
| reverse | 1 | 7 | 2 | verb | academic | no | yes |
| rigor | 1 | 5 | 2 | noun | non-specialized | no | no |
| robustness | 1 | 10 | 3 | noun | technical | implicit | no |
| sequence | 1 | 8 | 2 | noun | academic | no | yes |
| statistical | 1 | 11 | 4 | adjective | academic | no | yes |
| uniform | 1 | 7 | 3 | adjective | academic | explicit | no |

*1 Explicit elaboration means the lecturer explained the target lexical items through defining, describing, naming, or questioning. Implicit elaboration means the lecturer explaining the lexical items indirectly through examples, paraphrases, or synonyms.*

*2 Nonverbal elaboration means the lecturer explained the target items by writing the term on the board, using gestures or symbols.*

## Appendix 3. Information of the target collocations

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  Target collocations | Frequency of occurrence | MI score | Collocation components | Number of letters | Number of syllables  | Types of vocabulary | Verbal elaboration | Nonverbal elaboration |
| Word 1 | Word 2 | Word 1 | Word 2 |
| problem set | 9 | 8 | noun-noun | 7 | 3 | 2 | 1 | academic | implicit | no |
| insertion sort | 7 | 9.19 | noun-noun | 9 | 4 | 3 | 1 | technical | explicit | yes |
| running time | 7 | 6.38 | noun-noun | 7 | 4 | 2 | 1 | technical | no | no |
| user friendliness | 7 | 11.37 | noun-noun | 4 | 12 | 2 | 3 | technical | implicit | no |
| homework lab | 5 | 8.83 | noun-noun | 8 | 3 | 2 | 1 | administrative | explicit | no |
| theta notation | 4 | 8.09 | noun-noun | 5 | 8 | 2 | 3 | technical | implicit | yes |
| outer loop | 3 | 9.78 | adjective-noun | 5 | 4 | 2 | 1 | technical | no | yes |
| recitation instructor | 3 | 13.06 | noun-noun | 10 | 10 | 4 | 3 | administrative | implicit | no |
| asymptotic analysis | 2 | 9.85 | adjective-noun | 10 | 8 | 4 | 4 | technical | explicit | yes |
| asymptotic notation | 2 | 12.59 | adjective-noun | 10 | 8 | 4 | 3 | technical | no | yes |
| problem solution | 2 | 6.28 | noun-noun | 7 | 8 | 2 | 3 | academic | implicit | no |
| relative speed | 2 | 7.21 | adjective-noun | 8 | 5 | 3 | 1 | technical | no | yes |
| sorting problem | 2 | 6.27 | noun-noun | 7 | 7 | 2 | 2 | technical | explicit | yes |
| absolute speed | 1 | 4.59 | adjective-noun | 8 | 5 | 3 | 1 | technical | no | no |
| common assumption | 1 | 5.17 | adjective-noun | 6 | 10 | 2 | 3 | academic | no | no |
| common basis | 1 | 3.71 | adjective-noun | 6 | 5 | 2 | 2 | non-specialized | no | no |
| engineering commonsense | 1 | 11.56 | noun-noun | 11 | 11 | 4 | 3 | technical | no | no |
| grading policy | 1 | 7.9 | noun-noun | 7 | 6 | 2 | 3 | administrative | implicit | no |
| integral part | 1 | 5.16 | adjective-noun | 8 | 4 | 3 | 1 | academic | no | no |

## Appendix 4. Pretest (English translation)

**VOCABULARY TEST. V05.1**

*This test is only used for* ***research purposes.*** *Your performance on the test will have* ***no effects*** *on your class* ***grades****. Please be* ***honest*** *when giving the answers and* ***do not use dictionaries*** *or* ***discuss*** *with other students when doing the test.*

**PART 1.**

In this part, you will hear 52 words in turn. At the same time, you will see these words in the test paper. For each word, if you

* KNOW the word, translate it into Chinese, or explain its meaning in either English or Chinese, or write the symbol that the word stands for, or use any means to express your knowledge of the meaning of the words.
* DO NOT KNOW the word, leave the box blank.

You will have 10 seconds to write down your answer before you listen to the next word.

You will hear the recording TWICE. Here are four examples:

|  |  |
| --- | --- |
| Word |   Your answer |
| 0 | dog |  狗 |
| 00 | dad |  *Father* |
| 000 | hi |  *In informal situations, you say ‘hi’ to greet someone* |
| 0000 | beta |  β |

***Now it’s your turn!***

|  |  |
| --- | --- |
| Word |   Your answer |
| 1 | feasible |   |
| 2 | algorithm |   |
| 3 | precise |   |
| 4 | asymptotic |   |
| 5 | analysis |   |
| 6 | sequence |   |
| 7 | perspective |   |
| 8 | computation |   |
| 9 | robustness |   |
| 10 | coordinate |   |
| 11 | parameterize |   |
| 12 | commodity |   |
| 13 | iteration |   |
| 14 | cubed |   |
| 15 | manipulation |   |
| 16 | factor |   |
| 17 | assumption |   |
| 18 | invariant |   |
| 19 | bogus |   |
|  |  |  ***Go to the next page 🡪*** |
|  |  |  |
| Word | Your answer |
| 20 | theta |   |
| 21 | theoretical |   |
| 22 | distribution |   |
| 23 | nonlinear |   |
| 24 | integral |   |
| 25 | insertion |   |
| 26 | reverse |   |
| 27 | formula |   |
| 28 | orally |   |
| 29 | rigor |   |
| 30 | modularity |   |
| 31 | loop |   |
| 32 | notion |   |
| 33 | pseudocode |   |
| 34 | whereas |   |
| 35 | array |   |
| 36 | mandatory |   |
| 37 | element |   |
| 38 | probability |   |
| 39 | constant |   |
| 40 | mechanism |   |
| 41 | squared |   |
| 42 | statistical |   |
| 43 | exponential |   |
| 44 | correlated |   |
| 45 | recitation |   |
| 46 | explicit |   |
| 47 | notation |   |
| 48 | constraint |   |
| 49 | bound |   |
| 50 | uniform |   |
| 51 | quantify |   |
| 52 | machine |  |

**PART 2.**

In this part, you will hear 20 questions. For each question, you will hear four options. Circle the one correct answer. The correct answer is the two words that may often appear together in English. If you do not know or are not sure about the correct answer, choose the ‘I don’t know’ option.

After each question, you will have 5 seconds to circle and check your answer before you listen to the next question.

You will hear the recording ONCE only. Here are four examples:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Q0** | a | close friend |  | b | fine friend |  | c | long friend |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q00** | a | High house |  | b | True house |  | c | Big house |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q000** | a | Single hair |  | b | Light hair |  | c | Blonde hair |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q0000** | a | Low morning |  | b | Good morning |  | c | Blue morning |  | d | I don’t know |
| ***Now it’s your turn!*** |
| **Q1** | a | integral work |  | b | integral part |  | c | integral lab |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q2** | a | learning commonsense |  | b | relative commonsense |  | c | engineering commonsense |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q3** | a | absolute system |  | b | absolute speed |  | c | absolute lab |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q4** | a | problem phases |   | b | problem sets |   | c | problem tools |   | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q5** | a | theta part |  | b | theta notation |  | c | theta problem |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q6** | a | wrong basis |   | b | big basis |   | c | common basis |   | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  ***Go to the next page 🡪*** |
|  |  |  |  |  |  |  |  |  |
| **Q7** | a | grading policy |  | b | sitting policy |  | c | learning policy |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q8** | a | outer loop |  | b | outer notion |  | c | outer number |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q9** | a | user friendliness |   | b | user way |   | c | user time |   | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q10** | a | recitation instructor |  | b | recitation person |  | c | recitation user |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q11** | a | relative example |  | b | relative speed |  | c | relative reason |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q12** | a | insertion group |   | b | insertion sort |   | c | insertion post |   | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q13** | a | homework system |  | b | homework place |  | c | homework lab |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q14** | a | user solution |  | b | problem solution |  | c | money solution |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q15** | a | asymptotic bunch |  | b | asymptotic policy |  | c | asymptotic analysis |   | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q16** | a | turning time |  | b | running time |  | c | binding time |   | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q17** | a | asymptotic notation |  | b | form notation |  | c | world notation |  | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q18** | a | looking problem |  | b | doing problem |  | c | sorting problem |   | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q19** | a | common assumptions |  | b | leading assumptions |  | c | integral assumptions |   | d | I don’t know |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Q20** | a | computer phase |  | b | computer science |  | c | computer post |   | d | I don’t know |

## Appendix 5. Comprehension test (English translation)

***Comprehension test***

***Based on the content of the lecture, decide whether the statement True or False by circling the option.***

|  |  |  |
| --- | --- | --- |
| 1. Students who wish to be registered in this course must sign up on the course website by next week. | *True*  | *False*  |
| 2. If students fail to complete more than one out of nine tasks during the semester, this would have an impact on their grades. | *True*  | *False*  |
| 3. Collaboration is not encouraged in this course. | *True*  | *False*  |
| 4. The first part of the course focuses on analysis. | *True*  | *False*  |
| 5. The analysis part focuses on computer program performance and resource usage. | *True*  | *False*  |
| 6. Performance is more important than simplicity. | *True*  | *False*  |
| 7. Performance provides us the language for describing program behavior. | *True*  | *False*  |
| 8. Input is a sequence of numbers while output is the rearrangement of the numbers in that sequence. | *True*  | *False*  |
| 9. Input size does not affect running time. | *True*  | *False*  |
| 10. Students only need knowledge of Mathematics to understand the concepts presented in the lecture. | *True*  | *False*  |

## Appendix 6. *p-*value for the Kolmogorov-Smirnov test of normality

|  |  |  |
| --- | --- | --- |
|  | Control group | Experimental group |
| ***Single words*** |  |  |
| Pretest | 0.20 | 0.71 |
| Immediate posttest | 0.14 | 0.19 |
| Delayed posttest | 0.19 | 0.60 |
| ***Collocations*** |  |  |
| Pretest | 0.32 | 0.06 |
| Immediate posttest | 0.60 | 0.12 |
| Delayed posttest | 0.06 | 0.06 |