Quiz # 3

Dec. 5, 2013

1. **Implementing XOR with AND gates**

   Recall that the Exclusive-OR (XOR) function is 1 if an odd number of the inputs are 1, and 0 otherwise. Suppose that we have to build an circuit that computes XOR with AND gates, OR gates, and inverters.

   (a) Draw a circuit to compute the XOR of 7 variables with the fewest possible number of AND and OR gates gates, and as many inverters as you want.

   (b) Draw a circuit to compute the XOR of 26 variables with the fewest possible number of AND and OR gates gates, and as many inverters as you want.

   (You can have XOR "boxes" in your drawings. You don't have to show the contents of each box, but show the contents of each box of a given size.)

2. **Batcher Sorting Network**

A comparator is a device which sorts two numbers x and y, as shown in Figure 1.

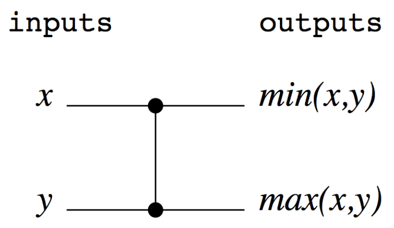

Figure 1: a comparator.

The Batcher sorting network was presented in class. It is constructed with a block called the `Merge` network. Given two sorted input sequences $x_0, x_1, \ldots, x_{n/2-1}$ and $x'_0, x'_1, \ldots, x'_{n/2-1}$, the `Merge[`$n$`]` network produces a sorted output sequence $y_0, y_1, \ldots, y_{n-1}$. The recursive construction of the `Merge[8]` network is shown in Figure 2. The recursive construction of the `Batcher[8]` network, based on the `Merge[8]` network, is shown in Figure 3. The `Merge[2]` and `Batcher[2]` networks both consist of a single balancer.

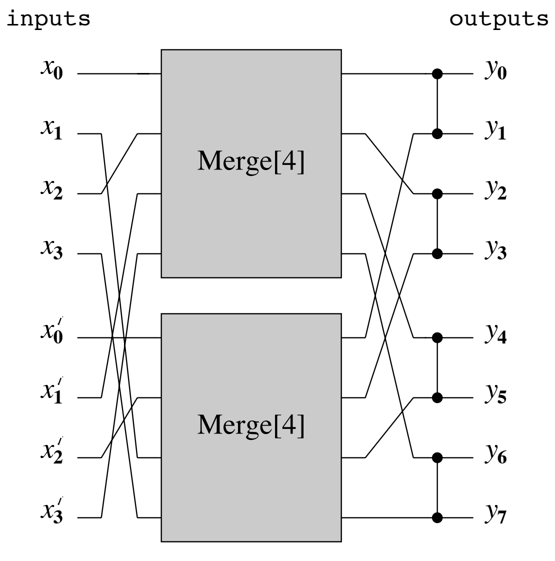

Figure 2: the `Merge[8]` network.

(a) What is the depth of the `Merge[8]` network?
(b) What is the depth of the `Batcher[8]` network?
(c) What is the depth of the `Merge[64]` network?

(d) What is the depth of the `Batcher[64]` network?

(The depth of a network is the number of stages, i.e., the number of comparators that one traverses from an input to an output.)

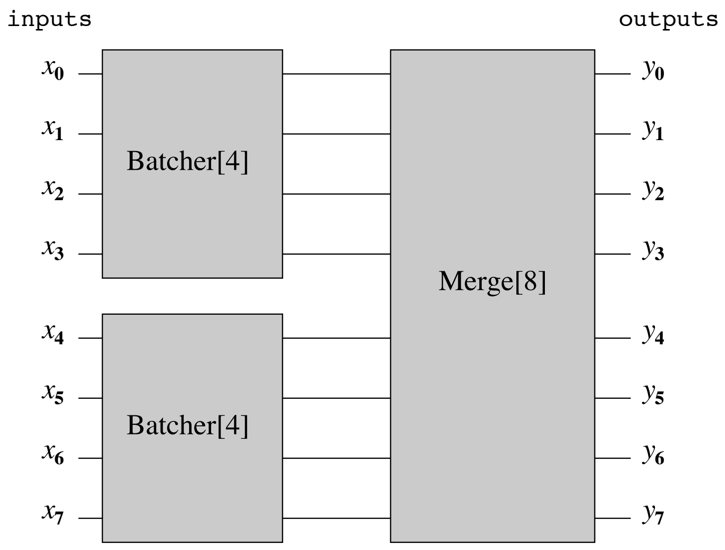

Figure 3: the `Batcher[8]` network.

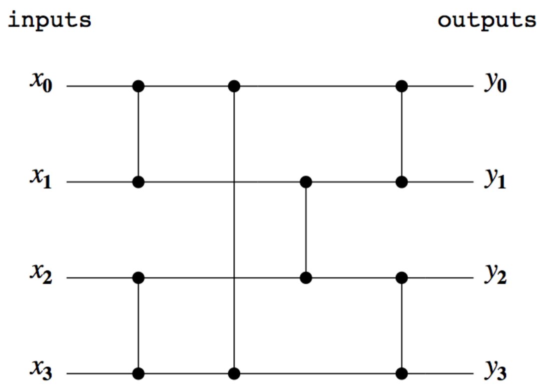

Figure 4: the `Batcher[4]` network.

3. **A Bit of Horticulture – Traversing Trees**

   Consider the following data structure:

```
struct node {
   int x;
   struct node *left;
   struct node *right;
};
```

   The tedious code to setup a tree is shown at the end. There is also a sketch of the corresponding tree.

   (a) What does the following function print out?

```
void dfs(struct node *p) {
   if (p->left != NULL) {
      dfs(p->left);
   }
   printf("%d ", p->x);
   if (p->right != NULL) {
      dfs(p->right);
   }
}
int main(int argc, char **argv) {
   struct node *p = setup_tree();
   dfs(p);
}
```

(b) What does the following function print out?

```
void dfs1(struct node *p) {
   printf("%d\n", p->x);
   if (p->left != NULL) {
      dfs2(p->right);
   }
   if (p->right != NULL) {
      dfs2(p->left);
   }
}

void dfs2(struct node *p) {
   if (p->left != NULL) {
      dfs1(p->left);
   }
   if (p->right != NULL) {
      dfs1(p->right);
   }
   printf("%d\n", p->x);
}

int main(int argc, char **argv) {
   struct node *p = setup_tree();
   dfs1(p);
}
```

Here is the (tedious) code to create the tree.

```
struct node *setup_tree(void) {
// create tree
struct node *p=             (struct node *)malloc(sizeof(struct node));
p->left=                    (struct node *)malloc(sizeof(struct node));
p->right=                   (struct node *)malloc(sizeof(struct node));
p->left->left=              (struct node *)malloc(sizeof(struct node));
p->left->right=             (struct node *)malloc(sizeof(struct node));
p->right->left=             (struct node *)malloc(sizeof(struct node));
p->right->right=            (struct node *)malloc(sizeof(struct node));
p->left->right->left=       (struct node *)malloc(sizeof(struct node));
p->left->right->right=      (struct node *)malloc(sizeof(struct node));
p->right->right->left=      (struct node *)malloc(sizeof(struct node));
p->right->right->right=     (struct node *)malloc(sizeof(struct node));
p->right->right->left->left=(struct node *)malloc(sizeof(struct node));
p->x = 1;
p->left->x = 2;
p->right->x = 3;
p->left->left->x = 4;
p->left->left->left  = NULL;
p->left->left->right = NULL;
p->left->right->x = 5;
p->right->left->x = 6;
p->right->left->left  = NULL;
p->right->left->right = NULL;
p->right->right->x = 7;
p->left->right->left->x = 8;
p->left->right->left->left  = NULL;
p->left->right->left->right = NULL;
p->left->right->right->x = 9;
p->left->right->right->left  = NULL;
p->left->right->right->right = NULL;
p->right->right->left->x = 10;
p->right->right->left->right = NULL;
p->right->right->right->x = 11;
p->right->right->right->left  = NULL;
p->right->right->right->right = NULL;
p->right->right->left->left->x = 12;
p->right->right->left->left->left  = NULL;
p->right->right->left->left->right = NULL;
return p;
}
```

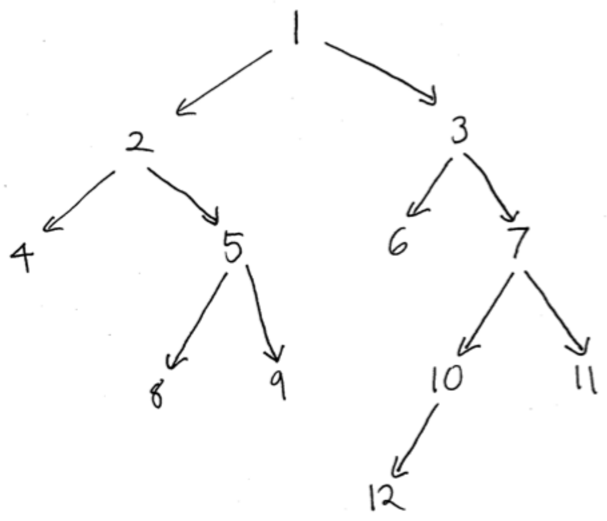

Figure 5: Tree

4. **Pointers**

   (a) What does the following program print out?

```
# include <stdio.h>

int main(int argc, char **argv) {

   int *p;
   int *q;
   int *r;

   int x = 1;
   int y = 2;
   int z = 3;

   p = &x;
   q = &y;
   r = &z;

   *r = x;
   x = *q;
   y = *r;

   printf("%d %d %d\n", x, y, z);

   x = 4;
   y = 5;
   z = 6;

   r = p;
   p = q;
   q = r;

   *p = 7;
   *q = 8;
   *r = 9;

   printf("%d %d %d\n", x, y, z);

}
```

(b) What does the following program print out?

```
# include <stdio.h>

int main(int argc , char **argv) {


   int *p;
   int **q;
   int ***r;

   int x = 1;
   int y = 2;

   p   = &x;
   q   = &p;
   r   = &q;

   *p    = 3;
   **q   = *p;
   ***r = **q;
   printf("%d %d %d %d %d\n", x, y, *p, **q, ***r);

   int z[5] = {2, 3, 4, 5, 6};

   z[z[z[0]]] = 7;

   int i;
   for (i = 0; i < 5; i++) {
      printf("%d ", z[i]);
   }
   printf("\n");
}
```

5. **Error Correcting Codes**

Suppose that Alice wants to send Bob 4 bits of information at a time, x_0, x_1, x_2, x_3, over a noisy Wi-fi connection that occasionally flips bits. She decides to *encode* her information by adding three extra bits x_5, x_6, x_7 computed as follows (here + represents exclusive OR):

$$\begin{aligned} x_4 &= x_1 + x_2 + x_3 \\ x_5 &= x_0 + x_2 + x_3 \\ x_6 &= x_0 + x_1 + x_3 \end{aligned}$$

She sends the 7 bits $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7$ to Bob.

Consider the following matrix, called a *parity check* matrix:

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

The vector of the seven 7 bits that Alice sends, $X = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7]$, satisfies

$$HX^T = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Problems

(a) Suppose that Alice wants to send the bits $1, 1, 1, 1$. What seven bits will she transmit?

(b) Suppose that Bob receives the seven bits $1, 0, 0, 1, 0, 0, 1$. Which bit was flipped? What will he conclude were Alice's original four bits?