Vinay Ruhil
BSc(H) Computer Science
16115

DS Ques Paper 1

1 (b)

Array	Linked List
(i) Contiguous memory allocation	(i) Non-contiguous memory allocation
(ii) Fixed size	(ii) Dynamic size
(iii) Typically simpler to implement	(iii) Requires more complex implementation

(c) 23, 43, 56, 12, 87, 14, 86

(i) 23 and 43 → no swap

(ii) 43 and 56 → no swap

(iii) 56 and 12 → 1 exchange

23, 43, 12, 56, 87, 14, 86

(iv) 56 and 87 → no swap

(v) 87 and 14 → 1 exchange

23, 43, 12, 56, 14, 87, 86

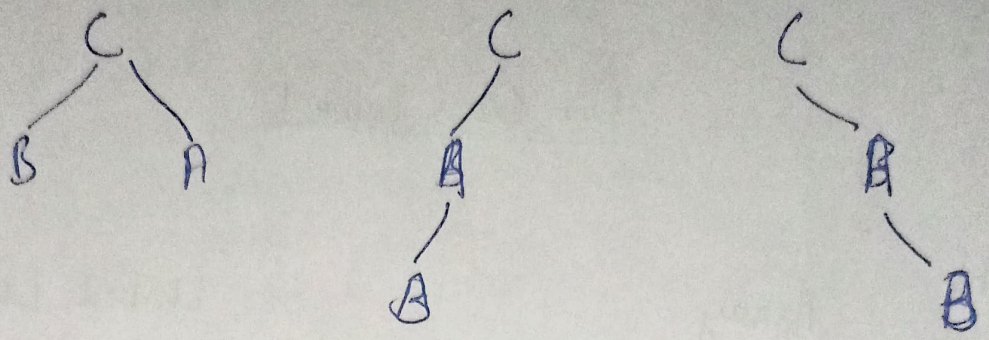
(vi) 87 and 86 → 1 exchange

23, 43, 12, 56, 14, 86, 87

In first pass, total exchanges = 3

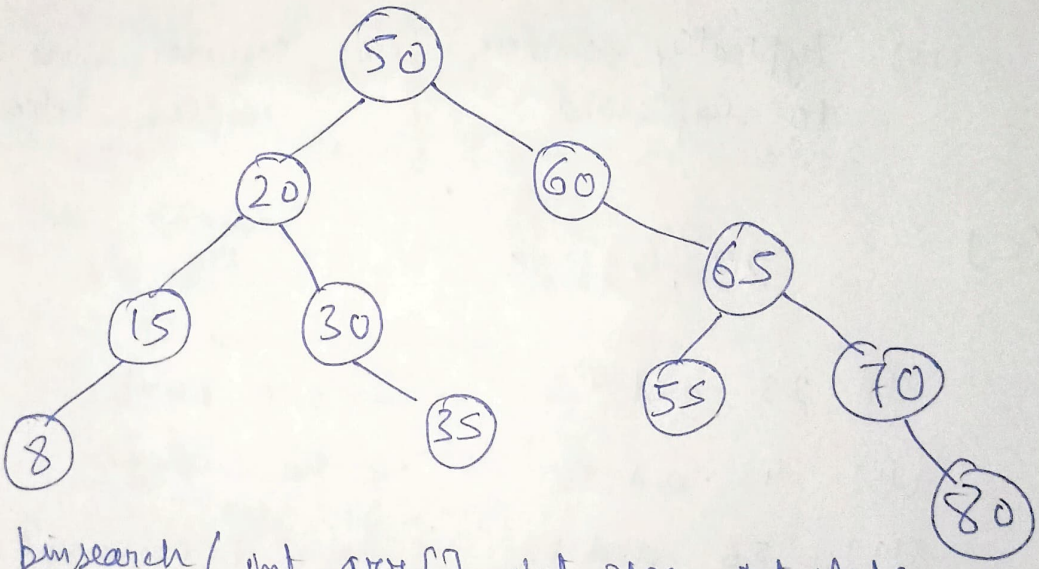
(d) Prefix : ^ - ^ + ^ BC - DE + FG

(e)



(f) Output at fun(6, 3) = 0

(h)



(i)

```
int binsearch(int arr[], int size, int ele){
    int low, mid, high;
    low = 0;
    high = size - 1;
    while(low <= high){
        mid = (low + high)/2;
        if(arr[mid] == ele){
            return mid;
        }
        if(arr[mid] < ele){
            low = mid + 1;
        }
        else{
            high = mid - 1;
        }
    }
    return -1;
}
```

(k)

```
Node * reverse (Node* head){
    Node * curr = head;
    Node * prev = NULL;
    while (curr != NULL){
        Node * nextptr = curr->next;
        curr -> next = prev;
        prev = curr;
        curr = nextptr;
    }
    Node * new_head = prev;
    return new_head;
}
```

2. (a)

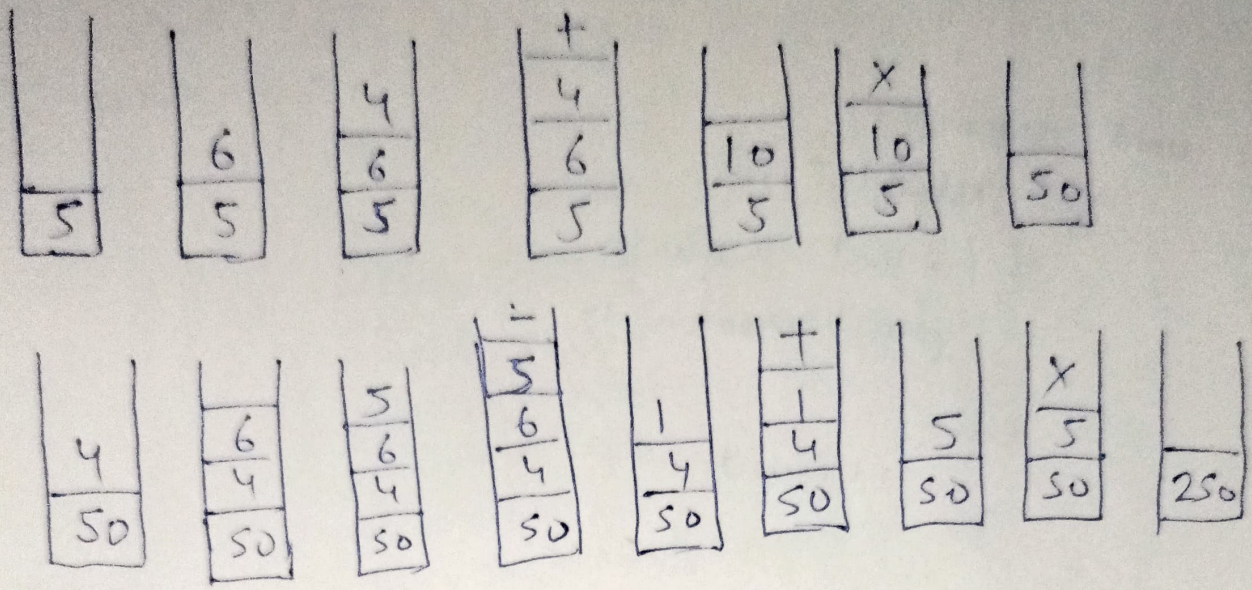
```
# include <iostream>
using namespace std;
template < typename T , int max size>
class stack {
    public:
    T arr [max size];
    int top;
    stack (){
        top = -1;
    }
    bool isEmpty (){
        return top == 1;
    }
    void push (T x){
        if ( top == max size - 1){
            cout << "Stack full";
            return;
        }
        arr [++top] = x;
    }
```

```
T Top(){
    if (top==-1){
        cout<<"No element";
        return 0;
    }
    return arr[top];
}

void pop(){
    if (!isEmpty()){
        --top;
    }
};

template <typename T, int maxSize>
class Queue {
    T arr[maxsize]
    int front, rear;
public:
    Queue (){
        front=-1;
        rear=-1;
    }
bool isEmpty(){
    return front==-1;
}
void enqueue(T x){
    if (rear==maxSize-1){
        cout<<"Queue Overflow";
    }
    else{
        arr[++rear]=x;
        if (front==-1){
            front=0;
        }
    }
}
```

```
T peek(){
    if(!isEmpty()){
        return arr[front];
    }
}
void dequeue(){
    if(!isEmpty()){
        if(front == rear){
            front = rear = -1;
        }
        else{
            ++front;
        }
    }
};
void reverse(Stack<T, maxSize> &stk){
    Queue<T, maxSize> qu;
    while(!stk.isEmpty()){
        qu.enqueue(stk.top());
        stk.pop();
    }
    while(!qu.isEmpty()){
        stk.push(qu.peek());
        qu.dequeue();
    }
};
```

(b) B ^ C + x (^ B - + x

B = 5, ^ = 6, C = 4



3.

```
template < class T >
class Node {
    public:
        T ele;
        Node * next;
        Node * prev;
    Node (const T n){
        ele = n;
        next = NULL;
        prev = NULL;
    }
};
class LinkedList {
    public:
        Node * head;
        LinkedList (){
            head = NULL;
        }
```

```
void deleteOddpos(Node* head){
    Node* curr = head;
    Node* prev = NULL;
    int pos = 1;
while (curr != NULL){
    if(pos%2 == 1){
        if(prev == NULL){
            Node* temp = curr;
            head = curr->next;
            curr = head;
            delete temp;
        } else {
            prev->next = curr->next;
            delete curr;
            curr = prev->next;
        }
    } else {
        prev = curr;
        curr = curr->next;
    }
    pos++;
}
}
```

4.

```
int calculatelength(Node* head){
    if(head == NULL){
        return 0;
    }
    return 1 + calculatelength(head->next);
}
```

5 (a)

Counting no. of right childrens

```
int countRight ( Node * node ){
    if ( node == NULL ){
        return 0;
    }
    int rightCount = countRight ( node -> right );
    return rightCount + 1 + countRight ( node -> left );
}
```

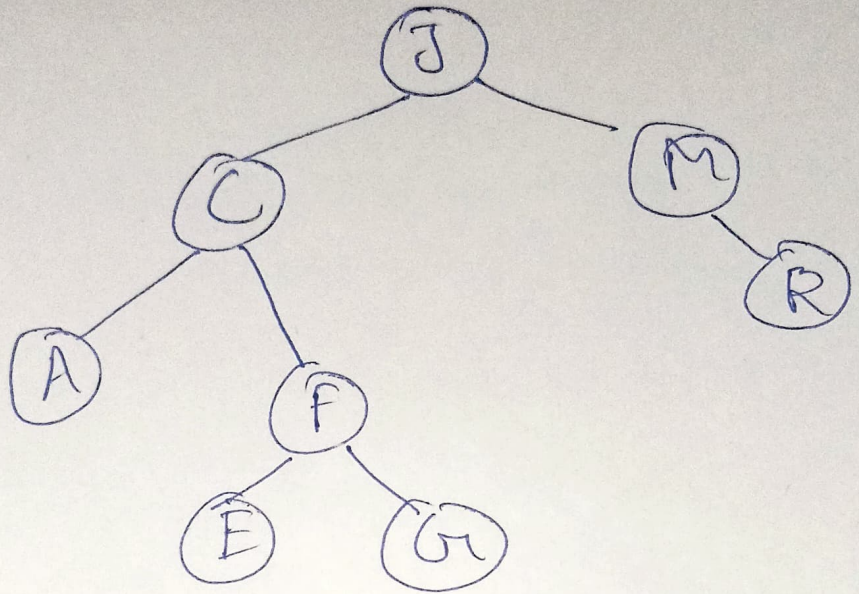
Calculating height of the tree

```
int calcHeight ( Node * node ){
    if ( node == NULL ){
        return 0;
    }
    int leftHeight = calculateHeight ( node -> left );
    int rightHeight = calculateHeight ( node -> right );
    return 1 + max ( leftHeight , rightHeight );
}
```

(b)

Breorder : J C A E G F M R

Inorder : A C E F G J M R



7(b)

12, 14, 11, 16, 7, 8

Pass 1

1. Compare 14 and 12 -> no swap

Pass 2

1. Compare 11 and 14 -> swap (14, 11)

12, 11, 14, 16, 7, 8

2. Compare 11 and 12 -> swap (12, 11)

11, 12, 14, 16, 7, 8

Pass 3

1. Compare 16 and 14 -> no swap

Pass 4

1. compare 7 and 16 -> swap (16, 7)

11, 12, 14, 7, 16, 8

2. swap (14, 7)

11, 12, 7, 14, 16, 8

3. swap (12, 7)

11, 7, 12, 14, 16, 8

4. swap (11, 7)

7, 11, 12, 14, 16, 8

Pass 5

1. compare 16 and 8 swap (16, 8)

7, 11, 12, 14, 8, 16

2. swap (14, 8)

3. swap (12, 8)

4. swap (11, 8)

7, 8, 11, 12, 14, 16

Quiz Paper - 2

1. (a)

```
template <class T>
class Node {
public:
    T elem;
    Node* next;
    Node(const T n){
        elem = n;
        next = NULL;
    }
};
class CircularLL{
public:
    Node* head;
    CircularLL(){
        head = NULL;
    }

void deletion(int pos){
    if(pos == 1){
        deleteathead();
        return;
    }
    Node* temp = head;
    int count = 1;
    while(count != pos - 1){
        temp = temp -> next;
        count++;
    }
    Node* todelete = temp -> next;
    temp -> next = temp -> next -> next;
    delete todelete; }
```

(b)

(i) Sorting can help in this case. If the words are sorted, checking if they are anagrams becomes a matter of comparing sorted version of words.

(ii) Yes, sorting can help in finding the min. value. If array is sorted is ascending order, the min. value is at the beggining of the array.

(iii) NO, sorting does not help in computing mean of values. Mean involves adding up all values and dividing by the number of values. Sorting doesn't change the sum or the count of values, so time complexity remains the same.

(iv) Yes, sorting can help. If array is sorted, finding the median is simpler. For odd-sized array, the median is the middle value, and for even sized array, it is the average of the two middle values.

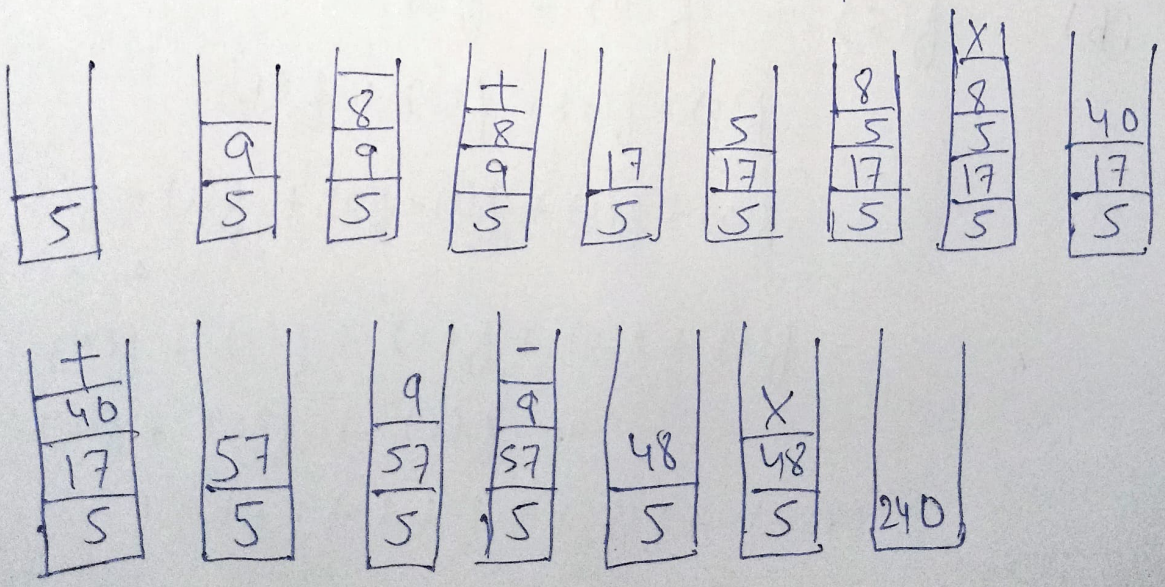
(v) No sorting doesn't help. Finding the mode involves counting ocurrance of each value, and sorting may disrupt the order, making it harder to count.

(c) Recursive function for binary search

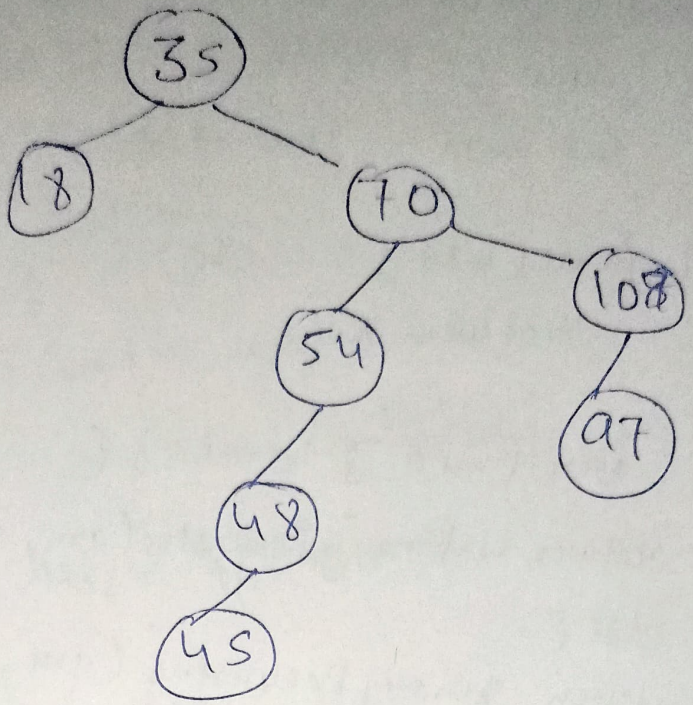
```
int binaryRecursive(int arr[], int low, int high, int ele){
  if (low <= high){
    int mid = low + (high - low) / 2;
    if (arr[mid] == ele){
      return mid;
    }
    if (arr[mid] < ele){
      return binaryRecursive(arr, mid+1, high, ele);
    } else{
      return binaryRecursive(arr, low, mid-1, ele);
    }
  }
  return -1;
}
```

(d) B A C + B C X + A - X

B = 5, A = 9, C = 8



(e) 35, 70, 54, 48, 18, 108, 97, 45



Preorder : 35, 18, 70, 54, 108, 48, 45, 97

Inorder : 18, 35, 45, 48, 54, 70, 97, 108

Post order : 18, 45, 48, 54, 97, 108, 70, 35

2.

(b) $f(5) = f(4) + f(3)$

$= f(3) + f(2) + f(2) + f(1)$

$= f(2) + f(1) + f(1) + f(0) + f(1) + f(0) + f(1)$

$= f(1) + f(0) + f(1) + f(1) + f(0) + f(1) + f(0) + f(1)$

$\therefore 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1$

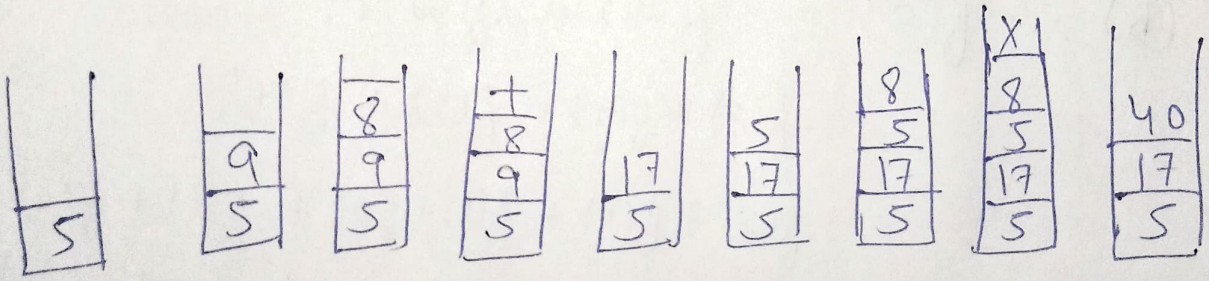
$= 8$

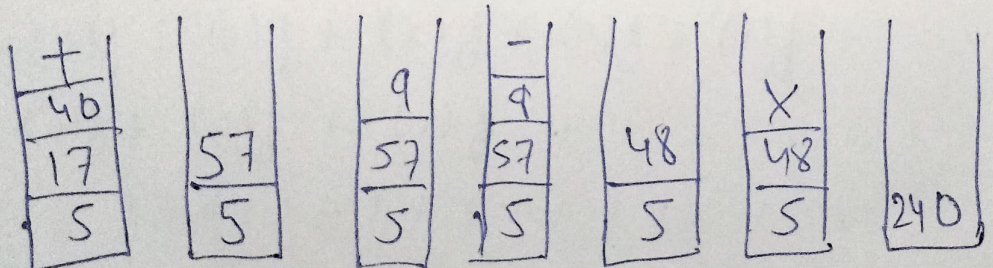
(c) Recursive function for binary search

```
int binaryRecursive(int arr[], int low, int high, int ele){
  if (low <= high){
    int mid = low + (high - low) / 2;

    if (arr[mid] == ele) {
      return mid;
    }
    if (arr[mid] < ele) {
      return binaryRecursive(arr, mid+1, high, ele);
    } else {
      return binaryRecursive(arr, low, mid-1, ele);
    }
  }
  return -1;
}
```

(d) $BAC + BCX + A - X$

$B = 5, A = 9, C = 8$





3.

(i) Counting no. of right childrens

```
int countRight ( Node * node){
    if ( node == NULL){
        return 0;
    }
int rightcount = countRight (node -> right);
return . rightcount +1 + countRight (node -> left);
}
```

Calculating height of the tree

```
int height ( Node * node ){
    if (node == NULL){
        return 0;
    }
int lefth = height ( node -> left );
int rightth = height ( node -> right );
return . 1 + max( lefth , rightth );
}
```

4. (b) $((A+B)*(C-D))^{(F+G)}$

Postfix : AB + CD - * FG + ^

Prefix : ^ * + AB - CD + FG

5 (a)

```
class Queue{
    int front;
    int rear;
    int * arr;
    int maxsize;
public:
    Queue (int size){
        max size = size;
        arr = new int [max size];
        front = -1;
        rear = -1;
    }

bool isEmpty (){
    return front == -1;
}

~~bool isfull~~
bool isfull(){
    return (rear+1) % maxsize == front;
}
```

```
void enqueue (int val){
    if (isfull()){
        cout << "Queue is full";
    }
    else if (isEmpty()){
        front = rear = 0;
        arr[rear] = val;
    }
    else{ rear = (rear+1)% max size;
        arr[rear] = val;
    }
}
void dequeue (){
    if (isEmpty()){
        cout << "Empty Queue";
    }
    else if (front == rear){
        front = rear = -1;
    }
    else{
        front = (front+1) % max size;
    }
}
```

(b)

```
void convert (int num){
    stack binaryStack(32);
    while (num > 0){
        binarystack.push(num%2);
        num %= 2;
    }
```

```
cout << " Binary representation: ";
while (!binaryStack.isEmpty()){
    cout << binaryStack.pop();
}
cout << endl;
}
```

6.

(i)

```
Node* reverse(Node* head){
    Node* curr = head;
    Node* prev = NULL;
    while(curr != NULL){
        Node* nextptr = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextptr;
    }
    node* new_head = prev;
    return new_head;
}
```

(ii)

```
void insert(int val){
    Node * p = new node(val);
    node * temp = head;
    while(temp != NULL){
        temp = temp->next;
    }
    temp->next = p;
    p->prev = temp;
}
```

(iii)

```
Node * merge(Node* l1, Node* l2){
    Node dummy(0);
    Node * current = &dummy;
while(l1 != NULL && l2 != NULL){
    if(l1->elem < l2->elem){
        current->next = l1;
        l1 = l1->next;
    }
    else{ current->next = l2
        l2 = l2->next;
    }
    current = current->next;
}
```

```
if (l1 != NULL) {
    current -> next = l1;
}
else {
    current -> next = l2;
}
return dummy -> next;
}
```

Ques Paper - 3

1 (a)

```
void enqueue (int val){
  Node * p = new node (val);
    if (isempty()){
       front = rear = p;
       rear -> next = front;
    }
    else {
       rear -> next = p;
       rear = p;
       rear -> next = front;
    }
}
void dequeue (){
   if ( front == rear ){
       delete front;
       front = rear = NULL;
   }
   else {
       Node * temp = front;
       front = front -> next;
       rear -> next = front;
       delete temp;
   }
}
```

(b)

```
void recursive(int n, int i = 1);
    if (i > n){
        return;
    }
    if (i % 2 == 0){
        cout << i*i*i;
    }
    recursive(n, i+1);
}
```

(C)

Expression	Stack
4	4
10	4 10
5	4 10 5
+	4 (10 + 5)
*	(4 * (10 + 5))
15	(4*(10 + 5)) 15
3	(4 * (10 + 5) 15 3
/	(4 * (10 + 5)) (15/3)
-	(4*(10 + 5)) - (15/3)

= 55

(d) Pass 1

8, 56, 12, 34, 92, 9, 44, 23

Pass 2

8, 9, 12, 34, 92, 56, 44, 23

Pass 3

8, 9, 12, 34, 92, 56, 44, 23

Pass 4

8, 9, 12, 23, 92, 56, 44, 34

Pass 5

8, 9, 12, 23, 34, 56, 44, 92

Pass 6

8, 9, 12, 23, 34, 44, 56, 92

Pass 7

8, 9, 12, 23, 34, 44, 56, 92

(b) If the keys to be searched are all present in the initial few position of a sorted array, a suitable search technique would be BINARY SEARCH.

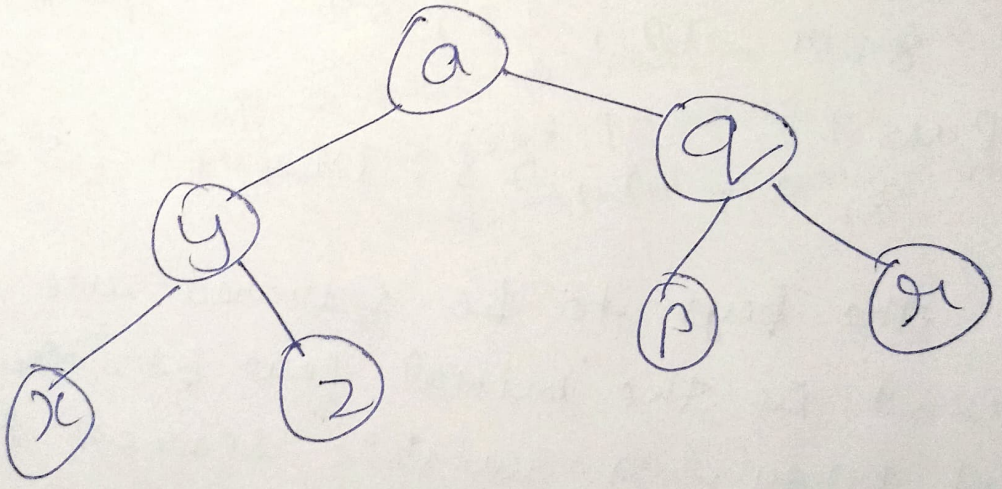
Efficiency: Binary search is efficient. It narrows down the search space by half in each step, making it particularly effective for large datasets.

Key in initial pos.: Since keys to be searched are all present in the initial pos., binary search can quickly locate them by repeatedly dividing the search interval in half.

Minimal comparisons: Binary search performs a minimal number of comparisons, and the time complexity is logarithmic.

(g) Inorder: x y z a p q r

Preorder: a y x z q p r



2. (a)
```
void dispReverse( Node* head){
    if (head == NULL){
        return;
    }
    dispReverse(head->next);
    cout << head->data << " ";
}
```

(b)
```
void delete(int pos){
    Node* temp = head;
    int count = 1;
    while (temp != NULL && count != pos){
        temp = temp->next;
        count++;
    }
    temp->prev->next = temp->next
    if (temp->next != NULL){
        temp->next->prev = temp->prev;
    }
    delete temp;
}
```

3. (a)
```
void reverseStack(){
    stack tempStack1;
    stack tempStack2;
    while (!isEmpty(1)){
        tempStack1.push(pop(1));
    }
    while (!tempStack1.isEmpty(1)){
        tempStack2.push(tempStack1.pop(1));
    }
```

```
while (!tempStack2.isEmpty()){
    push(tempStack2.pop());
    }
}
```

(b) For the given set [1, 2, 3, 4, 5, 6];

→ Both bubble sort and insertion sort will have optimal performance because array is already sorted.

Bubble sort

Pass 1 (1, 2, 3, 4, 5, 6) - No swaps needed already sorted.

Insertion sort

Pass 1 (1, 2, 3, 4, 5, 6) - No swaps needed, already sorted

4. [a] enqueue (14)

front = ~~10~~ 64

rear = 14

[1]	[2]	[3]	[4]	[5]	[6]
64	4	5	6	7	14
					1
Front					Rear

enqueue (56)

[1]	[2]	[3]	[4]	[5]	[6]	[7]
64	4	5	6	7	14	56
1						1
Front						Rear

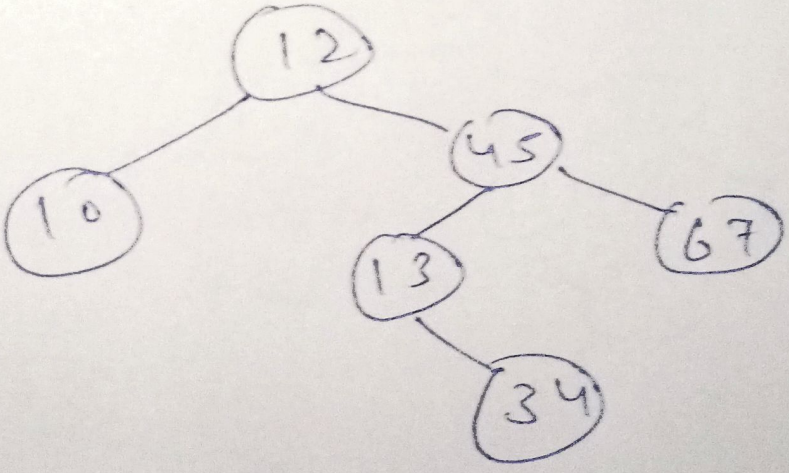
dequeue()

[2]	[3]	[4]	[5]	[6]	[7]
4	5	6	7	14	56
1					1
front					Rear

dequeue()

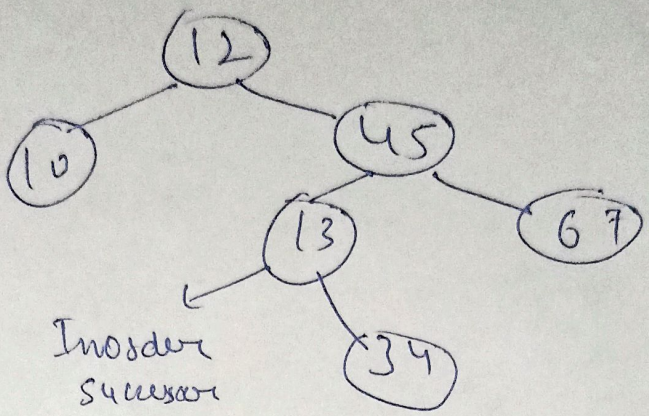
[3]	[4]	[5]	[6]	[7]
5	6	7	14	56
1				1
Front				Rear

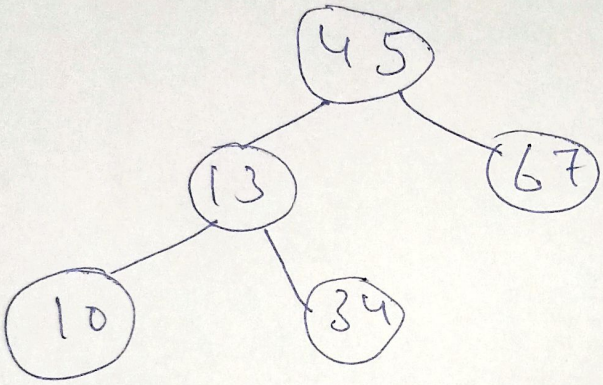
6. (a) 12, 45, 13, 67, 10, 34



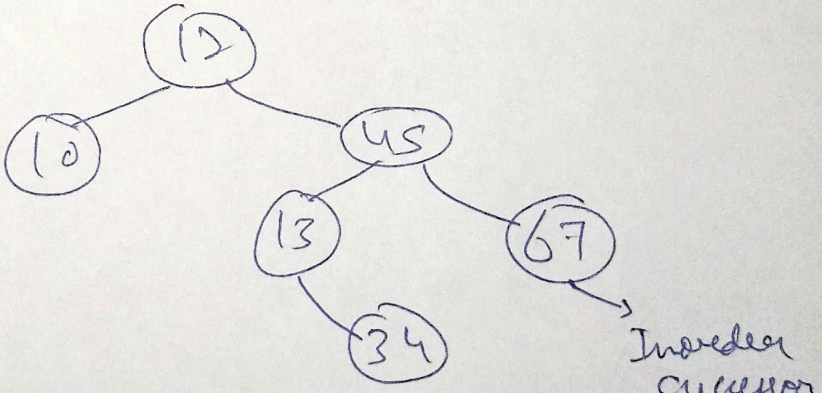
Inorder : 10 12 13 34 45 67

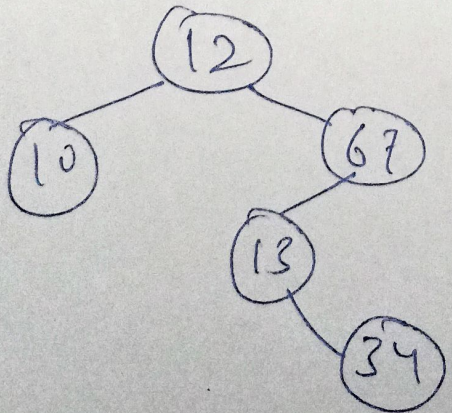
Deleting 12 by merging





Deleting 45 by copying





(b)

```
int countLeaves ( Node * root ){
    if ( root == NULL){
        return 0;
    }
    if ( root -> left == NULL && root -> right == NULL){
        return 1;
    }
    int leavesLeft = countLeaves ( root -> left );
    int leavesRight = countLeaves ( root -> right );
    return leavesLeft + leavesRight;
}
```