Question 1. [12 marks]

Part (a) [1 mark]

Derive the computational cost $f(n)$ of this code snippet. What is the asymptotic complexity O(f(n)) of this cost (bound should not be too loose)? Provide a proof for this via first principles.

```
void f (int n){
int j = 1;
   while (j <= n){
         n=n/2;
   }
 }
```

Part (b) [1 mark]

Derive the computational cost $f(n)$ of this code snippet. What is the asymptotic complexity O(f(n)) of this cost (bound should not be too loose)? Provide a proof for this via first principles.

```
void f (int n){
      int k=1,m=0, i;
      for(i=0; i<n; i++){
            while(k<n){
                  k=k*n;
            }
            m=m+1;
      }
}
```

Part (c) [1 mark]

Derive the computational cost $f(n)$ of this code snippet. What is the asymptotic complexity O(f(n)) of this cost (bound should not be too loose)? Provide a proof for this via first principles.

```
void f (int n){
      int k = 0, i, j;
      for (i = 1; i *=2; i <= n){
            for (j = 0; j < i; j++){
                  k += 1;
            }
      }
}
```

Part (d) [1 MARK]

Derive the computational cost $f(n)$ of this code snippet. What is the asymptotic complexity O(f(n)) of this cost (bound should not be too loose)? Provide a proof for this via first principles.

```
int F (int n){
      if( n==1)
            return 1;
      else
            return 2 + 2*F(n-1);
}
```

Part (e) [1 MARK]

Derive the computational cost $f(n)$ of this code snippet. What is the asymptotic complexity O(f(n)) of this cost (bound should not be too loose)? Provide a proof for this via first principles.

```
int F (int n){
      int i=0, count=0;
      if( n==1)
            return 1;
      else
            for(i=0; i<n; i++)
                  count=count+1;
      return count+2*F(n-1);
}
```

Part (f) [1 MARK]

Derive the computational cost $f(n)$ of this code snippet. What is the asymptotic complexity O(f(n)) of this cost (bound should not be too loose)? Provide a proof for this via first principles.

```
int F (int n){
      int i=0, count=0;
      if( n==1)
            return 1;
      else
            for(i=0; i<n; i++)
                  count=count+1;
      return count+2*F(n/2);
}
```

Part (g) [1 mark]

Derive the computational cost $f(n)$ of this code snippet. What is the asymptotic complexity O(f(n)) of this cost (bound should not be too loose)? Provide a proof for this via first principles. Hint: k^{th} Fibonacci number is bounded above by 2^k(i.e. $F_k \leq 2^k$).

```
int F (int n){
      int i=0, count;
      if( n==0)
            return 1;
      if( n==1)
           n=n+1;
            return n^2;
      else
       count=n+1;
      return count+F(n-1)+F(n-2);
}
```

Part (h) [1 mark]

Provide a proof for this via first principles. Derive the computational cost $g(n)$ of function `s`. What is the asymptotic complexity O(g(n)) of this cost (bound should not be too loose)? Verify that the cost of function f can be expressed as $C(n,k) = (n+1) + C(k-1) + C(n-k)$. Derive the computational cost $f(n)$ of function `f`. What is the asymptotic complexity O(f(n)) of this cost (bound should not be too loose)? Provide a proof for this via first principles. `A` is an integer array of size `n`. Hint: Harmonic number $H_n = \sum_{i=1}^{n} \frac{1}{i} = In(n) + \gamma + O(\frac{1}{n})$ where $\gamma = 0.57721566$.

```
void s(int * A, int * i, int * j){
      int p, t;
      p=A[(*i+*j)/2];
      do{
            while(A[*i]<p){
                  (*i)++;
            }
            while(A[*j>p){
                  (*j)--;
            }
            if(*i<=*j){
                  t=A[*i];
                  A[*i]=A[*j];
                  A[*j]=t;
                  (*i)++;
                  (*j)--;
            }
      } while(*i <= *j);
}
void f(int * A, int m, int n){
      int i,j;
```

```
        if (m<n){
                i=m; j=n;
                s(A, &i, &j);
                f(A, m, j);
                f(A, i, n);
        }
}
```

Part (i) [1 MARK]

What is the asymptotic complexity of the following $f(n) = 2^{\sqrt{n}log_2 n}$? Bound should not be too loose.

Part (j) [1 MARK]

What is the asymptotic complexity of the following $f(n) = 3n^{\sqrt{n}}$? Bound should not be too loose.

Part (k) [1 MARK]

What is the asymptotic complexity of the following $f(n) = (n+k)^m$ where k is a constant? Bound should not be too loose.

Part (l) [1 MARK]

What is the asymptotic complexity of the following $f(n) = 2^{2n+1}$? Bound should not be too loose.

For the following set of questions, assume that the root pointer `r` points to a valid tree with one or more nodes. Also, assume that `struct Node` has been defined as follows:

```
struct Data {
    int value;
};

struct Node {
    struct Data d;
    struct Node * rChild;
    struct Node * lChild;
};
```

Unless a specific data structure is specified to be used, if needed feel free to use any data structures such as queues, stacks or linked lists and assume that all interface functions (i.e. enqueue, dequeue, push, pop, etc) are available to you. Just specify the function prototypes of these interface functions.

Question 2. [5 marks]

Implement the function `int checkComplete(struct Node * r)` in a recursive manner to check whether a binary tree is complete or not. `r` is the pointer to the root of the tree. Make use of a queue ADT for your implementation.

Question 3. [5 marks]

Implement the function `int checkFull(struct Node * r)` in a recursive manner to check whether a binary tree is full or not. `r` is the pointer to the root of the tree.

Question 4. [5 marks]

Implement the function `int sumRightLeafs(struct Node * r)` in a recursive manner to compute the sum of all right leafs of a binary tree. `r` is the pointer to the root of the tree.

Question 5. [5 marks]

Implement the function `int iPrintPostOrder(struct Node * r)` in an iterative manner to print all elements in a binary tree in post order. One condition is that you must use a stack. `r` is the pointer to the root of the tree.

Question 6. [5 marks]

Suppose that you are provided with an array `A` containing sorted integer elements. Insert nodes into a binary search tree so that the tree remains balanced. Implement the function `int balancedBSTInsertion(int * A, int startIndex, int endIndex)`. `r` is the pointer to the root of the tree. `startIndex` and `endIndex` are integers taking values between 0 and `size` where `size` is a global integer variable denoting the length of the array `A`.

Question 7. [5 marks]

Implement the function `int checkBST(struct Node * r` that checks whether a tree with root `r` is a BST. `r` is the pointer to the root of the tree.

Question 8. [5 marks]

Suppose that you are given the pointer to the root of a tree (implemented via linked lists). Check whether this is a heap or not by implementing the function `int checkHeap(struct Node * r)`.