Long Questions

1. (50 points) Assume the following ADT of the queue. (Que.h)

```
#ifndef __QUE_H_
#define __QUE_H_
#include <stdio.h>

typedef int Element;

struct stQueue
{
     int iSize;
     // Data required to implement
     // Queue ADT
};
typedef struct stQueque* Queue;

Queue    CreateDeque(); // Creates an empty queue
void     EnqueueInQ(Element e, Queque Q); // Inserts at
    front of the queue
    rear
Element  DequeueInQ(Deque Q); //  Removes the front of the
                                              r
    queue and returns the element
int      GetSize(Queue Q); // Returns the number of
    elements in the queue in O(1)
#endif
```

```
#ifndef __STACK_H
#define __STACK_H
#include <stdio.h>
#include "Que.h"
/*The following Stack ADT needs to be implemented*/
typdef struct stStck Stack;
struct stStck{
    Queue myQforStack;
};

void    Push(Element e, Stack S); // Inserts e into the
    Stack S
Element Pop(Stack S); // Returns the top of the stack S
    and deletes it from the Stack S
#endif
```

i Implement a stack data structure (that is Push and PoP using a single Queue instance and the operations supported above.
(The code should be in C)

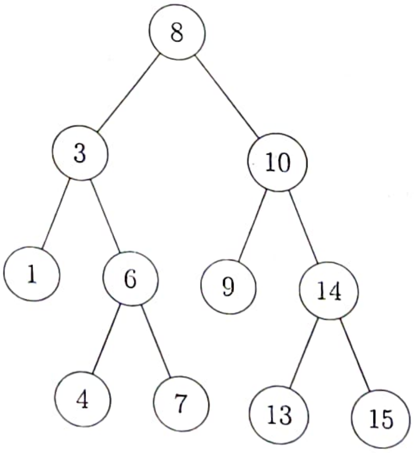
ii What is the complexity of Push and Pop?

iii How much extra memory do you need?
($O(1)$ or $O(N)$, N being the current size of the stack.)?
(HINT: Draw a small diagram queue and stack side by side)

2. (40 points) You have given two elements n_1 and n_2 which are part of an BST T. Write a routine in c language, LCA(BST T, int n_1, int n_2), to find an element in the tree T that is both nodes' lowest common ancestor.

```
#ifndef __BST_H
#define __BST_H


typedef struct stTreeNode* BinTree;
typedef BinTree Position;
typedef BinTree BST;

struct stTreeNode {
     Element Element;
     BinTree Left;
     BinTree Right;
};

// The following code returns the element, the least
    common ancestor for n1 and n2. Note it returns Element
    and not the pointer to Node, which is LCA.
Element LCA(BST T, Element n1, Element n2);

#endif
```

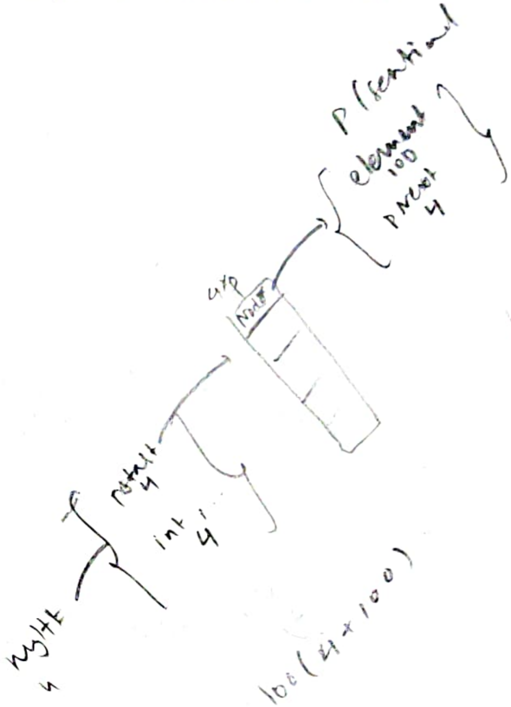
You can assume n_1, n_2 exists in the given T. Thus, there is no need to check for their presence. E.g., Let $T =$



In the above tree, LCA(T,1,4) is 3. LCA(T,9,14) is 10, LCA(T,7,13) is 8.

3. (20 points) Recall Hash Table ADT in the class. The implementation is a sentinel node in the front.

```

#ifndef _HASTABLE_H_

#define _HASTABLE_H

typedef struct stHT * HashTable;
typedef struct stNode * Node;
typedef int Key;
#define  _invalid -5555;

struct stHT{
     int iTableSize;
     Node *pStart;
};

 struct stNode{
     Element iElement;
     Node pNext;
};

HashTable CreateHashTable(int iTableSize);
void InsertHashTable(Element e, HashTable myHt);
#endif
```



Let sizeof(Element) be 100 bytes, and the size of any pointer is 4 bytes. You created HashTable myHT=CreateHashTable(p) where p is some prime number. Then, you made 100 calls to InsertHashTable for myHT. What is the total memory allocated to hold this HashTable? (That is the total memory required to hold this table of size p and 100 data points of type Element. Do not count the 4 bytes required to store the pointer myHT.)

Total memory = ~~Elect~~ 4(p) + p(104) + 100(104) ~~+ 4~~ + 4

Total memory = (108p + 10408) bytes

4. *(30 points) How will insert an* `integer` *in a* BST? Write a **non-recursive** routine *in C.*

```
BST Insert (int x, BST T) {
```