Exo 9

Problem 1

$$F: \{0,1\}^n \longrightarrow \{0,1\}^m$$

a) output $\{0,1\}$ $K = 2$

$$F = 2^{2^n}$$

There are 2^n inputs. For each input, we decide if the output is 0 or 1 $\rightarrow$ 2 choices per input

2^{2^n} total possible boolean functions

b) output $\{-1,0,1\}$ $K = 3$

$$F = 3^{2^n}$$

For each ~~ipu~~ input bitstring, you may output $-1, 0, 1$
So there are 3 choices for each of the 2^n inputs $\rightarrow$
3^{2^n} different functions

c) output $\{0,1\}^m$, the ouputs are bitstrings of length m

$$|\{0,1\}^m| = 2^m$$

$K = 2^m$

$$\rightarrow F = (2^m)^{2^n} = 2^{m \cdot 2^n}$$

The tree has 2^n leaves. Each leaf can have any of 2^m labels. So the total number of labelings is
$2^{m \cdot 2^n}$

Problem 2.

A	B	$A \uparrow B$
0	0	1
0	1	1
1	0	1
1	1	1

a) NOT A $= A \uparrow A$

For $A = 0 \rightarrow A \uparrow A = 1$
For $A = 1 \rightarrow A \uparrow A = 0$
} So $A \uparrow A = \neg A$

b)

A AND B $= (A \uparrow B) \uparrow (A \uparrow B)$

$A \uparrow B = \neg (A \wedge B)$

If we NAND that value with itself, we get NOT of it

$$(A \uparrow B) \uparrow (A \uparrow B) = \neg(\neg(A \wedge B)) = A \wedge B$$

So one NAND gives NOT(AND), the second NAND cancels the NOT

c) OR from NAND

Here we use Morgan's law

$$A \vee B = \neg(\neg A \wedge \neg B)$$

NOT A $= A \uparrow A$

NOT B $= B \uparrow B$

A OR B = NOT A $\uparrow$ NOT B ~~A OR B~~

$A \text{ OR } B = (A \uparrow A) \uparrow (B \uparrow B)$

$A \uparrow A = \neg A \qquad B \uparrow B = \neg B$ then

$$(A \uparrow A) \uparrow (B \uparrow B) = \neg(\neg A \wedge \neg B) = A \vee B$$

This shows NAND is functionally complete

From NAND alone we can build AND, OR, NOT and thus any Boolean circuit.

Problem 3

$$F: \{0,1\}^n \to \{0,1\}$$

$$\delta_x(y) = \begin{cases} 1 & \text{if } y = x \\ 0 & \text{otherwise} \end{cases}$$

So $\delta_x(y)$ is 1 only on one specific input $y = x$, and 0 everywhere else

a) Circuit for δ_x of size $O(n)$

let

$$x = (x_1, x_2, \dots, x_n), \quad y = (y_1, y_2, \dots, y_n)$$

We want $\delta_x(y) = 1$ only when all bits of y equal the bits of x.

If $x_i = 1$, we need $y_i = 1$

If $x_i = 0$, we need $y_i = 0$

1. For each position i

if $x_i = 1$

$$\ell_i = y_i$$

if $x_i = 0$

$$\ell_i = \neg y_i$$

2. Then all bits match only if all these literals are 1 at the same time

$$\delta_x(y) = \ell_1 \wedge \ell_2 \wedge \dots \wedge \ell_n$$

So the circuit

Has at most n NOT gates

Uses $n-1$ AND gates to combine $\ell_1, \dots, \ell_n$ into one output

Total number of gates is propositional to $n \to$ size $= O(n)$

B) Circuit for Fusing the δ_x circuits

We use the fact that any Boolean function can be written as an OR of indicator functions

if $F(x) = 1$ then $\delta_x(y)$ should contribute to the output

if $F(x) = 0$, we ignore that x.

$F(y)$ is 1 if there exists some input x with $F(x)=1$ such that y equals x.

That is exactly captured by OR overal all $\delta_x(y)$ where $F(x)=1$

There are at most 2^n different inputs x

For each such x with $F(x)=1$, we have circuit for $\delta_x(y)$ of size $O(n)$ from part(a)

Suppose there are k inputs where $F(x)=1$ (in worst case $k=2^n$)

Then total size of all δ_x circuits togethe is

$$\underline{k \cdot O(n)}$$

To combine them we use a big OR gate over these k outputs. This needs $k-1$ OR gates, which is

$$O(k)$$

So the total size is

$$O(nk)+O(k)=O(nk) \leq O(n \cdot 2^n)$$

Thus any Boolean function $F:\{0,1\}^n \to \{0,1\}$ can be computed by Boolean circuit of size at most $O(n \cdot 2^n)$