1. Using the linked list below, give the value of each of the following expressions:

a. p.item

b. q.next.item

c. p.next.next.item

d. ( q.next.item - p.next.item)

e. ( p.next.item >= q.next.next.item )
1. Using the linked list below, give the value of each of the following expressions:

a. \( p.item \) 6

b. \( q.next.item \) -4

c. \( p.next.next.item \) -4

d. \( (q.next.item - p.next.item) \) -17

e. \( (p.next.item >= q.next.next.item) \) true
2. State what the follow code snippets would print out; consider each separately, i.e., starting each time with the linked list shown below. Assume that p and q point to Nodes.

a. 
   p = head;
   q = p.next;
   p = q.next;
   q = p.next.next;
   System.out.println( p.item + " " + q.item );

b. 
   q = head;
   p = q.next;
   while(p != null && p.item < 6) {
      q = p;
      p = p.next;
   }
   System.out.println( p.item + " " + q.item );
2. State what the follow code snippets would print out; consider each separately, i.e., starting each time with the linked list shown below. Assume that p and q point to Nodes.

a.  
   p = head;
   q = p.next;
   p = q.next;
   q = p.next.next;
   System.out.println( p.item + " " + q.item );  
   -4  13

b.  
   q = head;
   p = q.next;
   while(p != null && p.item < 6) {
       q = p;
       p = p.next;
   }
   System.out.println( p.item + " " + q.item );  
   7  -4
3. Show what the linked list below would look like at each of the points (a) and (b) in the following code:

```
p = head;
q = p.next;
p.next = q.next;
q.next = p.next.next;

(a)

p.next = p.next.next;
q = p.next;
p.next = q.next;
p.next.next = q.next.next;

(b)
```

head → 6 → 3 → -4 → 7 → 13 → 2 → .
p = head;
q = p.next;
p.next = q.next;
q.next = p.next.next;

p.next = p.next.next;
q = p.next;
p.next.next = q.next.next;

p = head;
q = p.next;
p.next = q.next;
q.next = p.next.next;
4. Show the result at end of call to `mystery(head)`, showing all nodes and all pointer variables:

```cpp
void mystery(Node h) {
    if(h.next == null)
        return;
    Node p = h.next;
    Node q = h;
    Node r;
    while ( p != null && p.item < 10) {
        r = q;
        q = p;
        p = p.next;
        q.next = r;
    }
    head.next = null;
    // draw diagram of all pointers now!
}
```

head → 3 → 5 → 7 → 8 → 13 → 23 → .
4. Show the result at end of call to `mystery( head )`, showing all nodes and all pointer variables:

```java
void mystery(Node h) {
    if(h.next == null)
        return;
    Node p = h.next;
    Node q = h;
    Node r;
    while ( p != null && p.item < 10 ) {
        r = q;
        q = p;
        p = p.next;
        q.next = r;
    }
    head.next = null;
    // draw diagram of all pointers now!
}
```

Solution:
Apply the following method to this linked list, and (A) show the data structure that would result after calling it as shown, and (B) state in one sentence what this method does on an arbitrary list (int: think about what would happen if the list had one more node).

```java
public static Node conundrum(Node p) {
    if(p == null || p.next == null)
        return p;
    else {
        Node q = p.next;
        p.next = conundrum(q.next);
        q.next = p;
        return q;
    }
}

// Apply the method to the linked list as follows:

head = conundrum( head );
```
5. Solution:

(A)

(B) The method exchanges each pair of nodes, i.e., first with second, third with fourth, and so on. If there is an odd number of nodes it leaves the last node in place.
Apply the following recursive algorithm to this list, and show what it prints out.

```java
void mystery(Node p) {
    if (p == null)
        ;
    else {
        System.out.println(p.item);
        if (p.item > 5) {
            mystery(p.next);
        }
        else {
            mystery(p.next);
            mystery(p.next);
        }
    }
}
```
6.

**SOLUTION:**

6
1
8
2
8
2