Section-1

1. How do you find the middle element of a given linked list?

I chose this question because the solution makes use of a good concept which we all got to know.

Assume the following to be the structure of linked list...

typedef struct node
{
  int value;
  struct node *next;
  struct node *prev;
}mynode;

The concept is very simple. We make use of two pointers called fast pointer and slow pointer.

void getTheMiddle(mynode *head)
{
  mynode *p = head;
  mynode *q = head;

  if(q!=NULL)
  {
    while((q->next)!=NULL && (q->next->next)!=NULL)
    {
      p = p->next;
      q = q->next->next;
    }
  }

  // Now p is at the middle of the list.

  // Print the middle element.
  printf("The middle element is: %d\n", p->value);
}
Here p, the slow pointer, moves one step, whereas q, the fast pointer, moves two steps. So, when q reaches end, p will be at the middle of the linked list.

2. How do you split a given linked list into two halves?

Hope you could guess the answer now. This question too is based on the concept used in the previous question.

```c
void split(struct node* source, struct node** frontRef, struct node** backRef) {
    struct node* fast;
    struct node* slow;
    if (source==NULL || source->next==NULL) {
        // length < 2 cases
        *frontRef = source;
    }
    p=(p!=(mynode *)NULL?p->next:(mynode *)NULL);
    q=(q!=(mynode *)NULL?q->next:(mynode *)NULL);
    q=(q!=(mynode *)NULL?q->next:(mynode *)NULL);
}

printf("The middle element is [%d], \n", p->value);
```


3. Calculate the depth of a tree.

```c
int tree_height(mynode *p)
{
    if(p==NULL)return(0);
    if(p->left){h1=tree_height(p->left);}
    if(p->right){h2=tree_height(p->right);}
    return(1+max(h1,h2));
}
```
return(max(h1,h2)+1);
}

The degree of the leaf is zero. The degree of a tree is the max of its element degrees. A binary tree of height h, h > 0, has at least h and at most \((2^h - 1)\) elements in it. The height of a binary tree that contains n, n>0, elements is at most n and at least \(\log(n+1)\) to the base 2.

\[ \log(n+1) \text{ to the base } 2 = h \]

\[ n = (2^h - 1). \]

4. Print the numbers from 1 to 100 without using loops.

The word that should have lingered in your mind is Recursio.

```c
void printUp(int startNumber, int endNumber) {
    if (startNumber > endNumber)
        return;
    printf("[%d]
", startNumber++);
    printUp(startNumber, endNumber);
}
```
Section-2

1. Does 50[array] and array[50] mean the same?

   The answer is yes. Just know the following too…
   
   \[\text{array}[n] \equiv *((\text{array})+(n)) \equiv *((n)+(\text{array})) \equiv n[\text{array}]\]

2. Will the following code compile? If yes, what is the output? If no, what should be done so that it becomes compilable?

   ```
   main()
   {
       int a=1,b=2;
       printf("%d",a+++++b);
   }
   ```

   No. This code will not compile. The exact expression should be `a++ + ++b`. The white space is required to make this code compilable.

   Here `a++` will remain 1 and `++b` will be 3, so the answer is `1 + 3` which is 4.