1. Suppose we have the following structure definition (also throughout this example assume we use a 64-bit machine):

```
struct node{
    int x;
    float y;
};
```

And then we declare a variable of that structure: `struct node instance;`

(a) [1] Write the C statement that declares `p` as a pointer to the struct node:

```
struct node * p;
```

(b) [1] Write the C statement that make `p` points to `instance`.

```
p = &instance;
```

(c) [1] Using `instance`, write C statement that puts 5 in the field `x`.

```
instance.x = 5;
```

(d) [1] Do the same as (c) above but using `p` instead of `instance`.

```
p->x = 5;
```
2. Given the following C code:

```c
int index(int x){
    int i;

    for(i=31; i>=0; i--){
        if (x & (1<<i))
            return i;
    }
    return -1;
}
```

(a) [2] What will this function return if we call it with `index(0xF00000FF)`?

31

(b) [2] What will this function return if we call it with `index(0x00000001)`?

0

(c) [1] Based on your answer in the above two questions, what does the above code do? (Hint: No credit will be given if you explain the code. So, something like “There is a loop that …” is not accepted as correct answer).

Return the position of the most-significant bit that is equal to 1.
3. Given the following logic circuit:

![Logic Circuit Diagram]

a) [1] What is the logic function implemented by the above circuit?

\[ Y = (AB + C')' \]

b) [8] Fill-up the truth-table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

c) [3] Implement the above function using a decoder (of your choice) and one OR gate. Note that you have to specify the size of the decoder (how many inputs and how many outputs) and the connections with the OR gate.

You simply need a 3x8 decoder and connect outputs: 001, 011, and 101 corresponding to A'B'C, A'BC, and AB'C to the OR gate.
4. Consider the following x86 code fragment for computing the sum of an array of 10 32-bit integers, pointed to by ecx.

```assembly
movl $0, %eax
movl $0, %edx
TOP:
    cmpl $10, %edx
    jl OUT
    add (%ecx, %edx, ), %eax
    inc %edx
    jmp TOP
OUT:
```

(a) [4] There are two bugs in the code. What are they?

<table>
<thead>
<tr>
<th>Bug#</th>
<th>Description</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>jL will cause the loop to exit from first iteration</td>
<td>change jL to jge</td>
</tr>
<tr>
<td>2</td>
<td>The array elements are int so 4 bytes in length each. The add instruction</td>
<td>change (%ecx, %edx,) to (%ecx, %edx,4)</td>
</tr>
<tr>
<td></td>
<td>must multiply the index by 4</td>
<td></td>
</tr>
</tbody>
</table>

(b) [2] What are the values taken by edx during the whole execution of the above code, before you fix the bugs?

0

(c) [2] What are the values taken by edx during the whole execution of the above code, after you fix the bugs?

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
5. Suppose we have an unsorted list $A[]$ of $N$ numbers and we want to write an algorithm to find the minimum of these numbers.

(a) Write the algorithm to do so:

```c
min = A[0];
for( i = 1; i < N; i++)
    if(A[i] < min)
        min = A[i];
```

(b) What is the complexity of that algorithm (i.e. the big-O)?

$O(N)$

(c) Suppose the list is sorted in ascending order, what will be the complexity (do not take the complexity of the sorting operation into account)?

$O(1)$

(d) Suppose the list is sorted in descending order, what will be the complexity (do not take the complexity of the sorting operation into account)?

$O(1)$