You will be asked questions pertaining to fundamental algorithms and programming archetypes: recursion, iteration, and searching. All code is written in C. You may write your answers in any language. Don't worry about syntax; as long as your answer is conceptually correct, you'll get full points.

1. **Recursive Functions** [1.0 pts]

   What is the formula for computing the following sequence of numbers?

   ```
   1 2 0 -1 5 7 -6 -5 28 20 -51 -7 155 16 -315 118 796 -385 -1623 ...
   ```

   Write a *recursive* function that computes the n-th number in the sequence, for any $n > 2$. Assume that the function evaluates to 1 for $n = 1$ and to 2 for $n = 2$.

2. **Iterative Functions** [2.0 pts]

   (a) **Euclid's Algorithm**
   Consider the problem of computing the greatest common divisor (GCD) of two integers. For instance, the GCD of 90 and 198 is 18. The Greek mathematician Euclid described a simple and remarkably efficient procedure for this task in Books VII and X of his Elements. Write iterative code to compute the GCD of two positive integers a and b.

   (b) **Collatz Procedure**
   The *Collatz* conjecture is a famous open problem in mathematics, proposed by Lothar Collatz in 1937. Consider the following iterative procedure. For any positive integer x,

   - if $x = 1$ stop;
   - else if x is odd, let $x = 3x + 1$;
   - else let $x = x/2$.

   The conjecture is that, starting with any positive integer x, the procedure always terminates with $x = 1$. Proving this is evidently difficult. Paul Erdös said about the conjecture: "Mathematics is not yet ready for such problems". He offered a monetary reward of $500 for its solution.

   You are *not* asked to prove the *Collatz* conjecture on this exam. Rather you are asked to write iterative code that computes the *Collatz* procedure. The input to the system is a positive integer x. The output is the sequence of integers that the procedure produces until it hits one. For instance, for an input of $x = 1$, the output sequence should be:

   ```
   27 82 41 124 62 31 94 47 142 71 214 107 322 161 484 242 121 364
   182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790
   395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132
   566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619
   4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232
   4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61
   184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
   ```

3. **Search a Tree** [1.0 pts]

   Consider the following data structure.

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

   The tedious code for the function `setup_tree` is given below. You can ignore the code for that function and just follow this tree:

```
        1
      /   \
     /     \
    /       \
   2         3
 /   \      / \
4     5    6   7
     / \      / \
    8   9    10 11
            /
          12
```

   (a) What does the following code print out?

```
struct node *setup_tree(void);

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);
   printf("\n");
}
```

(b) What does the following code print out?

```
# include <stdio.h>
# include <stdlib.h>

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

struct list {
   struct node *item;
   struct list *next;
};

void bfs(struct node *p)
{
   struct node *q;
   struct list *l, *m, *r, *t;
   l = malloc(sizeof(struct list));
   l->item = p;
   l->next = NULL;
   r = l;
   while(l != NULL) {
      q = l->item;
      if (q->left != NULL) {
         r->next = (struct list *)malloc(sizeof(struct list));
         r->next->item = q->left;
         r->next->next = NULL;
         r = r->next;
      }
      if (q->right != NULL) {
         r->next = (struct list *)malloc(sizeof(struct list));
         r->next->item = q->right;
         r->next->next = NULL;
         r = r->next;
      }
      t = l;
      l = l->next;
      free(t);

      m = l;
      while(m != NULL) {
            printf("%d ", m->item->x);
            m = m->next;
        }
        printf("\n");
    }
}

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

```
# include <stdio.h>
# include <stdlib.h>

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;
}
```