1 a

1. 3
2. 5
3. 8
4. 13
5. 43
6. 38
7. 30
8. 128
9. 255
10. 215

B

1. 111
2. 1010
3. 100001
4. 101010
5. 1100000
6. 1101100
7. 11010110
8. 1111
9. 1000111
10. 10010010

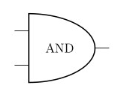
C

1. 10
2. 101
3. 1001
4. 11000
5. 100011

2.

a) A AND B

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



C

A

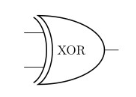
B

b) A XOR B

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

A

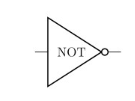
B



C

c) NOT(A OR B)

|  |  |
| --- | --- |
| A | C |
| 0 | 1 |
| 0 | 1 |
| 1 | 0 |
| 1 | 0 |

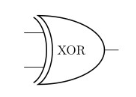
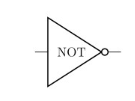


C

A

d) A XOR NOT B

|  |  |  |
| --- | --- | --- |
| A | B(NOT B) | C |
| 0 | 0(1) | 1 |
| 0 | 1(0) | 0 |
| 1 | 0(1) | 0 |
| 1 | 1(0) | 1 |



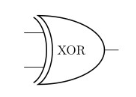
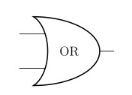
C

A

B

e) A OR B XOR C

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | D |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |



C

A

B

3.

Smart Traffic Intersection

A traffic intersection in a particular city consists of one major street and one minor street.

The city planners have asked us to design the traffic light such that it will account for traffic

on both streets. The light should only be green for the minor street if there is traffic on the

minor street, but no traffic on the major street.

Use the following convention to represent the information about the intersection:

• Bit A: 0 if Major Street has no traffic, 1 if it does.

• Bit B: 0 if Minor Street has no traffic, 1 if it does.

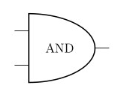
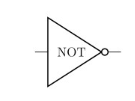
• Bit C: 0 if Major Street has green light, 1 Minor Street has green light.

1. Construct a truth table that properly represents how the intersection should behave.

Important rule(The light should only be green for the minor street): not A and B

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | Not A and B | C(result) |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |

1. Draw a circuit that would implement this logic.



C

A

B

Problem 4: Optional Problem

Algorithm Classes

At the end of the day, we are interested in using computing to solve real world problems.

In order to do this, we need to implement algorithms. The first step in solving a computing

problem is deciding what type of algorithm to use.

Label each of the following situations as one of the common classes of algorithms we discussed

in this lesson (shortest path, sorting/searching, or scheduling):

1. A group of students is organized by height. I want to find the student who is 504”.

‘searching’

b) A group of students needs tutors. Each student needs a tutor for a different amount of

time, and a tutor can only teach one student at a time. I want to assign students and

tutors so that everyone can be tutored in under five hours.

‘scheduling’

c) I am trapped in a maze and want to find my way out in the quickest amount of time.

It takes me the same amount of time to travel down each branch of the maze.

‘shortest path’

Problem 5: Optional Challenge Problem

**Functional Completeness of NAND**

This problem is quite challenging, and dives into the concept of “functional completeness”.

We use Boolean logic gates to implement operations in computing. It turns out that with a

specific set of logic gates, {NOT, OR, AND, XOR}, in combination with the COPY function,

we can construct any possible operation on a set of bits. Because this can be done, this set

of operations is called functionally complete.

There is an additional logic gate that is commonly used when construct logic circuits that

we did not discuss in detail: the NAND gate. The NAND gate comprises of an AND gate

immediately followed by a NOT gate. Here is it’s circuit symbol and corresponding truth

table:

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

A very interesting fact is that the NAND gate is functionally complete by itself. This

means that any conceivable operation can be performed using only NAND gates and COPY.

In this problem we would like to verify the functional completeness of NAND. Our approach

to do this is to show that we can recreate all of the gates in our original set {NOT, OR,

AND, XOR}, which we established as a functionally complete set. If we can use only NAND

and COPY to create all the gates in this set, then NAND is functionally complete. For an

added challenge, you can try to do this without looking at the rest of the problem, but parts

(a) - (d) go through the steps to do construct each gate.

a) First we will tackle the NOT gate. Verify that the following logic circuit produces the

result NOT(A) by making a truth table.

|  |  |  |
| --- | --- | --- |
| A | A1(A2) | C |
| 1 | 1 | 0 |
| 1 | 1 | 0 |
| 0 | 0 | 1 |
| 0 | 0 | 1 |

The result of C is the same as NOT gate.

We have created the NOT gate using NAND! this means that if we want to implement a

NOT gate, we simply insert this construction.

b) Next we can create the AND gate. Doing this requires two steps:

1) Create the AND gate using NAND and NOT

2) Insert the NOT we created using NAND from part (a)

Draw the logic circuit which would implement an AND gate using only NAND.



C

A

B

c) The OR gate is a little bit trickier. Let’s take a look at the truth tables for the NAND

gate and OR gate.

NAND

ABC

001

011

101

110

They are actually very similar! If we have a NAND gate, and swap the input (A=0, B=0) and

(A=1, B=1), then it will result in an OR gate. Try to implement this flipping of inputs, then

draw the logic circuit which would recreate OR. Check the solutions to verify your answer.

(Hint: try applying a NOT gate to A or B before they go into a NAND gate. How does this

affect the truth table?)

C

A

B

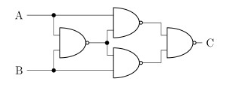


(d) Finally we must create the XOR gate. This one is really tricky, so we will show the

construction. Verify that it successfully reproduces the XOR gate by making a truth table.

This one is quite complex, so make sure to keep track of the input and output of each gate

in the circuit.



0 0 1 1 1 0

0 1 1 1 0 1

1 0 1 0 1 1

1 1 0 1 1 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | NAND1 | NAND2 | NAND3 | Result |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |

So the result is the same as the XOR gate.