Data Structures and Algorithm Lab

By – Tushya Gupta
Roll no – UE233106
CSE Sec2 Group 6

Q1 Create an array to traverse it insert at end, start and a given index, delete at end, start and given index and search for a given element in the array.

Code –

```
#include <iostream>
using namespace std;

class Arr {
private:
  int a[100];
  int l;
  int ins;

public:
  Arr() {
    l = sizeof(a) / sizeof(a[0]);
    cout << "Enter number of elements to enter: ";
    cin >> ins;
    if (ins > l) {
      ins = l;
      cout << "Max length of array is -> " << l << "\n";
    }
    for (int i = 0; i < ins; i++) {
      cout << i << ": ";
      cin >> a[i];
    }
  }
  void trav() {
    for (int i = 0; i < ins; i++) {
      cout << a[i] << " ";
    }
    cout << endl;
  }
  void insert_at_end(int ele) {
    a[ins] = ele;
    if (ins != l) {
      ins += 1;
    }
  }
  void insert_at_start(int ele) {
    if (ins != l) {
      for (int i = ins - 1; i >= 0; i--) {
        a[i + 1] = a[i];
      }
```

```
  } else {
    for (int i = ins - 2; i >= 0; i--) {
      a[i + 1] = a[i];
    }
  }
  a[0] = ele;
  if (ins != l) {
    ins += 1;
  }
}
void insert_at(int ele, int ind) {
  if (ind + 1 > ins) {
    a[ins] = ele;
  } else {
    if (ins != l) {
      for (int i = ins - 1; i >= ind; i--) {
        a[i + 1] = a[i];
      }
    } else {
      for (int i = ins - 2; i >= ind; i--) {
        a[i + 1] = a[i];
      }
    }
    a[ind] = ele;
  }
  if (ins != l) {
    ins += 1;
  }
}
void delete_at_end() {
  if (ins != 0) {
    a[ins - 1] = 0;
    ins -= 1;
  } else {
    cout << "No elements in array" << endl;
  }
}
void delete_at_start() {
  if (ins != 0) {
    for (int i = 0; i < ins - 1; i++) {
      a[i] = a[i + 1];
    }
    a[ins - 1] = 0;
    ins -= 1;
  } else {
    cout << "No elements in array" << endl;
  }
```

```
  }
  void delete_at(int ind) {
   if (ins != 0) {
    if (ind + 1 > ins) {
     a[ins - 1] = 0;
    } else {
     for (int i = ind; i < ins - 1; i++) {
      a[i] = a[i + 1];
     }
     a[ins - 1] = 0;
    }
    ins -= 1;
   } else {
    cout << "No elements in array" << endl;
   }
  }
  void search(int term) {
   for (int i = 0; i < ins; i++) {
    if (a[i] == term) {
     cout << term << " at index " << i << "\n";
     return;
    }
   }
   cout << term << " not found" << "\n";
  }
};

int main() {
  Arr yay;
  yay.trav();
  cout << "Inserting 5 at start" << endl;
  yay.insert_at_start(5);
  yay.trav();
  cout << "Inserting 10 at end" << endl;
  yay.insert_at_end(10);
  yay.trav();
  cout << "Inserting 20 at index 3" << endl;
  yay.insert_at(20, 3);
  yay.trav();
  cout << "Deleting element at index 3" << endl;
  yay.delete_at(3);
  yay.trav();
  cout << "Deleting element at start" << endl;
  yay.delete_at_start();
  yay.trav();
  cout << "Deleting element at end" << endl;
  yay.delete_at_end();
```

```
  yay.trav();
  cout << "Search for 10" << endl;
  yay.search(10);
  return 0;
}
```

Output –

```
Enter number of elements to enter: 5
0: 1
1: 2
2: 3
3: 4
4: 5
1 2 3 4 5
Inserting 5 at start
5 1 2 3 4 5
Inserting 10 at end
5 1 2 3 4 5 10
Inserting 20 at index 3
5 1 2 20 3 4 5 10
Deleting element at index 3
5 1 2 3 4 5 10
Deleting element at start
1 2 3 4 5 10
Deleting element at end
1 2 3 4 5
Search for 10
10 not found
```

Q2 Create a stack to pop and push an element

Code –

```
#include <iostream>
using namespace std;

class Stack {
private:
  int top;
  int stack[10];

public:
  Stack() { top = 0; }
  int peek() { return stack[top - 1]; }
  int pop() {
    if (top == 0) {
      cout << "Stack underflow";
      return 0;
    }
    top--;
    return stack[top];
  }
  void push(int ele) {
    if (top == 10 - 1) {
      cout << "Stack overflow";
      return;
    }
    stack[top] = ele;
    top++;
  }
  bool empty() {
    if (top == 0) {
      return true;
    }
    return false;
  }
  void trav() {
    for (int i = 0; i < top; i++) {
      cout << stack[i] << " ";
    }
    cout << endl;
  }
};

int main() {
```

```
  Stack s;
  cout << "Pushing 1" << endl;
  s.push(1);
  cout << "Pushing 20" << endl;
  s.push(20);
  cout << "Pushing 25" << endl;
  s.push(25);
  cout << "Peeking into stack: ";
  cout << s.peek() << endl;
  cout << "Is stack empty: ";
  if (s.empty()) {
    cout << "Yes" << endl;
  } else {
    cout << "No" << endl;
  }
  cout << "Popping once" << endl;
  s.pop();
  cout << "Popping twice" << endl;
  s.pop();
  cout << "Traversing stack" << endl;
  s.trav();
  return 0;
}
```

Output –

```
Pushing 1
Pushing 20
Pushing 25
Peeking into stack: 25
Is stack empty: No
Popping once
Popping twice
Traversing stack
1
```

Q3 Create a program to convert infix expression to postfix expression.

Code –

```
#include <iostream>
#include <string.h>
#include <unordered_map>
using namespace std;

class Stack {
private:
  int top;
  char stack[10];

public:
  Stack() { top = 0; }
  char peek() { return stack[top]; }
  char pop() {
    top--;
    return stack[top + 1];
  }
  void push(int ele) {
    stack[top + 1] = ele;
    top++;
  }
  bool empty() {
    if (top == 0) {
      return true;
    }
    return false;
  }
};

bool com_pred(char top, char curr, unordered_map<char, int> pred) {
  if (top == '(') {
    return false;
  }
  if (pred[top] >= pred[curr]) {
    return true;
  }
  return false;
}

int main() {
  unordered_map<char, int> pred;
  pred['-'] = 1;
```

```
  pred['+'] = 1;
  pred['/'] = 2;
  pred['*'] = 2;
  pred['^'] = 3;
  pred['('] = 4;
  string infix;
  cin >> infix;
  string postfix;
  Stack s;
  for (int i = 0; i < infix.size(); i++) {
    char sy = infix[i];
    if (sy == ')') {
      while (s.peek() != '(') {
        postfix += s.pop();
      }
      s.pop();
    } else {
      if (!pred[sy]) {
        postfix += sy;
      } else {
        while (!s.empty() && com_pred(s.peek(), sy, pred)) {
          char t = s.pop();
          postfix += t;
        }
        s.push(sy);
      }
    }
  }
  while (!s.empty()) {
    postfix += s.pop();
  }
  cout << postfix;
  return 0;
}
```

Output –

```
a+b-c*d/e+f*g-h
ab+cd*e/-fg*+h-⏎
```

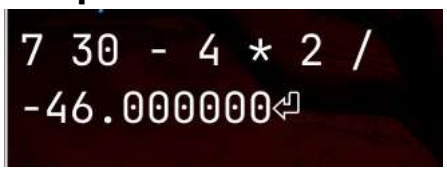
Q4 Create a program to evaluate a postfix expression.

Code –

```
#include <iostream>
#include <math.h>
#include <regex>
#include <string.h>
#include <unordered_map>
using namespace std;

class Stack {
private:
  int top;
  string stack[10];

public:
  Stack() { top = 0; }
  string peek() { return stack[top]; }
  string pop() {
    top--;
    return stack[top + 1];
  }
  void push(string ele) {
    stack[top + 1] = ele;
    top++;
  }
  bool empty() {
    if (top == 0) {
      return true;
    }
    return false;
  }
};

int main() {
  string postfix;
  getline(cin, postfix);
  Stack s;
  unordered_map<string, int> pred;
  pred["-"] = 0;
  pred["+"] = 1;
  pred["/"] = 2;
  pred["*"] = 3;
  pred["^"] = 4;
  regex delim("\\s+");
  sregex_token_iterator tokenIterator(postfix.begin(), postfix.end(), delim,
```

```
                                -1);
  sregex_token_iterator endIterator;
  while (tokenIterator != endIterator) {
    string sy = *tokenIterator;
    if (pred.count(sy) == 0) {
      s.push(sy);
    } else {
      float op_2 = stof(s.pop());
      float op_1 = stof(s.pop());
      float value;
      int v = pred[sy];
      if (v == 0) {
        value = op_1 - op_2;
      } else if (v == 1) {
        value = op_1 + op_2;
      } else if (v == 2) {
        value = op_1 / op_2;
      } else if (v == 3) {
        value = op_1 * op_2;
      } else if (v == 4) {
        value = pow(op_1, op_2);
      }
      s.push(to_string(value));
    }
    ++tokenIterator;
  }
  string result = s.pop();
  cout << result;
  return 0;
}
```

Output –



Q5 To implement insertion and deletion in a linear queue using an array.

Code –

```
#include <iostream>
using namespace std;

#define MAX 10

class Queue {
private:
  int front;
  int rear;
  int arr[MAX];

public:
  Queue() { front = rear = -1; }
  void insert(int x) {
    if (rear >= MAX - 1) {
      cout << "Queue overflow" << endl;
    } else if (rear == -1) {
      front = rear = 0;
      arr[rear] = x;
    } else {
      rear++;
      arr[rear] = x;
    }
  }
  void del() {
    if (front == -1) {
      cout << "Queue underflow" << endl;
    } else if (front == rear) {
      front = rear = -1;
    } else {
      front++;
    }
  }
};

int main() {
  Queue q;
  q.del();
  q.insert(5);
  q.insert(10);
  q.del();
  q.del();
```

```
  q.del();
  cout << "Inserting 10 elements" << endl;
  q.insert(5);
  q.insert(5);
  q.insert(5);
  q.insert(5);
  q.insert(5);
  q.insert(5);
  q.insert(5);
  q.insert(5);
  q.insert(5);
  q.insert(5);
  cout << "Inserting 11th element" << endl;
  q.insert(5);
  return 0;
}
```

```
Queue underflow
Queue underflow
Inserting 10 elements
Inserting 11th element
Queue overflow
```

Q6 To implement insertion and deletion in a circular queue using an array.

Code –

```
#include <iostream>
using namespace std;

#define MAX 10

class CirQueue {
private:
  int front;
  int rear;
  int arr[MAX];

public:
  CirQueue() { front = rear = -1; }
  void insert(int x) {
    if ((front == 0 && rear == MAX - 1) || (rear + 1 == front)) {
      cout << "Queue overflow" << endl;
    } else if (front == -1) {
      front = rear = 0;
      arr[rear] = x;
    } else if (rear != MAX - 1) {
      rear++;
      arr[rear] = x;
    } else if (front != 0, rear == MAX - 1) {
      rear = 0;
      arr[rear] = x;
    }
  }
  void del() {
    if (front == -1) {
      cout << "Queue underflow" << endl;
    } else if (front != -1 && front == rear) {
      front = rear = -1;
    } else if (front != -1 && front == MAX - 1) {
      front = 0;
    } else {
      front++;
    }
  }
};

int main() {
  CirQueue q;
```

```
 q.del();
 q.insert(5);
 q.insert(10);
 q.del();
 q.del();
 q.del();
 cout << "Inserting 10 elements" << endl;
 q.insert(5);
 q.insert(5);
 q.insert(5);
 q.insert(5);
 q.insert(5);
 q.insert(5);
 q.insert(5);
 q.insert(5);
 q.insert(5);
 q.insert(5);
 cout << "Inserting 11th element" << endl;
 q.insert(5);
 q.del();
 q.insert(5);
 q.insert(5);
 return 0;
}
```

```
Queue underflow
Queue underflow
Inserting 10 elements
Inserting 11th element
Queue overflow
Queue overflow
```

Q7 Insertion at the beginning, at the end of a singly linear linked list(SLL), displaying a SLL, insertion after and before a specified location, delete of first node, last node and deletion of a node whose info is given.

Code –

```
#include <iostream>
using namespace std;

class Node {
public:
  int val;
  Node *next;
  Node() { next = NULL; }
  Node(int v) {
    val = v;
    next = NULL;
  }
};

class SLL {
private:
  Node *start;

public:
  SLL() { start = NULL; }
  void insert_begin(int v) {
    Node *nn = new Node(v);
    nn->next = start;
    start = nn;
  }
  void insert_end(int v) {
    if (start == NULL) {
      this->insert_begin(v);
      return;
    }
    Node *nn = new Node(v);
    Node *temp = start;
    while (temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = nn;
  }
  void insert_after(int loc, int v) {
    Node *nn = new Node(v);
```

```
  Node *temp = start;
  for (int i = 0; i < loc - 1; i++) {
    if (temp == NULL) {
      cout << "Location out of length of SLL" << endl;
      return;
    }
    temp = temp->next;
  }
  if (temp == NULL) {
    cout << "Location out of length of SLL" << endl;
    return;
  } else {
    nn->next = temp->next;
    temp->next = nn;
  }
}
void insert_before(int loc, int v) {
  if (loc == 1) {
    this->insert_begin(v);
    return;
  }
  Node *nn = new Node(v);
  Node *temp = start;
  Node *prev;
  for (int i = 0; i < loc - 1; i++) {
    if (temp == NULL) {
      cout << "Location out of length of SLL" << endl;
      return;
    }
    prev = temp;
    temp = temp->next;
  }
  if (temp == NULL) {
    cout << "Location out of length of SLL" << endl;
    return;
  } else {
    nn->next = temp;
    prev->next = nn;
  }
}
void delete_begin() {
  if (start != NULL) {
    Node *r = start;
    start = start->next;
    free(r);
  } else {
    cout << "List has no elements" << endl;
```

```
  }
 }
 void delete_end() {
  if (start == NULL) {
   cout << "List is empty" << endl;
   return;
  }
  if (start->next == NULL) {
   Node *r = start;
   start = NULL;
   free(r);
   return;
  }
  Node *temp = start;
  Node *prev;
  while (temp->next != NULL) {
   prev = temp;
   temp = temp->next;
  }
  prev->next = NULL;
  free(temp);
 }
 void del(int r) {
  if (start == NULL) {
   cout << "List empty" << endl;
   return;
  }
  if (start->val == r) {
   this->delete_begin();
   return;
  }
  Node *temp = start;
  Node *prev;
  while (temp->next != NULL && temp->val != r) {
   prev = temp;
   temp = temp->next;
  }
  if (temp->val == r) {
   if (temp == start) {
    start = start->next;
    return;
   }
   prev->next = temp->next;
   free(temp);
  } else {
   cout << "Node not found" << endl;
  }
```

```
  }
  void display() {
    Node *temp = start;
    while (temp != NULL) {
      cout << temp->val << " -> ";
      temp = temp->next;
    }
    cout << "NULL" << endl;
  }
};

int main() {
  SLL l;
  cout << "Inserting 5 to 9 in reverse order at begining\n";
  l.insert_begin(5);
  l.insert_begin(6);
  l.insert_begin(7);
  l.insert_begin(8);
  l.insert_begin(9);
  l.display();
  cout << "Inserting 10 at the end\n";
  l.insert_end(10);
  l.display();
  cout << "Deleting 2 at begining and ending each\n";
  l.delete_begin();
  l.delete_begin();
  l.delete_end();
  l.delete_end();
  l.display();
  cout << "Inserting 11 after 2\n";
  l.insert_after(2, 11);
  l.display();
  cout << "Inserting 12 after 3\n";
  l.insert_after(3, 12);
  l.display();
  cout << "Inserting 13 before 1\n";
  l.insert_before(1, 13);
  l.display();
  cout << "Inserting 14 before 2\n";
  l.insert_before(2, 14);
  l.display();
  cout << "Deleting over the limit\n";
  l.delete_end();
  l.delete_end();
  l.delete_end();
  l.delete_end();
  l.delete_end();
```

```
 l.delete_end();
 l.delete_end();
 l.delete_end();
 l.display();
 cout << "Inserting 10 at end\n";
 l.insert_end(10);
 l.display();
 cout << "Inserting 32 at begining\n";
 l.insert_begin(32);
 l.display();
 cout << "Deleting over the limit\n";
 l.delete_begin();
 l.delete_begin();
 l.delete_begin();
 l.display();
 cout << "Inserting 10 before 1\n";
 l.insert_before(1, 10);
 l.display();
 cout << "Inserting 5 to 9 in reverse order at begining\n";
 l.insert_begin(5);
 l.insert_begin(6);
 l.insert_begin(7);
 l.insert_begin(8);
 l.insert_begin(9);
 l.display();
 cout << "Deleting node 5, 6 and 7\n";
 l.del(5);
 l.del(6);
 l.del(7);
 l.display();

 return 0;
}
```

```
Inserting 5 to 9 in reverse order at begining
9 -> 8 -> 7 -> 6 -> 5 -> NULL
Inserting 10 at the end
9 -> 8 -> 7 -> 6 -> 5 -> 10 -> NULL
Deleting 2 at begining and ending each
7 -> 6 -> NULL
Inserting 11 after 2
7 -> 6 -> 11 -> NULL
Inserting 12 after 3
7 -> 6 -> 11 -> 12 -> NULL
Inserting 13 before 1
13 -> 7 -> 6 -> 11 -> 12 -> NULL
Inserting 14 before 2
13 -> 14 -> 7 -> 6 -> 11 -> 12 -> NULL
Deleting over the limit
List is empty
List is empty
NULL
Inserting 10 at end
10 -> NULL
Inserting 32 at begining
32 -> 10 -> NULL
Deleting over the limit
List has no elements
NULL
Inserting 10 before 1
10 -> NULL
Inserting 5 to 9 in reverse order at begining
9 -> 8 -> 7 -> 6 -> 5 -> 10 -> NULL
Deleting node 5, 6 and 7
9 -> 8 -> 10 -> NULL
```

Q8 Implement push and pop operations in stack using linked lists

Code –

```
#include <iostream>
using namespace std;

class Node {
public:
  Node() { this->next = NULL; }
  Node(int v) {
    this->val = v;
    this->next = NULL;
  }
  int val;
  Node *next;
};

class Stack {
private:
  Node *start;

public:
  Stack() { start = NULL; }
  void push(int v) {
    Node *nn = new Node(v);
    nn->next = start;
    start = nn;
  }
  void pop() {
    if (start == NULL) {
      cout << "Stack underflow" << endl;
      return;
    }
    Node *r = start;
    start = start->next;
    free(r);
  }
  int top() {
    if (start == NULL) {
      cout << "Stack empty" << endl;
      return 0;
    }
    return start->val;
  }
};
```

```
int main() {
  Stack s;
  s.push(5);
  s.push(6);
  s.push(7);
  s.push(8);
  cout << s.top() << endl;
  s.pop();
  cout << s.top() << endl;
  s.pop();
  cout << s.top() << endl;
  s.pop();
  cout << s.top() << endl;
  s.pop();
  s.pop();
  return 0;
}
```

```
8
7
6
5
Stack underflow
```

Q9 Implement inserting and deletion in a queue using a linked list

Code –

```
#include <iostream>
using namespace std;

class Node {
public:
  Node() { this->next = NULL; }
  Node(int v) {
    this->val = v;
    this->next = NULL;
  }
  int val;
  Node *next;
};

class Queue {
private:
  Node *head;
  Node *tail;

public:
  Queue() {
    head = NULL;
    tail = NULL;
  }
  void enque(int v) {
    Node *nn = new Node(v);
    if (head == NULL) {
      head = nn;
      tail = nn;
    } else {
      tail->next = nn;
      tail = nn;
    }
  }
  void deque() {
    if (head == NULL) {
      cout << "Queue underflow" << endl;
      return;
    }
    if (head == tail) {
      Node *r = head;
      head = NULL;
```

```
    tail = NULL;
    free(r);
    return;
  }
  Node *r = head;
  head = head->next;
  free(r);
 }
 void trav() {
  if (head == NULL) {
    cout << "Queue is empty" << endl;
    return;
  }
  Node *n = head;
  while (n->next != NULL) {
    cout << n->val << "<-";
    n = n->next;
  }
  cout << n->val << endl;
 }
};

int main() {
 Queue q;
 q.enque(5);
 q.enque(10);
 q.enque(15);
 q.trav();
 q.deque();
 q.deque();
 q.deque();
 q.trav();
 q.enque(5);
 q.enque(10);
 q.enque(15);
 q.enque(20);
 q.trav();
 q.deque();
 q.deque();
 q.trav();
 q.enque(25);
 q.enque(30);
 q.trav();

 return 0;
}
```

```
5←10←15
Queue is empty
5←10←15←20
15←20
15←20←25←30
```

Q10 Implement search, insertion, deletion and non-recursive traversal in a BST

Code –

```
#include <iostream>
#include <queue>
#include <stack>
using namespace std;

class Node {
public:
  int val;
  Node *left;
  Node *right;
  Node *parent;
  Node() {
    left = NULL;
    right = NULL;
    parent = NULL;
  }
  Node(int a) {
    val = a;
    left = NULL;
    right = NULL;
    parent = NULL;
  }
};
class BST {
private:
  Node *root;
  void printBT(const std::string &prefix, const Node *node, bool isLeft) {
    if (node != NULL) {
      cout << prefix;

      cout << (isLeft ? "├──" : "└──");

      // print the value of the node
      cout << node->val << endl;

      // enter the next tree level - left and right branch
      printBT(prefix + (isLeft ? "│   " : "    "), node->left, true);
      printBT(prefix + (isLeft ? "│   " : "    "), node->right, false);
    }
  }
  void printBT(const Node *node) { printBT("", node, false); }
  Node *max_node(Node *node) {
```

```
  if (node->right != NULL) {
    return max_node(node->right);
  } else {
    return node;
  }
}
Node *min_node(Node *node) {
  if (node->left != NULL) {
    return max_node(node->left);
  } else {
    return node;
  }
}
void insert(int a, Node *node) {
  if (node->val > a) {
    if (node->left == NULL) {
      Node *nn = new Node(a);
      node->left = nn;
      nn->parent = node;
      return;
    }
    insert(a, node->left);
  } else if (node->val <= a) {
    if (node->right == NULL) {
      Node *nn = new Node(a);
      node->right = nn;
      nn->parent = node;
      return;
    }
    insert(a, node->right);
  }
}
void del_node(Node *node) {
  if (node->left == NULL && node->right == NULL) {
    if (node->val > node->parent->val) {
      node->parent->right = NULL;
    } else {
      node->parent->left = NULL;
    }
    free(node);
  } else if (node->left != NULL) {
    Node *max = max_node(node->left);
    node->val = max->val;
    if (max->left == NULL) {
      if (max->parent == node) {
        max->parent->left = NULL;
      } else {
```

```
      max->parent->right = NULL;
     }
     free(max);
    } else {
     del_node(max);
    }
  } else if (node->left == NULL && node->right != NULL) {
    Node *min = min_node(node->right);
    node->val = min->val;
    if (min->right == NULL) {
     if (min->parent == node) {
      min->parent->right = NULL;
     } else {
      min->parent->left = NULL;
     }
     free(min);
    } else {
     del_node(min);
    }
  }
}
void inorder(Node *node) {
  if (node == NULL) {
    return;
  }
  inorder(node->left);
  cout << node->val << " ";
  inorder(node->right);
}
void DFT(Node *node) {
  if (node == NULL) {
    return;
  }
  cout << node->val << " ";
  DFT(node->left);
  DFT(node->right);
}
void postorder(Node *node) {
  if (node == NULL) {
    return;
  }
  postorder(node->left);
  postorder(node->right);
  cout << node->val << " ";
}
void BFT(queue<Node *> q) {
  if (q.empty()) {
```

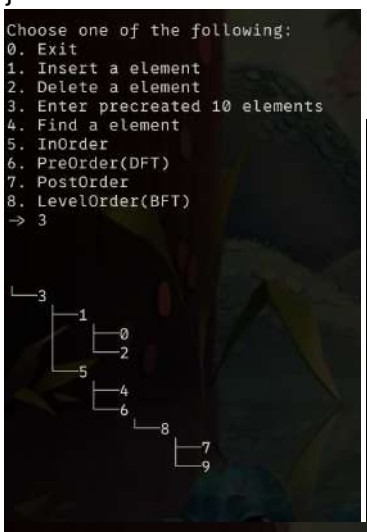
```
    cout << endl;
    return;
  }
  Node *n = q.front();
  cout << n->val << " ";
  q.pop();
  if (n->left != NULL) {
    q.push(n->left);
  }
  if (n->right != NULL) {
    q.push(n->right);
  }
  BFT(q);
}
void BFTiter(queue<Node *> q) {
  Node *node;
  while (!q.empty()) {
    node = q.front();
    cout << node->val << " ";
    q.pop();
    if (node->left != NULL) {
      q.push(node->left);
    }
    if (node->right != NULL) {
      q.push(node->right);
    }
  }
  cout << endl;
}
void DFTiter(stack<Node *> s) {
  Node *node;
  while (!s.empty()) {
    node = s.top();
    cout << node->val << " ";
    s.pop();
    if (node->right != NULL) {
      s.push(node->right);
    }
    if (node->left != NULL) {
      s.push(node->left);
    }
  }
  cout << endl;
}

public:
 BST() { root = NULL; }
```

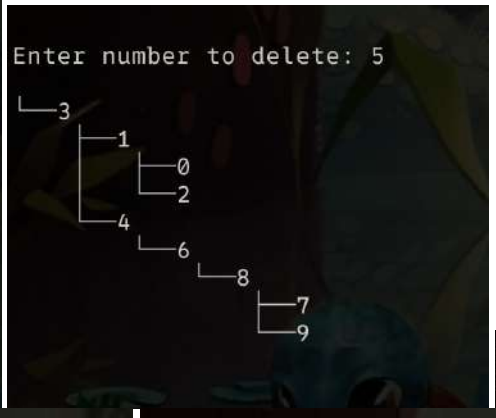
```
void insert(int a) {
  if (root == NULL) {
    Node *nn = new Node(a);
    root = nn;
    return;
  }
  insert(a, root);
}
void dis() { printBT("", root, false); }
void find_and_del(int a) {
  Node *node = root;
  while (node->val != a) {
    if (node->val > a) {
      node = node->left;
    } else {
      node = node->right;
    }
    if (node == NULL) {
      cout << "Node not found" << endl;
      return;
    }
  }
  del_node(node);
}
void find(int a) {
  Node *node = root;
  while (node->val != a) {
    if (node->val > a) {
      node = node->left;
    } else {
      node = node->right;
    }
    if (node == NULL) {
      cout << "Node not found" << endl;
      return;
    }
  }
  cout << "Node found" << endl;
}
void inorder() {
  inorder(root);
  cout << endl;
  cout << endl;
  dis();
}
void postorder() {
  postorder(root);
```

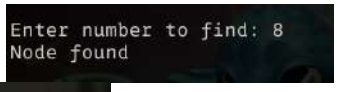
```
    cout << endl;
    cout << endl;
    dis();
  }
  void DFT() {
    cout << "Recurcive: \n";
    DFT(root);
    cout << endl;
    cout << "Iterative: \n";
    if (root == NULL) {
      return;
    }
    stack<Node *> s;
    s.push(root);
    DFTiter(s);
    cout << endl;
    dis();
  }
  void BFT() {
    if (root == NULL) {
      return;
    }
    queue<Node *> q;
    q.push(root);
    cout << "Recurcive: \n";
    BFT(q);
    cout << "Iterative: \n";
    queue<Node *> q1;
    q1.push(root);
    BFTiter(q1);
    cout << endl;
    dis();
  }
};

int main() {
  BST b;
  int i, c = 0;
  int a[10] = {3, 5, 1, 2, 6, 4, 8, 7, 9, 0};
  while (true) {
    cout << "\n\nChoose one of the following:\n";
    cout << "0. Exit\n";
    cout << "1. Insert a element\n";
    cout << "2. Delete a element\n";
    cout << "3. Enter precreated 10 elements\n";
    cout << "4. Find a element\n";
    cout << "5. InOrder\n";
```

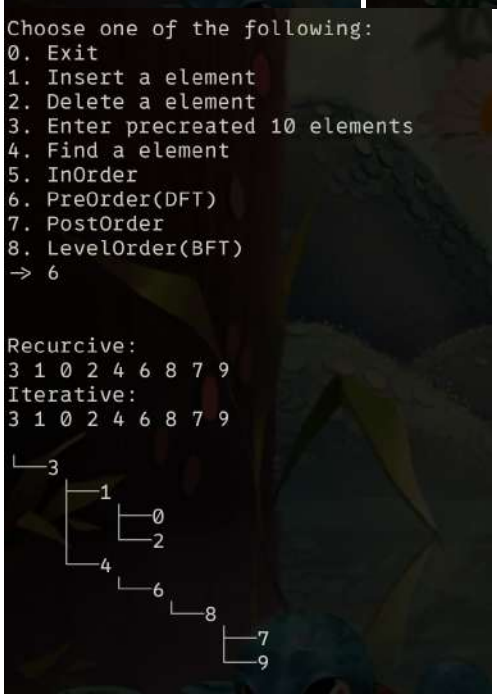
```
cout << "6. PreOrder(DFT)\n";
cout << "7. PostOrder\n";
cout << "8. LevelOrder(BFT)\n";
cout << "-> ";
cin >> c;
cout << "\n\n";
switch (c) {
case 0:
  return 0;
case 1:
  cout << "\nEnter number to insert: ";
  cin >> i;
  b.insert(i);
  cout << endl;
  b.dis();
  cout << endl;
  break;
case 2:
  cout << "\nEnter number to delete: ";
  cin >> i;
  b.find_and_del(i);
  cout << endl;
  b.dis();
  cout << endl;
  break;
case 3:
  for (int j = 0; j < 10; j++) {
    b.insert(a[j]);
  }
  cout << endl;
  b.dis();
  cout << endl;
  break;
case 4:
  cout << "\nEnter number to find: ";
  cin >> i;
  b.find(i);
  break;
case 5:
  b.inorder();
  break;
case 6:
  b.DFT();
  break;
case 7:
  b.postorder();
  break;
```

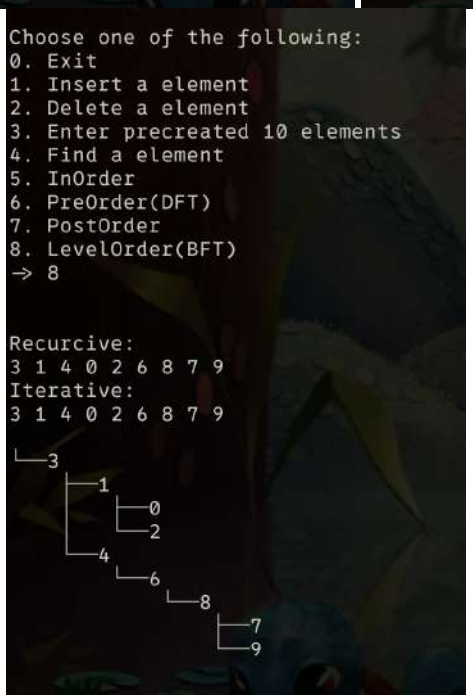
```
    case 8:
      b.BFT();
      break;
    default:
      continue;
    }
  }
  return 0;
}
```











Q11 Implement merge sort

Code –

```
#include <iostream>
using namespace std;

int main() {
 int na, nb;
 cout << "Enter length for array a: ";
 cin >> na;
 int a[na];
 for (int i = 0; i < na; i++) {
  cin >> a[i];
 }
 cout << "Enter length for array b: ";
 cin >> nb;
 int b[nb];
 for (int i = 0; i < nb; i++) {
  cin >> b[i];
 }
 int c[na + nb];
 int i = 0, j = 0, p = 0;
 while (i < na && j < nb) {
  if (a[i] < b[j]) {
   c[p] = a[i];
   p++;
   i++;
  } else {
   c[p] = b[j];
   p++;
   j++;
  }
 }
 if (i >= na) {
  while (j < nb) {
   c[p] = b[j];
   p++;
   j++;
  }
 } else if (j >= nb) {
  while (i < na) {
   c[p] = a[i];
   p++;
   i++;
  }
 }
```

```
  for (int i = 0; i < na + nb; i++) {
    cout << c[i] << " ";
  }
  return 0;
}
```

Q12 Implement quick sort

Code –

```
#include <iostream>
#include <vector>
using namespace std;

int partition(int (&a)[], int l, int h) {
  int pivot = a[l];
  int j = h, i = l + 1;
  while (j >= i) {
    while (a[j] > pivot && j > l) {
      j--;
    }
    while (a[i] < pivot && i <= h) {
      i++;
    }
    if (i >= j) {
      break;
    }
    swap(a[i], a[j]);
  }
  swap(a[l], a[i - 1]);
  return i - 1;
}

void quicksort(int (&a)[], int l, int h) {
  if (l < h) {
    int p = partition(a, l, h);
    quicksort(a, l, p - 1);
    quicksort(a, p + 1, h);
  }
}

int main() {
  int n;
  cout << "Enter length of array: ";
  cin >> n;
  int a[n];
  for (int i = 0; i < n; i++) {
    cin >> a[i];
  }

  quicksort(a, 0, n - 1);

  for (int i = 0; i < n; i++) {
```

```
  cout << a[i] << " ";
 }
 return 0;
}
```