Tree Traversal:

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

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

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

	return(node);
}

void printPostorder(struct node* node)
{
	if (node == NULL)
		return;

	printPostorder(node->left);

	printPostorder(node->right);

	printf("%d ", node->data);
}

void printInorder(struct node* node)
{
	if (node == NULL)
		return;

	printInorder(node->left);
```

```
        printf("%d ", node->data);

        printInorder(node->right);
}

void printPreorder(struct node* node)
{
        if (node == NULL)
                return;

        printf("%d ", node->data);

        printPreorder(node->left);

        printPreorder(node->right);
}

int main()
{
        struct node *root = newNode(1);
        root->left                        = newNode(2);
        root->right              = newNode(3);
        root->left->left  = newNode(4);
        root->left->right = newNode(5);

        printf("\nPreorder traversal of binary tree is \n");
        printPreorder(root);

        printf("\nInorder traversal of binary tree is \n");
        printInorder(root);

        printf("\nPostorder traversal of binary tree is \n");
        printPostorder(root);

        getchar();
        return 0;
}
```

Inoreder traversal on binary search tree:

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

```
#include<stdlib.h>

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

int max(int inorder[], int strt, int end);

struct node* newNode(int data);

struct node* buildTree (int inorder[], int start, int end)
{
    if (start > end)
        return NULL;

    int a = max (inorder, start, end);

    struct node *root = newNode(inorder[a]);

    if (start == end)
        return root;

    root->left  = buildTree (inorder, start, a-1);
    root->right = buildTree (inorder, a+1, end);

    return root;
}

int max (int arr[], int strt, int end)
{
    int a, max = arr[strt], maxind = strt;
    for(a = strt+1; a <= end; a++)
    {
        if(arr[a] > max)
        {
            max = arr[a];
            maxind = a;
        }
    }
    return maxind;
}

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

    return node;
}

void printInorder (struct node* node)
{
    if (node == NULL)
        return;

    printInorder (node->left);

    printf("%d ", node->data);

    printInorder (node->right);
}

int main()
{

    int inorder[] = {5, 10, 40, 30, 28};
    int len = sizeof(inorder)/sizeof(inorder[0]);
    struct node *root = buildTree(inorder, 0, len - 1);

    printf("\n Inorder traversal of the constructed tree is \n");
    printInorder(root);
    return 0;
}
```

Binary Search:

```
#include <stdio.h>
int main()
{
  int a, first, last, middle, b, find, array[100];
  printf("Enter total elements to be in array\n");
  scanf("%d", &b);

  printf("Enter %d integers\n", b);

  for (a = 0; a < b; a++)
    scanf("%d", &array[a]);

  printf("Enter value to find\n");
  scanf("%d", &find);

  first = 0;
  last = b - 1;
  middle = (first+last)/2;

  while (first <= last) {
    if (array[middle] < search)
      first = middle + 1;
    else if (array[middle] == find) {
      printf("%d found at location %d.\n", find, middle+1);
      break;
    }
    else
      last = middle - 1;

    middle = (first + last)/2;
  }
  if (first > last)
    printf("Not found! %d isn't present in the list.\n", find);

  return 0;
}
```

Linear Search:

```
#include <stdio.h>
int main()
{
  int array[100], find, a, b;

  printf("Enter number of elements should be in array\n");
  scanf("%d", &b);

  printf("Enter %d integer(s)\n", b);

  for (a = 0; a < b; a++)
    scanf("%d", &array[a]);

  printf("Enter a number to find\n");
  scanf("%d", &find);

  for (a = 0; a < b; a++)
  {
    if (array[a] == find)
    {
      printf("%d is present at location %d.\n", find, a+1);
      break;
    }
  }
  if (a == b)
    printf("%d isn't present in the array.\n", find);

  return 0;
}
```

Depth First Search:

```
#include<stdio.h>

void DFS(int);
int A[20][20],visited[20],c;

void main()
{
    int a,b;
    printf("Enter number of vertices:");

            scanf("%d",&c);

            printf("\nEnter adjecency matrix of the graph:");
```

```
       for(a=0;a<c;a++)
    for(b=0;b<c;b++)
                      scanf("%d",&A[a][b]);

  for(a=0;a<c;a++)
     visited[a]=0;

   DFS(0);
}

void DFS(int a)
{
   int b;
       printf("\n%d",a);
   visited[a]=1;

       for(b=0;b<c;b++)
    if(!visited[b]&&A[a][b]==1)
       DFS(b);
}
```

Breadth Search First:

```
#include<stdio.h>
int Z[30][30]t[30]={0},n,visited[20]={0},a,b,x=0,r=-1;
void BFS(int v)
{
for(a=0;a<n;a++)
if(Z[v][a]&&visited[a]==0)
t[++r]=a;
if(x<=r)
{
visited[t[x]]=1;
BFS(t[f++]);
}
}
void main()
{
int v;
printf("Enter number of vertices: ");
scanf("%d",&n);
printf("\nEnter Graph data in matrix form :\n ");
for(a=0;a<n;a++)
{
for(b=0;b<n;b++)
scanf("%d",&Z[a][b]);
}
printf("\nEnter the start vertex: ");
scanf("d",&v);
BFS(v);
printf("\nReachable nodes are : ");
for(a=0;a<n;a++)
{
if(visited[a])
printf("%d\t",a);
else{
printf("Unable to reach all nodes.BFS impossible");
break;
    }
  }
}
```