You will be asked questions pertaining to fundamental algorithms and programming archetypes: recursion, searching, and dynamic programming. 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 Sequence** [0.8 pts]

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

   ```
   1 2 1 3 3 4 6 7 10 13 17 23 30 40 53 70 93 123 163 216 286 379 502 665 ...
   ```

   Write a *recursive* function that computes the n-th number in the sequence, for any $n > 2$.

   Solution

$$\begin{aligned} f(0) &= 0 \\ f(1) &= 1 \\ f(2) &= 2 \\ f(n) &= f(n-2) + f(n-3) \end{aligned}$$

```c
# include <stdio.h>

int fibonacci(int n) {
   if (n <= 2) {
       return n;
   } else {
      int m = fibonacci(n-2)+fibonacci(n-3);
      return m;
   }
}

int main(int argc, char **argv) {
   printf("%d\n", fibonacci(atoi(argv[1])));
}
```

2. **Ackermann Function** [0.8 pts]

What will the following C program print out?

```
# include <stdio.h>

int ackermann(int m, int n) {
   printf("%d, %d\n", m, n);
    if (m == 0)
        return n + 1;
    else if (n == 0)
        return ackermann(m - 1, 1);
    else
        return ackermann(m - 1, ackermann(m, n - 1));
}

int main() {
   printf("%d\n", ackermann(2, 1));
}
```

This is a mind-blowing little function. Suppose that I had asked you to compute ackermann(4,4). This equals

$$2^{2^{2^{2^{2^2}}}} - 3,$$

a number far greater than the number of atoms in the observable universe!

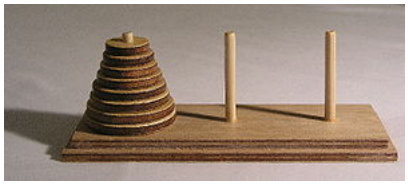
Solution

```
2, 1
2, 0
1, 1
1, 0
0, 1
0, 2
1, 3
1, 2
1, 1
1, 0
0, 1
0, 2
0, 3
0, 4
5
```

3. **Tower of Hanoi** [0.8 pts]

   The Towers of Hanoi game is played with three rods and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks neatly stacked in order of size on the first rod, the smallest at the top. The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:

   - Only one disk may be moved at a time.
   - Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
   - No disk may be placed on top of a smaller disk.

   Consider the following program.

```c
# include <stdio.h>

int hanoi(int n, int a, int c, int b) {
    if (n == 1)
        printf("Move disk from %d to %d\n", a, b);
    else {
        // FILL IN THIS LINE
        printf("Move disk from %d to %d\n", a, b);
        // FILL IN THIS LINE
    }
}

int main(int argc, char **argv)
{
    int n = atoi(argv[1]);
    hanoi(n, 1, 2, 3);
}
```

Suppose that the program prints out the following:

```
./hanoi 4
Move disk from 1 to 3
Move disk from 1 to 3
Move disk from 2 to 1
Move disk from 1 to 3
Move disk from 2 to 1
Move disk from 2 to 1
Move disk from 3 to 2
Move disk from 1 to 3
Move disk from 2 to 1
Move disk from 2 to 1
Move disk from 3 to 2
Move disk from 2 to 1
Move disk from 3 to 2
Move disk from 3 to 2
Move disk from 1 to 3
```

Fill in the missing lines in the program above.

Solution

```
# include <stdio.h>

int hanoi(int n, int a, int c, int b) {
   if (n == 1)
      printf("Move disk from %d to %d\n", a, b);
   else {
      hanoi(n - 1, a, c, b);
      printf("Move disk from %d to %d\n", a, b);
      hanoi(n - 1, c, b, a);
   }
}

int main(int argc, char **argv)
{
   int n = atoi(argv[1]);
   hanoi(n, 1, 2, 3);
}
```

4. **Ninjas and Knapsacks** [0.8 pts]

   Suppose that a ninja has broken into the Imperial Palace in Tokyo to steal valuable trinkets. He will catch the direct return flight with Delta to Minneapolis, but he's very annoyed at Delta's new fees for checked-in luggage. Out of principle, he refuses to pay such fees; everything that he steals from the Imperial Palace must fit in his backpack as carry-on. The tensile strength of his UMN backpack is such that he can safely pack 10 lbs.

   The ninja assesses the value and weight of the trinkets to be as follows:

item	value	weight
1	¥ 3	1
2	¥ 4	3
3	¥ 9	6
4	¥ 8	3
5	¥ 11	4
6	¥ 2	3
7	¥ 4	2
8	¥ 6	1
9	¥ 1	1
10	¥ 7	3

   What is the value of the best choice of trinkets?

   Solution

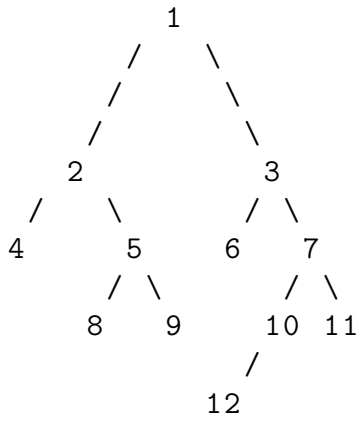
   29

5. **Horticulture** [0.8 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:



(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");
}
```

Solution

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

(b) What does the following code print out?

```
struct node *setup_tree(void);

void dfs1(struct node *p);
void dfs2(struct node *p);

// depth-first search, version 1
void dfs1(struct node *p) {
   if (p->left != NULL) {
      dfs2(p->left);
   }
   printf("%d ", p->x);
   if (p->right != NULL) {
      dfs1(p->right);
   }
}

// depth-first search, version 2
void dfs2(struct node *p) {
  if (p->left != NULL) {
      dfs1(p->left);
   }
   if (p->right != NULL) {
      dfs2(p->right);
   }
   printf("%d ", p->x);
}

int main(int argc, char **argv) {
   struct node *p = setup_tree();
   dfs1(p);
   printf("\n");
   dfs2(p);
   printf("\n");
}
```

Solution

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

```c
# 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;
}
```