Assignment 8

Answer 1:

```
#include <stdio.h>
#include <stdlib.h>

struct Node{
   int data;
   struct Node *left,*right;
};

struct Node* create(int x){
   struct Node* p=malloc(sizeof(struct Node));
   p->data=x;
   p->left=p->right=NULL;
   return p;
}

void preorder(struct Node* root){
   if(!root) return;
   printf("%d ",root->data);
   preorder(root->left);
   preorder(root->right);
}

void inorder(struct Node* root){
   if(!root) return;
   inorder(root->left);
   printf("%d ",root->data);
   inorder(root->right);
}

void postorder(struct Node* root){
   if(!root) return;
   postorder(root->left);
   postorder(root->right);
   printf("%d ",root->data);
}
```

Answer 2(a):

```
struct Node* searchRec(struct Node* root,int x){
    if(!root || root->data==x) return root;
    if(x < root->data) return searchRec(root->left,x);
    return searchRec(root->right,x);
}

struct Node* searchItr(struct Node* root,int x){
    while(root){
        if(root->data==x) return root;
        if(x < root->data) root=root->left;
        else root=root->right;
    }
    return NULL;
}
```

Answer 2(b):

```
int maxBST(struct Node* root){
    while(root->right) root=root->right;
    return root->data;
}
```

Answer 2(c):

```
int minBST(struct Node* root){
    while(root->left) root=root->left;
    return root->data;
}
```

Answer 2(d):

```
struct Node* inorderSuccessor(struct Node* root,struct Node* x){
    struct Node* succ=NULL;
    while(root){
        if(x->data < root->data){
            succ=root;
            root=root->left;
        } else root=root->right;
    }
    return succ;
}
```

Answer 2(e):

```
struct Node* inorderPredecessor(struct Node* root,struct Node* x){
    struct Node* pre=NULL;
    while(root){
        if(x->data > root->data){
            pre=root;
            root=root->right;
        } else root=root->left;
    }
    return pre;
}
```

Answer 3(Insert):

```
struct Node* insert(struct Node* root,int x){
    if(!root) return create(x);
    if(x < root->data) root->left=insert(root->left,x);
    else if(x > root->data) root->right=insert(root->right,x);
    return root;
}
```

Answer 3 (Delete):

```
struct Node* minNode(struct Node* root){
    while(root->left) root=root->left;
    return root;
}

struct Node* delete(struct Node* root,int x){
    if(!root) return root;
    if(x < root->data) root->left=delete(root->left,x);
    else if(x > root->data) root->right=delete(root->right,x);
    else{
        if(!root->left){
            struct Node* t=root->right;
            free(root);
            return t;
        }
        else if(!root->right){
            struct Node* t=root->left;
            free(root);
            return t;
        }
        struct Node* t=minNode(root->right);
        root->data=t->data;
        root->right=delete(root->right,t->data);
    }
    return root;
}
```

Answer 3 (Max Depth):

```
int maxDepth(struct Node* root){
    if(!root) return 0;
    int l=maxDepth(root->left);
    int r=maxDepth(root->right);
    return (l>r?l:r)+1;
}
```

Answer 3 (Min Depth):

```
int minDepth(struct Node* root){
    if(!root) return 0;
    int l=minDepth(root->left);
    int r=minDepth(root->right);
    if(!root->left) return r+1;
    if(!root->right) return l+1;
    return (l<r?l:r)+1;
}
```

Answer 4:

```
int isBSTUtil(struct Node* root,int min,int max){
    if(!root) return 1;
    if(root->data<=min || root->data>=max) return 0;
    return isBSTUtil(root->left,min,root->data) &&
           isBSTUtil(root->right,root->data,max);
}

int isBST(struct Node* root){
    return isBSTUtil(root,-1000000,1000000);
}
```

Answer 5:

```
void heapify(int a[],int n,int i){
    int largest=i;
    int l=2*i+1;
    int r=2*i+2;
    if(l<n && a[l]>a[largest]) largest=l;
    if(r<n && a[r]>a[largest]) largest=r;
    if(largest!=i){
        int t=a[i]; a[i]=a[largest]; a[largest]=t;
        heapify(a,n,largest);
    }
}

void heapSort(int a[],int n){
    for(int i=n/2-1;i>=0;i--) heapify(a,n,i);
    for(int i=n-1;i>=0;i--){
        int t=a[0]; a[0]=a[i]; a[i]=t;
        heapify(a,i,0);
    }
}
```

Answer 6:

```
int size=0;

void push(int a[],int x){
    int i=size++;
    a[i]=x;
    while(i>0 && a[(i-1)/2] < a[i]){
        int t=a[i]; a[i]=a[(i-1)/2]; a[(i-1)/2]=t;
        i=(i-1)/2;
    }
}

int pop(int a[]){
    if(size==0) return -1;
    int root=a[0];
    a[0]=a[--size];
    heapify(a,size,0);
    return root;
}
```