Reverse Level Order

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 500

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

struct node *newNode(int data)
{
	struct node *temp = (struct node *)malloc(sizeof(struct node));
	temp->data = data;
	temp->left = temp->right = NULL;
	return temp;
}

struct node **createQueue(int *front, int *rear)
{
	struct node **queue = (struct node **)malloc(sizeof(struct node *)*MAX);
	*front = *rear = 0;
	return queue;
}

void enqueue(struct node **queue, int *rear, struct node *newNode)
{
	queue[(*rear)++] = newNode;
}

struct node *dequeue(struct node **queue, int *front)
{
	return queue[(*front)++];
}

int isQueueEmpty(int *rear, int *front)
{
	return ((*rear) == (*front));
}
struct node **createStack(int *top)
{
	struct node **stack = (struct node **)malloc(sizeof(struct node *)*MAX);
	*top = -1;
	return stack;
}

void push(struct node **stack, struct node *node, int *top)
{
	stack[++(*top)] = node;
```

```
}

struct node *pop(struct node **stack, int *top)
{
	return stack[(*top)--];
}

int isStackEmpty(int *top)
{
	return (*top == -1);
}

void printReverseLevelOrder(struct node *root)
{
	int front, rear, top;
	if(root)
	{
		struct node **queue = createQueue(&front, &rear);
		struct node **stack = createStack(&top);
		struct node *temp;
		enqueue(queue, &rear, root);
		while(!isQueueEmpty(&rear, &front))
		{
			temp = dequeue(queue, &front);
			push(stack, temp, &top);
			if(temp->right)
				enqueue(queue, &rear, temp->right);
			if(temp->left)
				enqueue(queue, &rear, temp->left);
		}
		while(!isStackEmpty(&top))
		{
			temp = pop(stack, &top);
			printf("%d\t", temp->data);
		}
	}
}

int main()
{
	struct node *root=NULL;
	root = newNode(10);
	root->left = newNode(20);
	root->right = newNode(30);
	root->left->left = newNode(40);
	root->right->left = newNode(50);
	root->right->right = newNode(60);
	printReverseLevelOrder(root);
	return 0;
}
```

Time complexity: O(n)

Space complexity is O(n)