CS 2123
Data Structures
Midterm 2

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1</td>
<td></td>
</tr>
<tr>
<td>Question 2</td>
<td></td>
</tr>
<tr>
<td>Question 3</td>
<td></td>
</tr>
<tr>
<td>Question 4</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>

NAME:_________________________

Instructions
1. Do all of the 4 problems
3. You have 50 minutes for the exam
4. Show all your work
5. Do not separate midterm papers

<table>
<thead>
<tr>
<th>Easy</th>
<th>Difficulty Level</th>
<th>Difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐</td>
</tr>
</tbody>
</table>
1. (25 pts) Draw the binary search tree for insertion of numbers in the following order. You do not have to balance the tree.

(a) Insert 4, 6, 5, 2, 3, 7, 1

(b) Insert 6, 4, 5, 7, 2, 3, 1
2. (25 pts) Write a function to split a singly linked list into two sublists. One for the front half, and one for the back half. If the number of elements is odd, the extra element should go in the front list. So Split() on the list \{2, 3, 5, 7, 11\} should yield the lists \{2, 3, 5\} and \{7, 11\}.

List node definition is as follows

    struct node
    {
        int info;
        struct node *next;
    };
    typedef struct node node;

Complete the following function for split

    void split(node *source, node **front, node **back)
    {
    }
3. (25 pts) Span of a binary search tree is the difference between the largest and smallest elements in the binary search tree. Write a function `span()` to compute the span of a binary search tree. Node declaration of the tree is given below.

```c
struct node {
    int key;
    struct node *left, *right;
};
typedef struct node node;
```

Span of the following tree is 16-4=12.

```
  10
 /   \
6     14
 /     /   \
4     8   12 16
```

```c
int span(node *ptr)
{
}
```
4. (25 pts) Write a recursive function to reverse a binary search tree. In a regular binary search tree all the elements on the left subtree have keys less than parent node and all the elements on the right subtree have keys larger than the parent node. After the reverse, all the nodes on the left subtree will have keys larger than parent node and all the elements on the right subtree have keys less than the parent node. Original tree and tree after reverse are shown below as an example.

```c
int reverse(node *ptr)
{

}
```