HOW TO GET OVER 100 SCORE ON WRITTEN TEST (GUARANTEED)

Written by yours truly, Jason Siu

Thanks for revising the first draft for 20 minutes, Affan Khurram

DISCLAIMER:

This guide will assume that you have basic knowledge of java, such as:

* General Vocabulary regarding comp. sci, such as concatenation
* Knowing how to declare/instantiate variables
* What a string is (stores words, and is a class, not data type)
* Types of loops (for, while, do/while, foreach [aka enhanced for loop])
* Switch
* Know what scanner is used for (input)
* Having some idea of what a class is (stores variables and methods)
* And other miniscule and simple concepts, able from the absolute beginner’s level

IF you have taken Computer Science AP with Langston, then you should ignore the things listed above, as you already have solid foundation to work with.

We are sure that even if you only have a very slim idea of all of the above, that you will develop these concepts in your head above naturally and intuitively by following this guide. But still, it is still good know have some idea of what’s going on before proceeding. But don’t be too intimidated, and that is the key to learning more, not to be afraid from the start.

Some tests will not have some of the stuff discussed in here, just because written tests are very different from one to another when testing concepts. However, to do well consistently given any test, you must know many of the things discussed in this book.

**If you are in a class before computer science AP, looking at this book and practicing on written tests will surely make you years ahead of your peers, and that’s no exaggeration.**

Anyways, let us proceed…

What I’ll talk about (**Note that these chapters aren’t meant to be in a particular order**):

1. What is written test?
2. Basic Number conversion
3. General Output
4. Math Methods
5. String Methods
6. Boolean Logic
7. Boolean Algebra
8. Boolean Expressions displayed through logic gates
9. Fundamentals of Data structures
10. Basics of Regular Expressions
11. Sorts and Big O Notation
12. Recursion
13. Two’s complement
14. Characteristics of Graphs, and traversals
15. Prefix, postfix notation
16. Tree traversals
17. What is the written test?

The written test is a multiple choice test that always tries to trick you, it was designed as if it had the purpose of whoever could make the least mistakes, assuming that you have some baseline of knowledge. It covers mainly the topics mentioned above, and has a structure to a written test that is very consistent. Most people generally think that the test starts from very easy to difficult at the end, but don’t be too afraid or confident, as it is mainly a matter of personal opinion and depends on what you know as well. Sometimes you could get really lucky, and some of the stuff that you haven’t learned isn’t on the test, therefore scoring greater than you usually would. There are more types of examples of how the written test can be inconsistent in scoring how much knowledge you actually know.

Typically a good score is for first time is like the 70 or 80s – that’s really good. The average for first time is like a 50 score maybe 40 or 20 doesn’t really matter. However, it’s always a beginner’s goal to get above a 100, and this is what this guide will exactly prepare for the beginners to do.

Scoring goes as is: 6 points for a correct question, -2 points for a wrong answer, and you don’t miss anything for skipping any questions. There’s a total of 40 questions, so the max score you could possibly get is a 240.

Now, let us begin learning

1. Basic Number Conversions

There is usually one or two questions on number conversions for a Java written test. They are meant to trick beginners and try to throw him off with any like weird looking question they might have never seen before, but overall the concept of number conversions is pretty simple.

The number conversions knowledge you can just get by with are just converting numbers to base 10, and base 10 to any number. You also want to get used to being familiar with base 2 numbers, as that can apply to other questions as well.

However, the ideal knowledge you want to have is that you want to learn how to convert from any base to any base and knowledge like that is pretty hard to teach within specific text so I recommend visiting other websites as well as like asking others to help. Just reading this chapter won’t actually like help unless you have the experience to approach these type problems. What I have for you is the AP comp sci that teaches number conversions just a basic idea but it’s enough to get you started. Just a disclaimer the AP comp sci chapters aren’t that good so if you don’t understand anything I recommend you just ask someone or visit a website that explains easier for beginners to understand.

When you see a number conversions question usually will see the type of Base as a subscript so if you see the number 32 with a subscript of 4 you’ll know that the number was meant to be in base 4.

**3214 base-5 = 3 x 53 + 2 x 52 + 1 x 51 + 4 x 50**

**3214 base-5 = 3 x 125 + 2 x 25 + 1 x 5 + 4 x 1**

**3214 base-5 = 375 + 50 + 5 + 4**

**3214 base-5 = 434 base-10**

Did that computation make sense? The whole point is that number representation is the exact same in any base. The only difference is the value of the base. The **3** in **3215 base-5** has a value of **375**. The same three in **3215 base-4** has a different value because the base is different, and the value is **3 x 43 = 3 x 64 = 192**. Check out the following five exercise examples, and then try it on your own.

|  |
| --- |
| **Exercise 3.1:**  **213 base-4 = ??? base-10** |
| 2 x 42 + 1 x 41 + 3 x 40 =  2 x 16 + 1 x 4 + 3 x 1 =  32 + 4 + 3 =  **39 base-10** |

|  |
| --- |
| **Exercise 3.2:**  **2134 base-5 = ??? base-10** |
| 2 x 53 + 1 x 52 + 3 x 51 + 4 x 50 =  2 x 125 + 1 x 25 + 3 x 5 + 4 x 1 =  250 + 25 + 15 + 4 =  **294 base-10** |

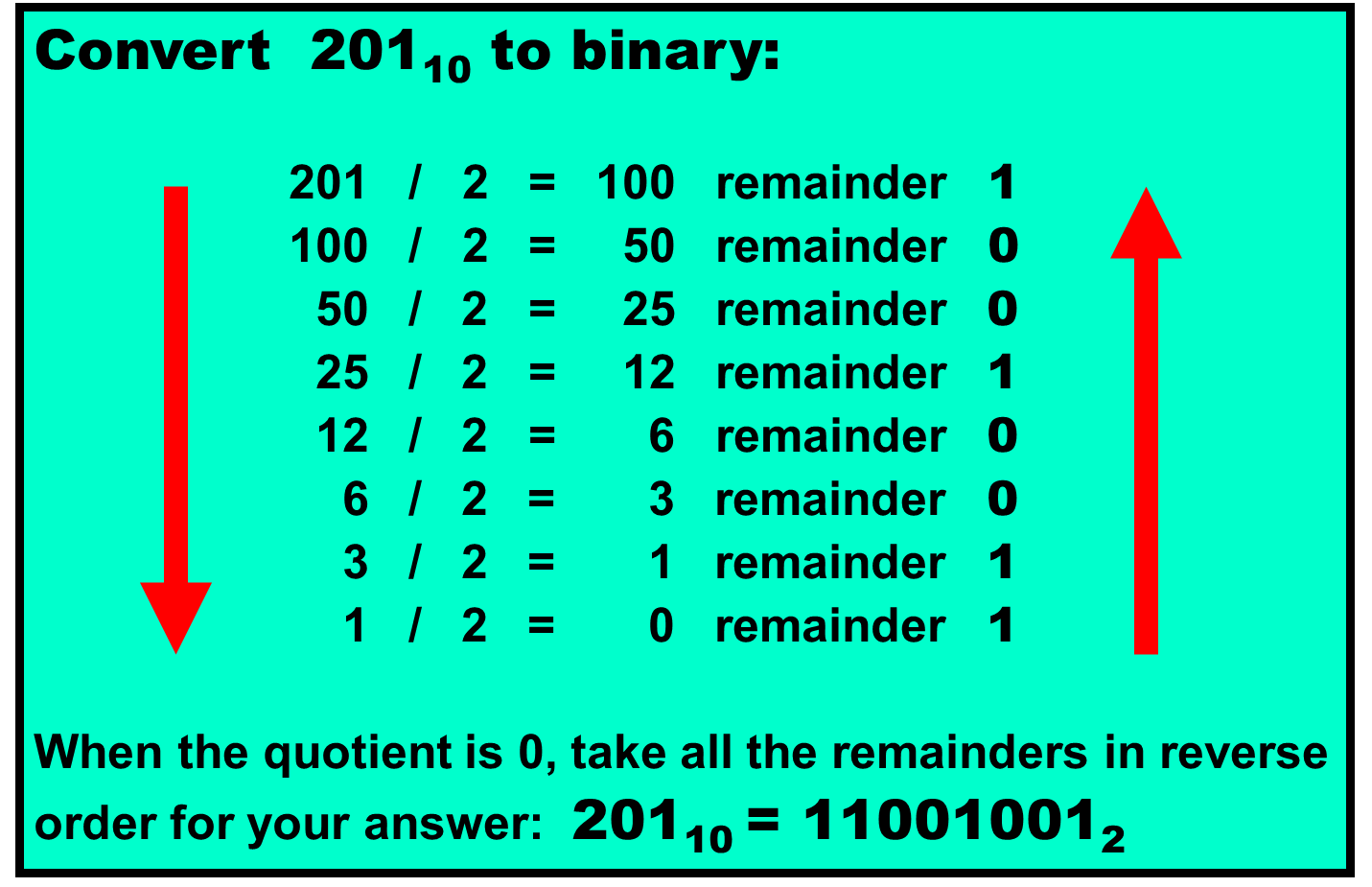
|  |
| --- |
| **Exercise 3.3:**  **175 base-8 = ??? base-10** |
| 1 x 82 + 7 x 81 + 5 x 80 =  1 x 64 + 7 x 8 + 5 x 1 =  64 + 56 + 5 =  **125 base-10** |

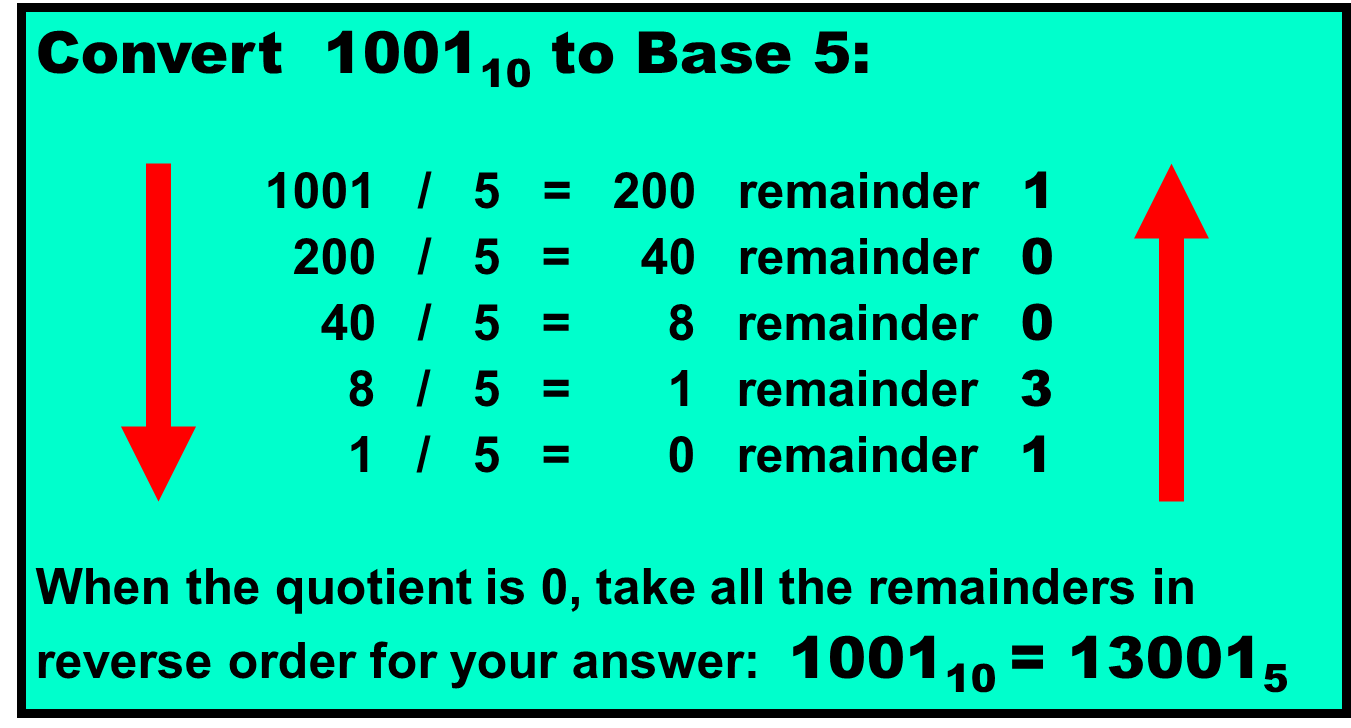
When you have to convert from base 10 to any base, the process is very different. I can describe it by dividing, then taking the remainder, but of course, with text, that can be hard to understand.

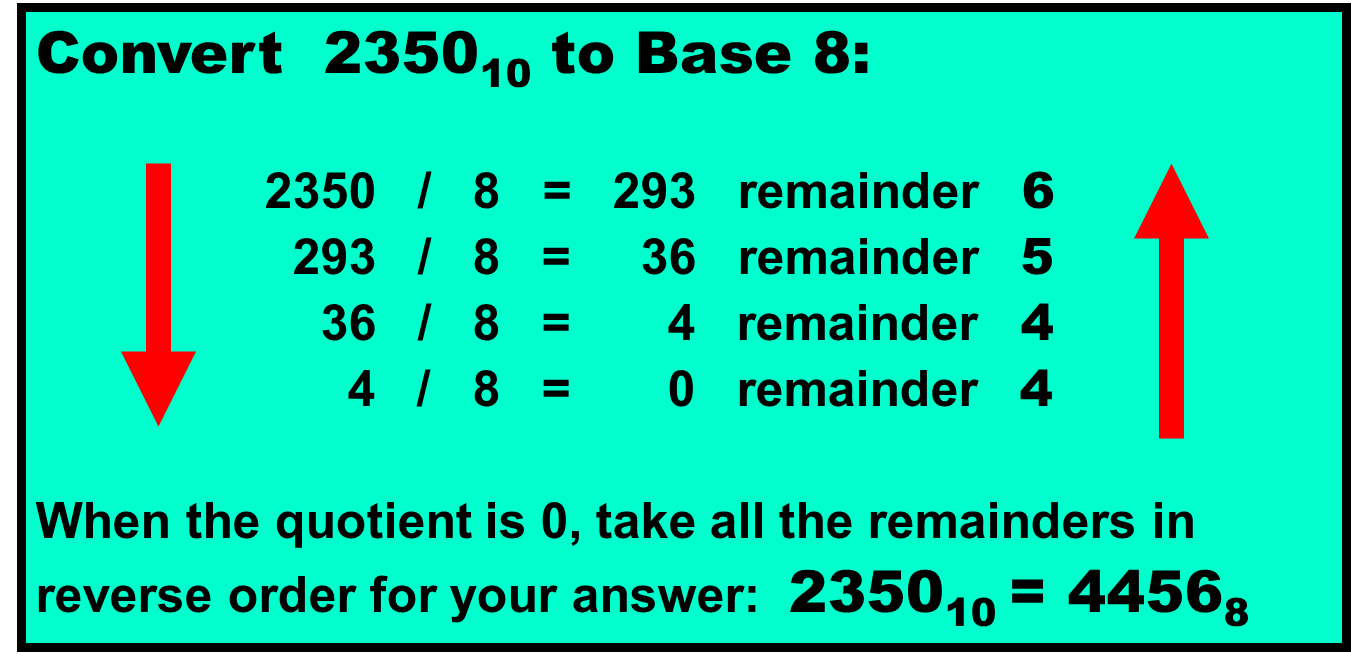
The next example demonstrates converting to base-2 for simplicity. This is followed by other examples with different bases. Other examples will follow.

You may be surprised that the expansion method is not used in reverse. That certainly works. You can set up a table with place holder values for each base.

Here is more of a visual example in order to help you understand more…







|  |  |
| --- | --- |
| **Convert 50610 to hexadecimal:** | |
| **Using the division method, when we take the remainders in reverse order we get** 1(15)(10).  **We need to do one extra step to make this right.** |  |
| **When the** 10 **and** 15 **are converted**  **to** A **and** F**, we can now properly find the answer of** 1FA**.** |  |

Here is a nice Little trick for you to convert to base 2 to base 16.

Suppose we have number 110001112 and we want to convert to base 16, we can just split there number up into groups, each having 4 numbers, like so: 1100 0111. Then, we can just find the values of each group, so the first group would be the value of 12 (because 8 + 4), and the second group would be 7 (4+2+1). Now, in base 16, as we know, 12 is represented as C, and 7 is just 7. So we just group those two together, and we get a value of C716. There same goes for any “power of 2” base, such as base 4 and 8. What’s different is that for base 4, you split the number into group size of 2, and for base 8, you would split it into group size of 3. This makes sense because log2(4) = 2 aka the group size for converting base 2 to base 4. Same thing with 8 : log2(8) = 3 – the group size when converting from base 2 to base 8.

For going back to base 2, let’s stick with our example.

We have a number C716 and we want to convert to base 2. Well we just convert C aka 12, to base 2. That is 1100. Then we convert 7 to base 2. That is 111. However, when we join the two numbers together, we want to make sure that all numbers have a length of 4. In this case, 111 has a length of 3, so we want it to be 0111. Now, we combine the two numbers, getting 110001112. You can see a pattern going on with the length. If you want to convert from base 8 to base 2, you want to have all numbers with a length of 3 before combining.

Anyways, that it at least the basics for learning this chapter. Make sure you actually have some practice with this, because this isn’t something you can just memorize and see instant results.

1. General Output

In the written test there will be many questions regarding output. So, we will generalize it in this chapter. Most of the easier questions will be at the beginning. Most of these target what you know about the order of operations by giving you expressions that you have to evaluate. Remember, there is no calculator allowed on the written test, so make sure you don’t make any simple mistake. On the rare occasion, you will need scratch paper, but you shouldn’t be too afraid, as many of these questions are the freebies, considered by most. This chapter will indeed be short because you are just doing everyday math.

An example question would be:

What is the output?

System.out.println(3+9/3.0)

And the answer would be 6.0

Note that the answer is the data type of double, and not integer. This is because we involved doubles in our PEMDAS expressions

However, you must see some examples from tests, so I would encourage you to see some of these computer output questions online by searching for an old practice test online.

Once you do these questions, you will find quickly that they are way easier than the other questions.

1. Math Methods

Math methods are used frequently on the written test. Typically these questions are aimed at how much you know about the return types of the math methods. You will be given a page on your test that has only some of the math methods, and most likely looking at that page won’t guarantee that you automatically know the answer.

Here, I will organize the list of most commonly used math methods by return type:

Returns LONG:

Round(double a)

Returns INT:

Min(int a, int b)

Max(int a, int b)

Returns DOUBLE:  
 abs

Pow

Sqrt

Floor

Min(double a, double b)

Max(double a, double b)

Random

Cbrt (cubed root)

Hypot(returns hypotenuse of two sides of a triangle.)

You see how there were min and max methods having a different return type? That is because of their parameter’s data type, and it corresponds to their respective return type as well. This is an example of an overloaded method. If you don’t know what that is, I suggest asking another person, or looking it up online. It is one of the basics you must know because truly having a grasp of this concept.

I went over the math methods because it is easy to assume the wrong data type being returned. I remember I had once missed a question because I thought you could assign an integer value using the round method. However, you can’t because it returns long, and long has a bigger size than int, therefore returning an error. If you don’t know the sizes of the data type. I highly recommend looking over that, as there might be 1 question in about 10 tests over size. Though this may seem small, it is good to know to have some basic understanding of the language.

1. String Methods

As you already know, String is a class. And it has methods for the convenience of the user. You, the person who will take the written test, will need to know the basics of some of the more important string methods.

[**boolean equals(Object anObject)**](https://www.tutorialspoint.com/java/java_string_equals.htm)

Compares this string to the specified object. You would want to use this method when you want to see if two strings are equal. It is a bad practice to use the ‘==’ operator with strings.

[**char charAt(int index)**](https://www.tutorialspoint.com/java/java_string_charat.htm)

Returns the character at the specified index. It is similar to finding the specific element in an array. It starts at 0, then proceeds until length – 1.

[**int compareTo(Object o)**](https://www.tutorialspoint.com/java/java_string_compareto.htm)

Compares this String to another Object.

Method compareTo returns 0 if s1 equals s2,

otherwise an integer is returned based on the difference between s1 and s2.

If the returned value is negative, it means that s1 goes before s2.

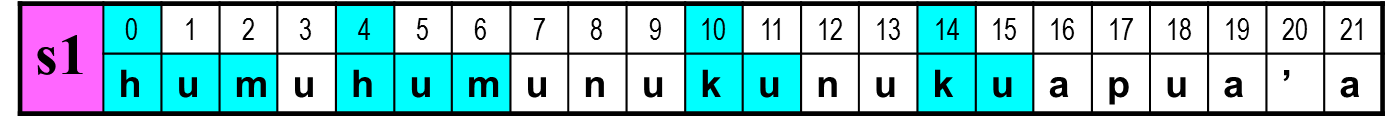
If the returned value is positive, it means that s1 goes after s2.

It is called that it compares in lexicographic order, or dictionary order, if you will

[**int indexOf(int ch)**](https://www.tutorialspoint.com/java/java_string_indexof.htm)

Returns the index within this string of the first occurrence of the specified character.

**indexOf** returns the first occurrence of a substring.

****

**s1.indexOf(“hum”);** returns **0**

**s1.indexOf(“ku”);** returns **10**

**s1.indexOf(“qwerty”);** returns **-1**

If the substring cannot be found a value of **-1** is returned.

[**int length()**](https://www.tutorialspoint.com/java/java_string_length.htm)

Returns the length of this string.

[**String substring(int beginIndex, int endIndex)**](https://www.tutorialspoint.com/java/java_string_substring_beginendindex.htm)

Returns a new string that is a substring of this string.

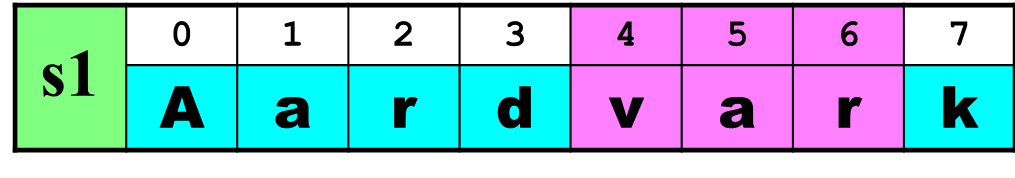
[**String substring(int beginIndex)**](https://www.tutorialspoint.com/java/java_string_substring.htm)Returns a new string that is a substring of this string.

**String s1 = "aardvark";**

**String s2 = s1.substring(j,k);**

Method **substring** returns a set of consecutive characters from string **s1**, starting at index **j**, and ending at index **k-1**.

**String s3 = s1.substring(4,7);**

****

**s3** becomes **"var"**

NOTE: The first index of a **String** is always **0**

.

1. Boolean Logic

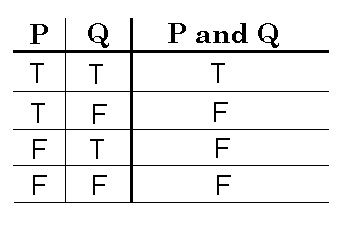
Boolean variables, as we should know, can ONLY have a value of true or false. There will be questions on the test which will say something like “This Boolean expression can be expressed the same thing as”, and you would have options that would you have to choose on which is equal to the expression given.

For an example: What is equivalent to !(a == b)

And the answer would be (a != b). Though this is a very easy question (because I lack the creativity to create a better one), you will be introduced to other types of these questions. If this may seem difficult to you at first, try using a truth table.

A truth table is a table that simplifies this whole process, making it much more intuitive to understand. However, writing out a truth table is time consuming, but very much worth to get a correct answer.

Example of a truth table.



You can see how it shows us the expression P and Q, for all combinations. If we keep doing this process for longer and complex expressions, things will become much easier. If you still don’t understand, I suggest asking an ACO officer who performs well on written tests for further clarification or just look up some guides online and you can get some further understanding there.

1. Boolean Algebra

There is usually only one or two questions about this per test, so if this may seem difficult, don’t worry too much.

Yes, there is algebra for Boolean variables. Though most questions aren’t about solving for X, it is mainly simplify Boolean expressions. This will not give any practice questions, as there are videos to guide you through simplifying Boolean expressions. Beware, on the test, these types of questions because you are always prone to making a mistake. As you will see in future examples, the + means OR, and the dot, representing a multiplication sign, means AND. Hopefully you can understand why these rules exist, if you kind of think about them in your head. If you do that, you wouldn’t need to memorize these rules, and they will become much easier. You also notice that most of these rules are derived from the first few equations. Also, when you look at the equations below, 1 would mean TRUE, and 0 would mean FALSE.

Again, I will emphasize to NOT memorize these formulas UNLESS you don’t understand them!

Here are the rules that we have in Boolean algebra:

Laws and Theorems of Boolean Algebra

1a. X • 0 = 0 1b. X + 1 = 1 Annulment Law

2a. X • 1 = X 2b. X + 0 = X Identity Law

3a. X • X = X 3b. X + X = X Idempotent Law

4a. X • X = 0 4b. X + X = 1 Complement Law

5. X = X Double Negation Law

6a. X • Y = Y • X 6b. X + Y = Y + X Commutative Law

7a. X (Y Z) = (X Y) Z = (X Z) Y = X Y Z Associative Law

7b. X + (Y + Z) = (X + Y) + Z = (X + Z) + Y = X + Y + Z Associative Law

8a. X • (Y + Z) = X Y + X Z 8b. X + Y Z = (X + Y) • (X + Z) Distributive Law

9a. X • Y = X + Y 9b. X + Y = X • Y de Morgan's Theorem

10a. X • (X + Y) = X 10b. X + X Y = X Absorption Law

11a. (X + Y) • (X + Y) = X 11b. X Y + X Y = X Redundancy Law

12a. (X + Y) • Y = XY 12b. X Y + Y = X + Y Redundancy Law

13a. (X + Y) • (X + Z) • (Y + Z) = (X + Y) • (X + Z) Consensus Law

13b. X Y + X Z + Y Z = X Y + X Z Consensus Law

From this point on, these rules you won’t need to know, but still interesting to look at:

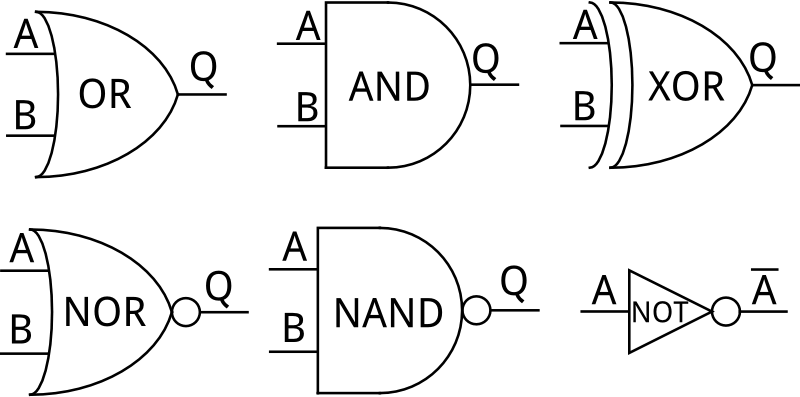
14a. X ⊕ Y = (X + Y) • (X + Y) 14b. X ⊕ Y = X Y + X Y XOR Gate

15a. X ⊙ Y = (X + Y) • (X • Y) 15b. X ⊙ Y = X Y + X Y XNOR Gate

15c. X ⊙ Y = (X + Y) • (X + Y) XNOR Gate

1. Boolean Expressions displayed through logic gates

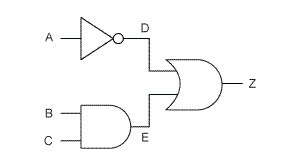
This is a very simple concept. All you have to do for this chapter is memorize which picture matches with what operation.



If you can’t understand the picture, the logic gate for OR would be that pointed, dented oval. The AND logic gate would be the half oval, and XOR would be the dented oval with a line at the back.

You can see the pattern that the circle at the end of a logic gate would mean NOT. The line on top of the latter is the same thing as NOT. In java, this would be represented as !a.

You can combine these gates to get Boolean expressions. Guess what this combination of logic gates is equivalent to:



In case you were wondering the expression is equal to

!A || (B && C)

This chapter was all about making sure you know what these pictures mean. You can just memorize the pictures and you will be fine.

1. Fundamentals of data structures

You’ll hear many different things about the fundamentals of data structures. Most tutorials online focus on the absolute beginner, and they go into extensive depth that one does not need to know. Again, you only need to know the basics, and then once you take tests, build your knowledge from there.

Data structures in computer science are basically things that contain other variables or data structures themselves. Data structures are different in the way they store things, access, or do other operations.

You’ll need to be most familiar with:

* ArrayList (same thing as primitive array, but it can change size)
  + Has the same thing as a primitive array, but it can change size. This is why it’s also called a dynamic array, because of the size changing structure. Has methods that I encourage to look at your own (get, set, remove)
* LinkedList
  + An element in a list has the address of the element after it. The End element doesn’t connect back to the first. However, there are other types of linked lists that will connect the end to the first. What I just described to you initially is also called a singly-linked list
* Stack
  + Last in, First out. (LIFO) Basically if you are the last one in line, you are the first one to get out.
* Queue
  + First in, First out (FIFO). Basically how a standard lunch line works, or the line at the drive through works in your McDonald’s restaurant
* Binary Search Tree
  + A binary search tree (BST), also known as an ordered binary tree, is a node-based data structure in which each node has no more than two child nodes. Each child must either be a leaf node or the root of another binary search tree. Additionally, if a node has sub nodes below it, the left subnode would have a lesser value than the root node. The right subnode would have a greater value than the root node.
* Heap
  + A binary heap is a complete binary tree which satisfies the heap ordering property. The ordering can be one of two types:
  + the min-heap property: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.
  + the max-heap property: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.
* Graphs (we will have separate chapter for)

There are other data structures that are not as common, and I firmly believe that if you just know the basics of the data structures above, you should be fine for answering any data structures questions on the written test.

1. Basics of Regular Expressions

Regular Expressions (aka regex) are a computer’s way of knowing what you want to search for given a string, or other forms of text. These types of questions are mainly formatted

If you need extra help on this chapter, visit <https://regex101.com/> for a better understanding

What is the output?

System.out.println(Pattern.matches(“[ab]”, “Hello”));

And the answer would be false, and you’ll see why soon

Here are the basics you need to know for regular expressions, you may be confused on what the following may mean, but read it for now, and later on I will continue to explain more.

\d – matches any number

\D – anything but a number

\s – space

\S – anything but a space

\w – any character

\W – anything but a character

. – any character except for linebreak

\b - matches space that precedes/follows a whole word

+ - one or more

? – zero or one repetitions

\* - 0 or more repititions

{n} – except n number of times of the code that precedes it

Ex. \d{1,5} match 1 to 5 digits in a row

[a-z] – searches all lowercase letters

[0-9] – searches 0-9

Ex. [A-Fa-t0-4] – searches A-F, a-t, and 0-4 (dashes aren’t included, the dash simply means that it matches any of the values in between say A and F, or a and t, and so on.

| - OR

Ex. (Bob|Bobby|BobChreste) = will search for Bob, Bobby, or BobChreste

^ - (if at the beginning of a regex, means NOT

() – capture everything enclosed

Practice:

(a|b) – matches either a or b

A{3} – match exactly 3 of A

A{3,} – match ‘A’ three or more times

[^abc] – matches whatever is NOT a, b or c

[ab^cd] – matches a,b,^,c,d

**Example problem:**

What does

\d{1,2}[\,\.]\d{1,2}

Match to?

Answer: any number 1 or 2 times(can be two different numbers) , followed by a comma or period, then any number 1 or 2 times.

And so the example problem I gave:

System.out.println(Pattern.matches(“[ab]”, “Hello”));

It would output false because it doesn’t match the whole string. There are no instance of a, and no instance of b.

The matches method from the pattern class has to match the entire string, not just some parts of it to output true. And there can’t be multiple matches, only 1 in order for it to output true.

If you have any questions, ask an ACO officer who does well on written tests.

1. Sorts and Big O Notation

Yes everyone has heard of sorting, but not so much of Big O Notation. Those things go hand in hand though.

An example question from the written test would be

Bubble sort has a worst case time complexity of:

And the answer would be O(N2)

We will see in a moment why this is the correct answer

**At first, know that Big O notation isn’t just applied to sorts, its also applied to data structures as well. This is why if you go to helpful websites about big o notation such as :**

<http://cooervo.github.io/Algorithms-DataStructures-BigONotation/>

or

<http://bigocheatsheet.com/>

**They have data structures in them as well. For now, what you need to sort of memorize is the time complexity (fancy word of saying efficiency), of the more common sorts such as:**

Bubble Sort

Insertion Sort

Selection Sort

Quick Sort

Merge Sort

Again, because this is all of the time complexity is given online, I will give an extremely brief rundown of each sort

Bubble (aka garbage sort, but still better than bogo sort)

Compares two elements then, if needed, swaps, then compares the next two elements. Keeps going until sorted

Selection

Picks the minimum element from the list, then puts that into the first spot, then finds the second least element, and puts that in the next spot, etc until sorted.

Insertion

Creates a sorted list of the first two elements, then the first 3, then the first 4. This goes on until sorted entire list.

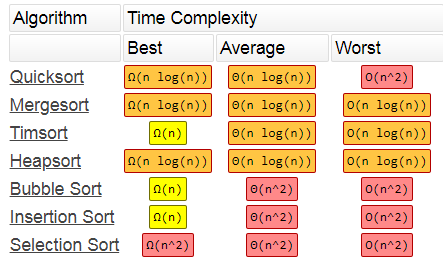
Quick Sort

Creates a randomly selected element. All of the elements below it go on the left, all of the elements greater than it go on the right.. Repeats this process until sorted.

Merge Sort

Splits the list into two, and it keeps doing that until you have 1 element separated. Once you have that, it merges single elements to form sorted lists of size 2. Then it merges those types of lists to form a bigger and sorted list of size 4. It keeps combining sorted lists (aka what we just described), until you have the entire list sorted.

HERE IS A TABLE OF BIG O NOTATION FOR THOSE SORTS (IGNORE TIMSORT and heap sort)



O(1) – means that it takes constant time aka it only takes certain amount of time, regardless of the Size.

O(log n) – means that it takes a time that is similar to the logarithm of the size

O(n) – means that it takes a time proportional to the size

O(n \* logn) – takes longer than O(N)

O(N2) – time that is quadratic to the size (worst)

The lower the operation is inside of the O, the better because you want to take less time for everything for more efficiency.

As you can see, the worst performing sorts are bubble, selection, and insertion.

The better ones are quick and merge.

This is pretty hard to explain in text, so I advise you to ask an ACO officer, or check out the links I have provided for help.

For this chapter, you don’t need to understand why these sorts have a certain Big O notation for the best, worst, and average case scenario.

I know this may seem very unsatisfying to read, just memorize. If you want a deeper understanding, again, ask an ACO officer.

1. Recursion

Recursion is simply when a function calls itself. Tutorials that you read online may say something about the stack, but you do not need to know that.

Example Problem:

What does the value func(3) return?

Int func(int x)

If(x == 0)

Return 0;

Else

Return x + func(x-1);

And the answer to this would be 6, and you’ll see why.

Recursion is when a function calls itself, but eventually, it needs to stop, or else there will be a runtime error. This condition where the computer needs to stop calling itself is called the base case. In the example above, the base case is the x==0, return 0 part.

Let’s see why our answer was 6:

Func(3) = 3 + func(2)

Func(2) = 2 + func(1)

Func(1) = 1 + func(0)

Func(0) = 0

Because we know the value of func(0), we know the value of func(1), which is 1 + 0, meaning that func(1) = 1. Because we know the value of func(1) now, we know the value of func(2), which is 2 + 1, meaning that func(2) = 3. We do the same thing for func(3), which is 3+3, meaning that func(3) is equal to 6.

That is the key to recursion. You keep going deeper and deeper, than once you reach the base case, you go back up, and solve for everything else.

There are other examples that you can find online, so if you are curious still, ask an ACO officer for a deeper understanding, if you want.

On the written test, there are problems that may take more time, but overall, it has the difficulty of the example problem given.

Here’s another problem I will give you:

What is the output when func(2,4) is called:

Int func(int a, int b)

If(b == 1)

Return a;

Else

Return a + (a, b-1);

The answer is: 8

1. Two’s complement

If you didn’t understand chapter 2 very well, this chapter should re inforce those concepts mentioned in the chapter.

As a disclaimer, binary means base 2 for displaying values.

Binary representation, so far, has only been for positive numbers. However, even asked for the complement, things can get kind of tricky.

The complement of a binary number is getting a negative version of the binary representation.

For example, 101011102 would equate to 174 in base 10.

However, in 2’s complement form, this would equate to -82.

You’ll see why later.

There are plenty of other methods to calculate the 2’s complement, but the way I will show you is a method I do myself.

OK, so here goes how I got -82.

In any binary digit number, the first digit will be of the greatest value. The position of the number determines its value. For example, 1010, the first digit represents a number that is greater than those below. If you don’t seem to understand this, imagine the number 11 in base 10. The first digit represents the ten’s place, while the second digit represents the one’s place, or single digit number, ten times less than the digit as seen before.

So, the first digit of a binary number will be its initial negative value. What do I mean by that?

This means that the 2’s complement of 100000002 would be -128, because the first digit represents the negative value, so having a 1 on the first digit (where it would regularly mean POSITIVE 128 in regular binary conversion), would mean that it would equate to -128.

Let’s go back with our previous example, but this time I will show it to you in vertical direction that is much more understandable.

1 – means -128

0 - nothing

1 – means positive 32

0 - nothing

1 - means 8

1 - means 4

1 - means 2

0 – nothing

If we take the sum of those values, it would equate to -82.

Here’s an example with all ones for every digit

1: -128

1: 64

1: 32

1: 16

1: 8

1: 4

1: 2

1: 1

The sum is -1, therefore the 2’s complement of 111111112 is -1.

Keep in mind the colons mean nothing, and I the dashes from the previous example would be confusing with the -128 because there would be two dashes in a row.

Example problem:

What is 2’s complement of the binary number of :100101112?

Solution:

1: -128

0

0

1: 16

0

1: 4

1: 2

1: 1

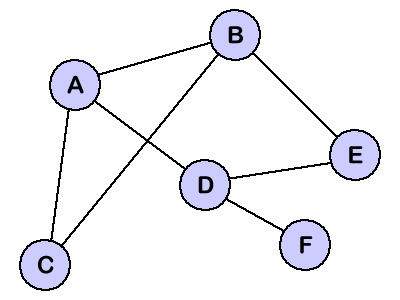
-128 + 16 + 4 + 2 + 1 = -105

Answer: -105

1. Characteristics of graphs and their traversals

Graphs, you’ll see looks like a bunch of circles with lines connecting them.

Example:



As you read the following characteristics, if you don’t understand what I’m fully saying, search up pics on google for a visual understanding.

Characteristics of graphs:

1. Weighted/Unweighted
   1. Weights means that there are numbers to each of the lines (or paths), describing the path. And example could be traffic – the higher the number, the slower it is to get to each circle (or nodes). AKA if there are numbers on the lines, it’s a weighted graph
   2. Unweighted mean there are no numbers on the lines / paths
2. Directed/Undirected
   1. Directed shows an arrow or indicator where a path leads, and also says that the path may not go in the opposite direction. The path is only one way, and that way is indicated by the arrow.
   2. Undirected means that it can go both ways, and there are no arrows.
3. Complete/Uncomplete
   1. Complete means that every node has a path to every other node. The technical definition is that every single pair of nodes has a connection, which means the same thing.
   2. Uncomplete means that there isn’t a direct connection to every node.
4. Connected/Unconnected
   1. Connected means that a graph is connected when there is a path between every pair of vertices. In a connected graph, there are no unreachable vertices. So that means no matter what, there is always a path to ALL nodes.
   2. Unconnected means that there are instances of unreachable nodes.

Traversals (optional to know):

Breadth first search - A breadth-first search (BFS) is another technique for traversing a finite graph. BFS visits the neighbor vertices before visiting the child vertices, and a queue is used in the search process. This algorithm is often used to find the shortest path from one vertex to another.

Depth first search - A depth-first search (DFS) is an algorithm for traversing a finite graph. DFS visits the child vertices before visiting the sibling vertices; that is, it traverses the depth of any particular path before exploring its breadth. A stack (often the program's call stack via recursion) is generally used when implementing the algorithm.

1. Prefix, Postfix Notation

These types of notation are for evaluating expressions, so that a computer could read them easier.

Mathematical expressions such as 1\*2 + 3, which would evaluate to 5. IN this case, this is the regular types of expressions we are used to seeing, which are also called INFIX notation, where there is an operator ( +, - , \*, / ), between two operands (numbers).

**POSTFIX:**

Postfix notation is when the operator is after the two operands.

Consider this example:

5 7 \*

This would evaluate to 35 in postfix notation. Because you have the two operands (5 and 7), and you have the operator which multiplies the two. Let’s try another more complex example.

9 7 2 - \*

This would evaluate to 45.

The first operator read from left to right is the subtraction sign. Before the subtraction sign, we see 7 and 2. Subtract those and get 5. We now have the expression of

9 5 \*

And it just multiplies 9 and 5, getting a value of 45.

Here is a more complex example:

5 7 + 6 2 - \*

This simplifies down to 48.

Again, we scan from left to right, looking for the first operand, and we see a plus sign. We look at the two numbers before it and add those two together.

The resulting expression is now

12 6 2 - \*

Again, we read from left to right, looking for the first operand, and we see a subtraction sign. We look at the numbers is corresponds to, and subtract those values, which is 6 – 2 = 4.

Now, we get the resulting expression

12 4 \*

And now we just multiply the two to get 48.

**PREFIX:**

Same thing as postfix, but instead, the operator occurs before the two operands.

Example:

+ 9 5

Would result in 14. The operator is a plus sign, and you have the two operands 9 and 5, so you add the two to get 14.

Here’s another example:

/ + 2 7 3

The answer to this question is 3. This is because you read from right to left, and look for the first operator. In this case, it’s a plus sign. Then you look at the operands after, which is 2 and 7. You add them up. The resulting simplified expression would look like:

/ 9 3

And this is the last step you just divide 9 by 3 to get 3.

Here’s another more complex example:

+ - 4 2 \* 2 3

The answer is 8.

We read from right to left and find that the first operator is a multiplication sign. We then look at the operands that correspond to the multiplication sign, which is 2 and 3.

The simplified expression would look like

+ - 4 2 6

Again, look right to left, and we see a subtraction sign. The operands that correspond to the subtraction sign are 4 and 2. We subtract those two values. 4 – 2 = 2.

The simplified expression looks like

+ 2 6

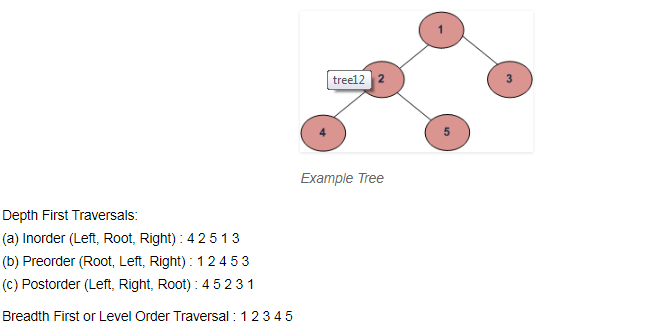
We just add these two numbers to get 8.

1. Tree traversals

There are 4 types of tree traversals

* Inorder
  + This traversal goes through the left node, then the root node (aka the center node), and then the right.
* Preorder
  + Goes to the root first, then the left node, then the right node.
* Post order
  + Goes to left node, then the right node, then the root
* Level order
  + Goes through the highest node, then reads the second layer down from left to right, then the third layer down from left to right, until it reads all nodes.

The following example shows the difference between each traversal.



How I understood this part was when I manually had my finger on the tree, and this gave me more understanding on where the traversal goes. It’s like knowing where to go on a map, once you actually go to that area, you have a much better understanding of the palce rather than you just look on a map.

The inorder traversal is 4 2 5 1 3 because it goes to the most left node, which is 4. It then gets the root of 4, which is 2, then goes to the right of that respective node, which is 5. It goes back up because we finished the left side of the root 1, it then goes to 1, then it goes to the right of that node, which is 3.

This case was confusing the node that contains 1 because it had another tree for its left node, which is why 2 is also considered a root. However, it is only a root when compared to 4 and 5, but is a left node for the root 1, which is why we started at 4.

The preorder traversal went 1 2 4 5 3 because it starts at the main root, which is 1. Then it goes to its left, which is two. However, two is also a root node, so it goes to the left node of 2, which is 4. Now we go to the right node of 2, which is 5. We then go back up to the main node because we finished traversing through the left node, and we go to the right of the main node, which is 3.

Again the confusing part is how the left node of root 1 was also a sub tree in of itself.

The postorder traversal went 4 5 2 3 1. This is because you start at the root node, which is one, but you must find the left node, which is 2. However, 2 also has a left node which is 4. So it goes to 4 first. 2 also has a right node which is 5, so it goes to 5. It then goes to the root node of this subtree which is 2. Now we go to the right side of the root, which is 1, and we go to 3. Then go to the main node which is one.\

The level order is easy to understand. You basically read the nodes as it appears in a book – starting from the top, and ending to the bottom, going from left to right. So, we read the top node which is 1, then we go to the next layer. We read from left to right, getting 2 then 3. Next layer, we read from left to right and we get 4 then 5.

Final words of this book:

This is a brief overview of the topics mentioned. All of the material covered has appeared on written tests. If you actually want to improve, you will have to take practice tests, and go over the answers to improve. Anyway, good luck on your written tests, and I hope this guide helped you out! I spent about 6 hours doing this, and I hope this guide is to stay for more years to come! Thanks to Affan Khurram for helping me revise about 3 sentences.

Again, I hope this book is for future years to come. If you want a specific year this guide was made, it was made in the summer of 2018.

Additionally, do not be disappointed about your test score. Even if someone jokes around about someone’s test score (which can happen with people like Affan and his 162 at UIL), they mean no harm. This ACO club is great, and the journey of improvement is one worth taking. Just adding a little rant to this, people usually tend to hate comp sci as they are taught in schools, which I find very sad because their view of comp sci is limited to quizzes and tests. So even if you may hate your comp sci class, this club is very different and we try to make comp sci more intuitive and engaging as well. This club was built on a few passionate people, and I am so glad that this club is growing, igniting people’s interest regarding comp sci even further.

This is the original author, who wrote all of this, Jason Siu, hoping that you continue to improve and share your knowledge! All of this information isn’t really hard, its more just memorization.