



AUTUMN END SEMESTER EXAMINATION-2017
Data Structure & Algorithms Solution and Evaluation Scheme
CS-2001/CS-301

Full Marks: 60

Time: 3 Hours

Answer any six questions including question No.1 which is compulsory.

The figures in the margin indicate full marks.

*Candidates are required to give their answers in their own words as far as practicable
and all parts of a question should be answered at one place only*

Q1 Answer all

(2 x 10)

a) What is the time complexity of the following function func()?

```
void func( ) {  
    int i, j;  
    for (i=1; i<=n; i++)  
        for(j=1; j<=log(i); j++)  
            printf("KIIT University");  
}
```

Evaluation Scheme: 2 marks for correct answer. Depending upon the steps, partial marks may be awarded judiciously.

Solution:

The outer loop will run for n times.

The inner loop will run as : $\log(1) + \log(2) + \log(3) + \dots + \log(n) = \log(n!)$

So the time complexity will be **O (n log n)**

b) Write a pseudo code to search an element in a single linked list.

Evaluation Scheme: 2 marks for correct answer. Depending upon the steps, partial marks may be awarded judiciously. There may be different possible approaches.

Solution:

/* This function checks whether an element item is present in linked list and returns true if the element is present otherwise returns false. */

bool search(struct node* start, int item)

```
{  
    struct node* ptr = start;  
    while (ptr != NULL)  
    {  
        if (ptr->data == item)  
            return true;  
        ptr = ptr->next;  
    }  
    return false;  
}
```

c) Distinguish between tridiagonal & triangular sparse matrix with example.

Evaluation Scheme: One mark for the difference and one mark for the example. Depending upon the steps, partial marks may be awarded judiciously.

Solution:

Tridiagonal Sparse Matrix:

In this matrix, all the elements are zero except the elements present in the main diagonal, the diagonal present above the main diagonal and the element present below the main diagonal. Below example is illustrating 6X6 matrix.

12	50	0	0	0	0
90	32	33	0	0	0
0	06	45	55	0	0
0	0	41	11	6	0
0	0	0	8	70	52
0	0	0	0	25	88

Triangular Sparse Matrix:

A triangular sparse matrix is a special kind of square matrix. A square matrix is called lower triangular if all the entries above the main diagonal are zero. Similarly, a square matrix is called upper triangular if all the entries below the main diagonal are zero. A triangular matrix is one that is either lower triangular or upper triangular.

Example:

0	0	0	0	0	0
90	0	0	0	0	0
01	06	0	0	0	0
37	21	41	0	0	0
44	32	23	08	0	0
90	50	55	20	25	0

(I: Lower Triangular Sparse Matrix)

0	10	33	43	78	99
0	0	55	79	57	11
0	0	0	23	09	23
0	0	0	0	13	45
0	0	0	0	0	02
0	0	0	0	0	0

(II: Upper Triangular Sparse Matrix)

d) Write a function to test whether a loop exists in a single linked list.

Evaluation Scheme: 2 marks for correct answer. Depending upon the steps, partial marks may be awarded judiciously. There may be different approaches possible.

Solution:

/* below function detects the loop that exists in a single linked list. The function takes the address of the first node of the single linked list as the parameter and returns 1 if the loop exists else returning 0. */

```
int detectloop(struct node *start)
{
    struct node *p1 = start, *p2 = start;
    while (p1 && p2 && p2->next )
    {
        p1 = p1->next;
        p2 = p2->next->next;
        if (p1 == p2)
        {
            printf("Found Loop");
            return 1;
        }
    }
    return 0;
}
```

e) What is stack ADT error & implementation error? Explain with suitable example.

Evaluation Scheme: 0.5 mark for ADT error, 0.5 marks for implementation error and 1 mark for example. Depending upon the steps, partial marks may be awarded judiciously.

Solution:

ADT error :- In stack, if the insertion and deletion operations are not performed based on LIFO (Last In First Out) principle, then it qualifies for the ADT error.

Stack Implementation error: - It signifies the overflow and underflow error condition.

Example - Let's assume that stack is implemented using array

```
#define n 10
int stack[n];
int top = -1;

void push (int item)
{
    if (top >= n)
    {
        printf("Overflow"); // Stack Implementation error for Overflow
        return;
    }
    stack[++top] = item; //proper line
    /* if the push operation is defined by anything other than the proper line like
       stack [0] = item; OR stack [n-1] = item; then leads to ADT error.
    */
}
```

```

int pop()
{
    if (top == -1)
    {
        printf("Underflow"); // Stack Implementation error for Underflow
        exit(0);
    }
    return stack[--top]; // proper line
    /*
        if the pop operation is anything other than the proper line like
        return stack[0]; OR return stack[n-1]; is an ADT error.
    */
}

```

f) Let ptr is a pointer to a node in a circular single linked list. Write pseudo code to delete the node to which ptr is pointing.

Evaluation Scheme: 2 marks for correct answer. Depending upon the steps, partial marks may be awarded judiciously.

Solution:

/* This function deletes a node with address a ptr. start is the starting address of the list and is a global variable. Assume that the list has at least one node.*/

void delete (struct node *ptr)

```

{
    struct node* p1;
    for(p1=start;p1->next!=ptr;p1=p1->next);
    if(p1==ptr)
    {
        free(ptr);
        start=NULL;
    }
    else if(ptr==start)
    {
        p1->next=ptr->next;
        start=p1->next;
        free(ptr);
    }
    else
    {
        p1->next=ptr->next;
        free(ptr);
    }
}

```

g) Write the condition for overflow and underflow situation in a circular queue.

Evaluation Scheme: One mark for writing the overflow and one mark for underflow condition.

No step marking.

Solution:

For a Circular Queue implemented using an array of size "max" and rear & front represent the point of insertion and point of deletion respectively

Overflow Condition:

Front = (rear+1)%max

Underflow Condition:

rear = front = -1

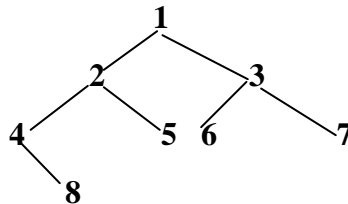
h) Find the preorder traversal of a binary tree which has been constructed from the given postorder and inorder traversals: in[] = {4, 8, 2, 5, 1, 6, 3, 7}, post[] = {8, 4, 5, 2, 6, 7, 3, 1}.

Evaluation Scheme: As per the question drawing the binary tree from the given inorder and postorder traversal is optional. So, 2 marks should be awarded for the correct preorder traversal sequence. **No step marking.**

Solution:

The preorder traversal of the tree is: pre[] = {1, 2, 4, 8, 5, 3, 6, 7}

Binary tree is as follows:



i) Write a C code to find number of nodes in a binary tree.

Evaluation Scheme: There are different approaches possible. Number of nodes in a binary tree can be computed using any traversal technique. 2 marks should be awarded for any correct recursive or non-recursive code.

Solution:

/* below function finds the number of nodes in a binary tree. The function takes the address of the root node of the binary tree as the parameter. The *lptr* and *rptr* represents the address of the left child node and address of the right child node respectively. */

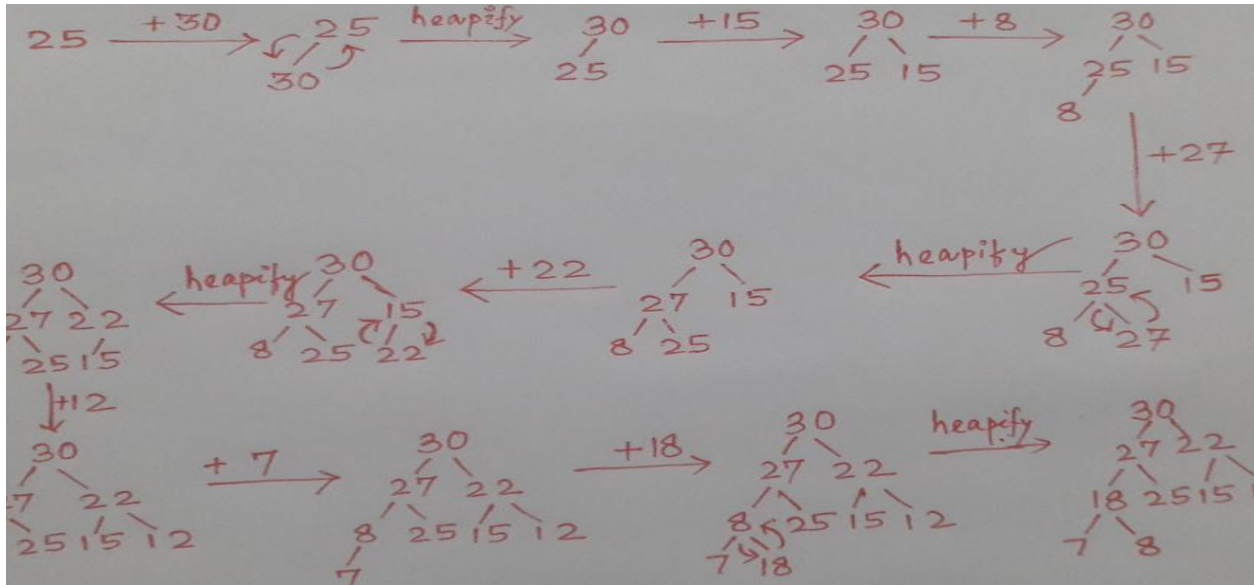
```
int count(struct node *n)
{
    int c = 1;
    if (n == NULL)
        return 0;
    else
    {
        c += count(n->lptr);
        c += count(n->rptr);
        return c;
    }
}
```

j) Construct a max heap with the following data elements.

25, 30, 15, 8, 27, 22, 12, 7, 18

Evaluation Scheme: 2 marks for correct answer. Depending upon the steps, partial marks may be awarded judiciously.

Solution:



2.a Write an algorithm/function to find the minimum difference between any two elements in a given array. [4 marks]

Evaluation Scheme: 4 marks for correct answer. Step marking should be given judiciously.

Solution:

```
void min_diff(int a[], int n)
{
    int i,j,min,temp,diff;
    for(i=0;i<n;i++)
    {
        min=i;
        for(j=i+1;j<n;j++)
        {
            if(a[j]<a[min])
                min=j;
            temp=a[i];
            a[i]=a[min];
            a[min]=temp;
            diff=a[i];
        }
    }
    for(i=0;i<n;i++)
    {
        if(a[i+1]-a[i]<diff)
            diff=a[i+1]-a[i];
    }
    printf("min diff. is %d",diff);
}
```

2.b) Write an algorithm /Pseudo code/C code snippet to remove duplicate elements from unsorted single linked list. [4 marks]

Evaluation Scheme: 4 marks for correct answer. Step marking should be given judiciously.

Solution:

```
/* Function to remove duplicates from an unsorted linked list */
void removeDuplicates(struct node *start)
{
    struct node *ptr1, *ptr2, *dup;
    ptr1 = start; /* Pick elements one by one */
    while (ptr1 != NULL && ptr1->next != NULL)
    {
        ptr2 = ptr1;
        /* Compare the picked element with rest
        of the elements */
        while (ptr2->next != NULL)
        {
            /* If duplicate then delete it */
            if (ptr1->data == ptr2->next->data)
            {
                /* sequence of steps is important here */
                dup = ptr2->next;
                ptr2->next = ptr2->next->next;
                delete(dup);
            }
            else /* This is tricky */
                ptr2 = ptr2->next;
        }
        ptr1 = ptr1->next;
    }
}
```

3.a) Write a Pseudo code to reverse a single linked list while traversing the list once. [4 marks]

Evaluation Scheme: 4 marks for correct answer. Step marking should be given judiciously. There are different approaches possible.

Solution:

```
struct node * invert(struct node * head)
{
    struct node *p,*q,*r;
    if (head == NULL) return;
    p=head;
    q=p->next;
    r=q->next;
    p->next=NULL;
```

```

while(q)
{
    q->next=p;
    p=q;
    q=r;
    if(r!=NULL)
        r=r->next;
}
head=p;
return head;
}

```

3.b) Write an algorithm /Pseudo code to merge two sorted linked list to produce a sorted linked list. [4 marks]

Evaluation Scheme: 4 marks for correct answer. Step marking should be given judiciously. There are different possible approaches.

Solution:

```

struct node
{
    int data;
    struct node *next;
};

/* h1 is pointing to the start node of 1st linked list and h2 is pointing to the start node of 2nd
linked list */
struct node* merge(struct node *h1, struct node *h2)
{
    if (!h1)
        return h2;
    if (!h2)
        return h1;
    if (h1->data < h2->data)
    {
        h1->next = merge(h1->next, h2);
        return h1;
    }
    else
    {
        h2->next = merge(h1, h2->next);
        return h2;
    }
}

```


Q4)(a) Write a C- Pseudo code to implement the push and pop operations of two different stacks at two ends of a given array. [4 marks]

Evaluation Scheme: 4 marks for correct answer. Step marking should be given judiciously.

Solution:

```
#define SIZE 10
```

```
int ar[SIZE];
int top1 = -1;
int top2 = SIZE;
```

/* Push functions for two stacks. It can also be combined into a single push function with an extra input to choose the appropriate stack. */

<pre>void push_stack1 (int data) { if (top1 < top2 - 1) { ar[++top1] = data; } else { printf ("Stack Overflow"); } }</pre>	<pre>void push_stack2 (int data) { if (top1 < top2 - 1) { ar[--top2] = data; } else { printf ("Stack Overflow"); } }</pre>
---	---

/* Pop functions for two stacks. It can also be combined into a single pop function with an extra input to choose the appropriate stack. */

<pre>void pop_stack1 () { if (top1 >= 0) { int popped_value = ar[top1--]; printf ("%d is popped \n", popped_value); } else { printf ("Stack Under flow"); } }</pre>	<pre>void pop_stack2 () { if (top2 < SIZE) { int popped_value = ar[top2++]; printf ("%d is popped \n", popped_value); } else { printf ("Stack Under flow"); } }</pre>
--	--

4(b) Write C-Pseudo code for evaluation of postfix expression using stack.

[4 mark]

Evaluation Scheme: 4 marks for correct answer. Step marking should be given judiciously.

Solution:

/* This program assume that there are only four arithmetic operators (*, /, +, -) in an expression
operand is single digit only */

```
# define MAXSTACK 100      /* for max size of stack */  
int stack[MAXSTACK];  
int top = -1 ;
```

/* push operation */

void push(int item)

```
{  
    if(top >= MAXSTACK -1)  
    {  
        printf("stack over flow");  
        return;  
    }  
    else  
    {  
        top = top + 1 ;  
        stack[top]= item;  
    }  
}
```

/* pop operation */

int pop()

```
{  
    int item;  
    if(top <0)  
    {  
        printf("stack under flow");  
    }  
    else  
    {  
        item = stack[top];  
        top = top - 1;  
        return item;  
    }  
}
```

```

/* define function that is used to input postfix expression and to evaluate it */
void EvalPostfix(char postfix[])
{
    int i ;
    char ch;
    int val;
    int A, B ;
    /* evaluate postfix expression */
    for (i = 0 ; postfix[i] != ')'; i++)
    {
        ch = postfix[i];
        if (isdigit(ch))
        {
            push(ch - '0');
/* ch-48 can also be used to push the numeric value of the operand not the ASCII value*/
        }
        else if (ch == '+' || ch == '-' || ch == '*' || ch == '/')
        {
            /* For an operator
            * pop top element A and next-to-top element B
            * from stack and compute B operator A
            */
            A = pop();
            B = pop();
            switch (ch) /* ch is an operator */
            {
                case '*':
                    val = B * A;
                    break;

                case '/':
                    val = B / A;
                    break;

                case '+':
                    val = B + A;
                    break;

                case '-':
                    val = B - A;
                    break;
            }
            /* push the value obtained above onto the stack */
            push(val);
        }
    }
    printf( " \n Result of expression evaluation : %d \n", pop()) ;
}

```

Q5)(a) Why AVL trees are used? Discuss the all steps to ensure that the given tree remains AVL after every insertion. Illustrate the steps involved while building an AVL tree with values:15,20,24,10,13,7,30,36,25. [6 marks]

Evaluation Scheme:

Usage of AVL Tree **1 mark**

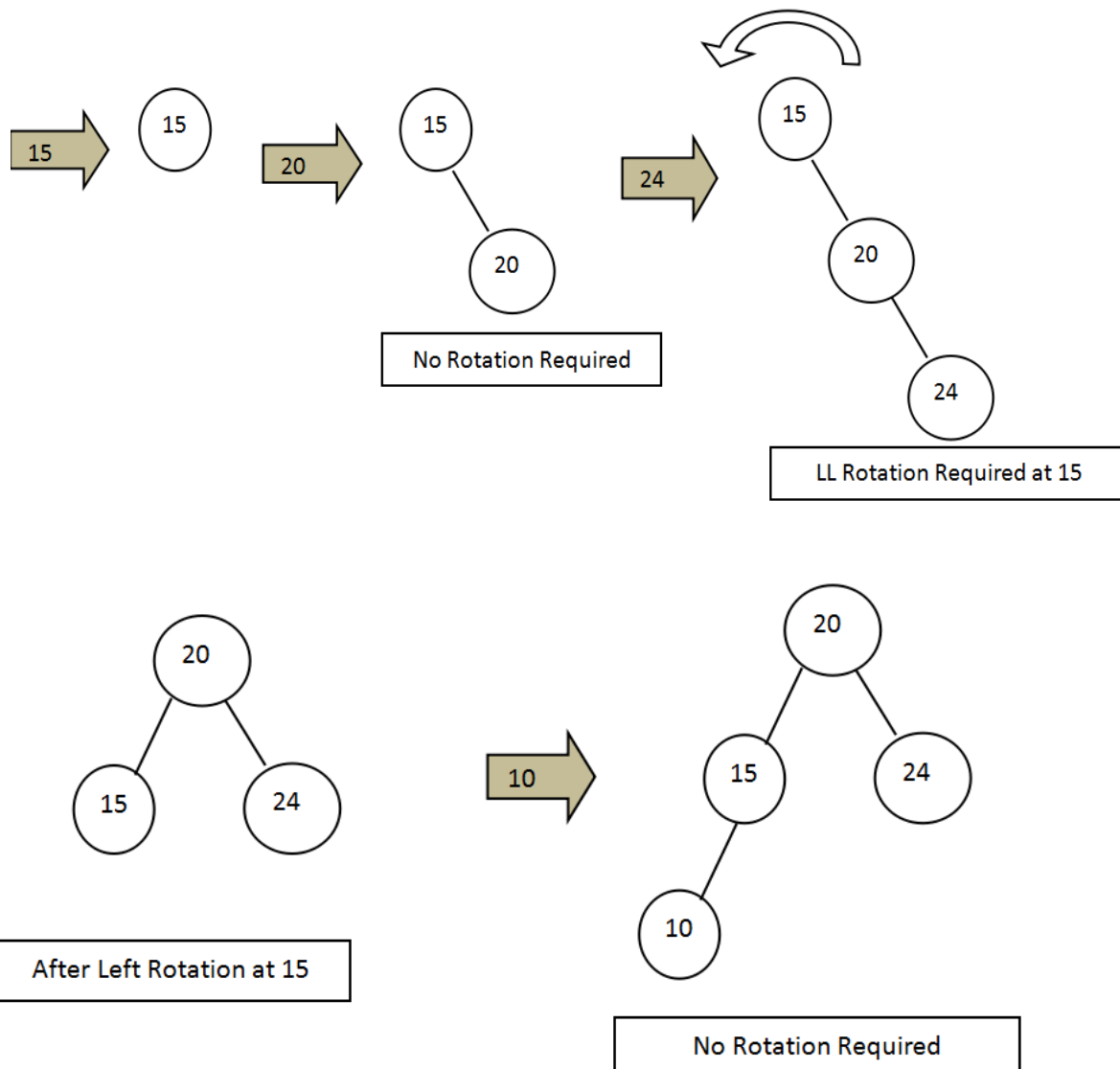
Explanations of Rotations **1 mark**

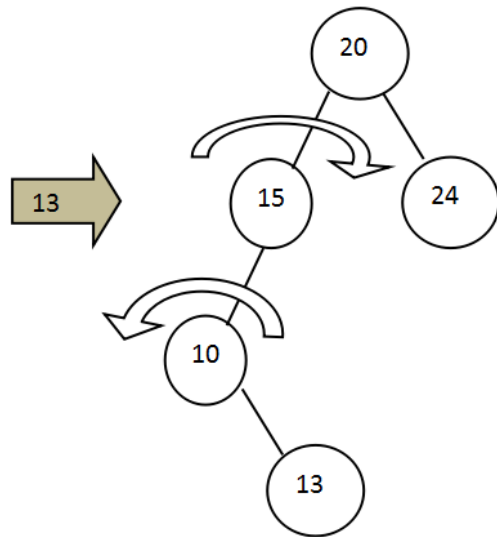
Construction of AVL tree with the specified values by performing suitable rotations **4 marks**

Solution:

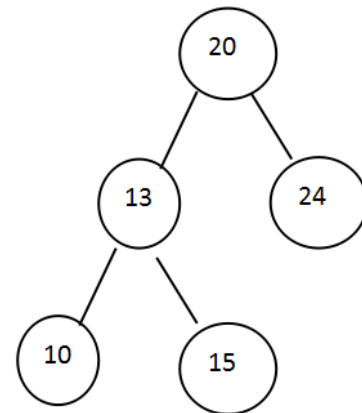
Usage of AVL Tree - AVL trees are self balancing Binary Search Trees. They are useful in the situation where insertion/ deletions are rarely done but lookup/searching operations are done frequently.

Explanations of Rotations – Look for explanation of left, right, left-right and right-left rotations
Steps illustration for AVL tree construction -



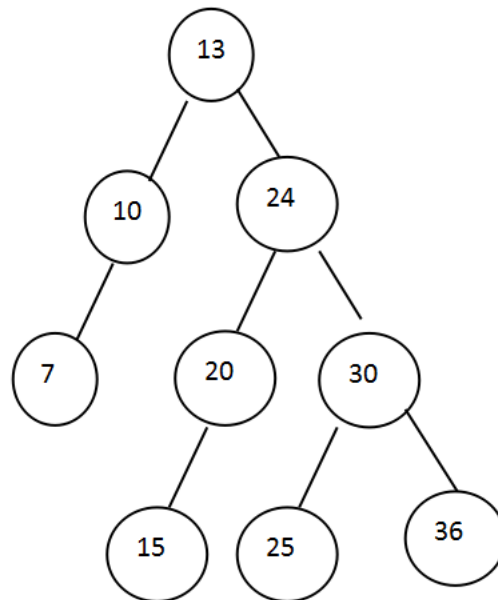


Double Rotation Required
Left Rotation at 10 then Right
Rotation at 15



After Rotations

After inserting 7, 30, 36 and 25 and performing suitable rotations, the Final AVL Tree is:



5(b) Explain recursive in order tree traversal with suitable example.

[2 marks]

Evaluation Scheme:

Recursive procedure 1 mark

Explanation with example carries 1 mark

Solution:

```
void inorder(struct node * r)
{
    if(r!=NULL)
    {
        inorder(r->lchild);
        printf("%d", r->info);
        inorder(r->rchild);
    }
}
```

6. Write the pseudo code for Quick sort including partition() function. Also illustrate the step-by-step process of partition() function with a given array: {10, 80, 30, 90, 40, 50, 70}, considering last element as the pivot element.

[8 marks]

Evaluation Scheme: Pseudo code for partition() function carries 4 marks and step-wise illustration of the example carries 4 marks. Step marking should be given judiciously.

Solution:

Quicksort is based on the divide-and-conquer paradigm. Here is the three-step divide-and-conquer process for sorting a typical sub array $A[p \dots r]$.

Divide: The array $A[p \dots r]$ is partitioned (rearranged) into two nonempty sub arrays $A[p \dots q]$ and $A[q + 1 \dots r]$ such that each element of $A[p \dots q]$ is less than or equal to each element of $A[q + 1 \dots r]$. The index q is computed as part of this partitioning procedure.

Conquer: The two sub arrays $A[p \dots q]$ and $A[q + 1 \dots r]$ are sorted by recursive calls to quick sort.

Combine: Since the sub arrays are sorted in place, no work is needed to combine them. Then entire array $A[p \dots r]$ is now sorted.

The following procedure implements quick sort:

```

/* low --> Starting index, high --> Ending index */
QUICK SORT (arr[], low, high)
{
    if (low < high)
    {
        pi = PARTITION(arr, low, high); //pi is partitioning index
        QUICKSORT(arr, low, pi - 1);
        QUICKSORT(arr, pi + 1, high);
    }
}

```

To sort an entire array A, the initial call is QUICKSORT (A, 1, length[A]).

Partitioning the array →

/* This function takes last element as pivot, places the pivot element at its correct position in sorted array, and places all smaller (smaller than pivot) to left of pivot and all greater elements to right of pivot */

```

PARTITION (arr[], low, high)
{
    pivot = arr[high];
    i = (low - 1);
    for (j = low; j <= high- 1; j++)
    {
        if (arr[j] <= pivot)
        {
            i++;
            swap arr[i] and arr[j]
        }
    }
    swap arr[i + 1] and arr[high])
    return (i + 1)
}

```

The following procedure illustrate the step-by-step process of partition() function with a given array: { 10, 80, 30, 90, 40, 50, 70}, considering last element as the pivot element.

arr[] = { 10, 80, 30, 90, 40, 50, 70}
 Indexes: 0 1 2 3 4 5 6

low = 0, high = 6, pivot = arr[h] = 70
 Initialize index of smaller element, i = -1

Traverse elements from $j = \text{low}$ to $\text{high}-1$

$j = 0$: Since $\text{arr}[j] \leq \text{pivot}$, do $i++$ and $\text{swap}(\text{arr}[i], \text{arr}[j])$

$i = 0$

$\text{arr}[] = \{10, 80, 30, 90, 40, 50, 70\}$ // No change as i and j are same

$j = 1$: Since $\text{arr}[j] > \text{pivot}$, do nothing

// No change in i and $\text{arr}[]$

$j = 2$: Since $\text{arr}[j] \leq \text{pivot}$, do $i++$ and $\text{swap}(\text{arr}[i], \text{arr}[j])$

$i = 1$

$\text{arr}[] = \{10, 30, 80, 90, 40, 50, 70\}$ // We swap 80 and 30

$j = 3$: Since $\text{arr}[j] > \text{pivot}$, do nothing

// No change in i and $\text{arr}[]$

$j = 4$: Since $\text{arr}[j] \leq \text{pivot}$, do $i++$ and $\text{swap}(\text{arr}[i], \text{arr}[j])$

$i = 2$

$\text{arr}[] = \{10, 30, 40, 90, 80, 50, 70\}$ // 80 and 40 Swapped

$j = 5$: Since $\text{arr}[j] \leq \text{pivot}$, do $i++$ and swap $\text{arr}[i]$ with $\text{arr}[j]$

$i = 3$

$\text{arr}[] = \{10, 30, 40, 50, 80, 90, 70\}$ // 90 and 50 Swapped

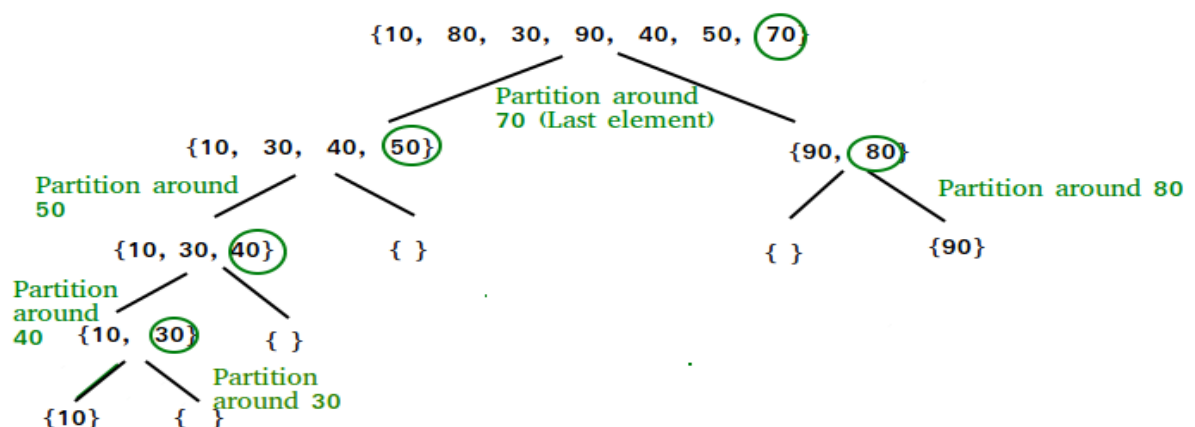
We come out of loop because j is now equal to $\text{high}-1$. Finally we place pivot at correct position by swapping

$\text{arr}[i+1]$ and $\text{arr}[\text{high}]$ (or pivot)

$\text{arr}[] = \{10, 30, 40, 50, 70, 90, 80\}$ // 80 and 70 Swapped

Now 70 is at its correct place. All elements smaller than 70 are before it and all elements greater than 70 are after it.

Pictorial depiction:-





Q7(a) Differentiate between BFS and DFS with suitable example .**[4 marks]****Evaluation Scheme:**

At least 4 differences and each difference to carry 0.5 marks, so total 2 marks

Illustration with examples 2 marks

Solution:

BFS	DFS
BFS Stands for “Breadth First Search”.	DFS stands for “Depth First Search”.
BFS starts traversal from the root node and then explore the search in the level by level manner i.e. as close as possible from the root node.	DFS starts the traversal from the root node and explore the search as far as possible from the root node i.e. depth wise.
Breadth First Search can be done with the help of queue i.e. FIFO implementation.	Depth First Search can be done with the help of Stack i.e. LIFO implementations.
This algorithm works in single stage. The visited vertices are removed from the queue and then displayed at once.	This algorithm works in two stages – in the first stage the visited vertices are pushed onto the stack and later on when there is no vertex further to visit those are popped-off.
BFS is slower than DFS.	DFS is faster than BFS.
BFS requires more memory compare to DFS	DFS require less memory compare to BFS
Applications of BFS > To find Shortest path > Single Source & All pairs shortest paths > In Spanning tree > In Connectivity	Applications of DFS > Useful in Cycle detection > In Connectivity testing > Finding a path between V and W in the graph. > Useful in finding spanning trees & forest.
BFS is useful in finding shortest path.BFS can be used to find the shortest distance between some starting node and the remaining nodes of the graph.	DFS in not so useful in finding shortest path. It is used to perform a traversal of a general graph and the idea of DFS is to make a path as long as possible, and then go back (backtrack) to add branches also as long as possible.
<p>Example:</p>  <p>BFS Traversal A, B, C, D, E, F</p>	<p>Example:</p>  <p>DFS Traversal A, B, D, C, E, F</p>

Q7(b) What do you mean by collision in hashing ? How to handle collisions? Draw the neat diagram to illustrate separate chaining method to handle collisions using hash function as “key mod 7” and sequence of keys as 50,700,76,85,92,73,101. [4marks]

Evaluation Scheme:

- (i) Definition of collision in hashing -1 mark
- (ii) Diagram along with illustration-3 marks

Solution:

Hashing is a concept that is used to search an item in $O(1)$ time. It is a completely different approach from the comparison-based methods (binary search, linear search). Rather than navigating through a list data structure comparing the search key with the elements, hashing tries to reference an element in a table directly based on its search key k .

A situation when the resultant hashes for two or more data elements maps to the same location in the hash table, is called a hash collision.

There are 2 ways to handle collision:

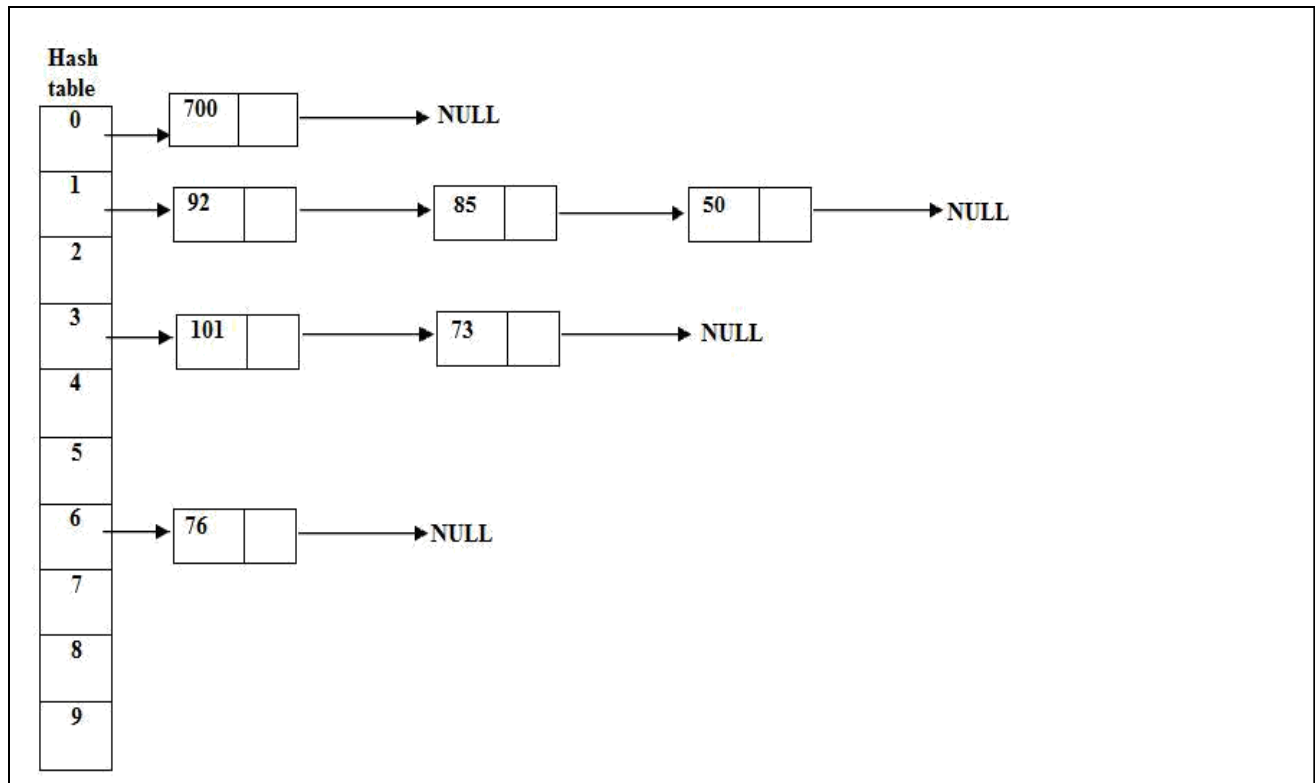
Closed Hashing—Array based implementation. Also called as Open Addressing

Open Hashing—Array of linked list implementation. Also called as Separate Chaining

The difference has to do with whether collisions are stored outside of the hash table (open hashing) or whether collisions result in storing one of the records at another slot in the hash table (closed hashing)

Open Addressing includes:

- i) Linear Probing (Linear Search)
- ii) Quadratic Probing (Non linear Search)
- iii) Double Hashing (Uses two hash function)



**8(a) Define B-tree. Create a B-tree of order 3 with the following elements.
8,9,10,11,12,7,6,5,20,26,22.**

[4 marks]

Evaluation Scheme:

Definition of B-tree-1 mark

Construction of B tree -3 marks

Solution:

B-tree:

M-way binary search trees have the advantages of having a node storing multiple values. However it is essential that the height of the tree is to be kept as low as possible and therefore there arises the need of maintaining the balanced m-way search trees. Such balanced m-way search tree is B-tree.

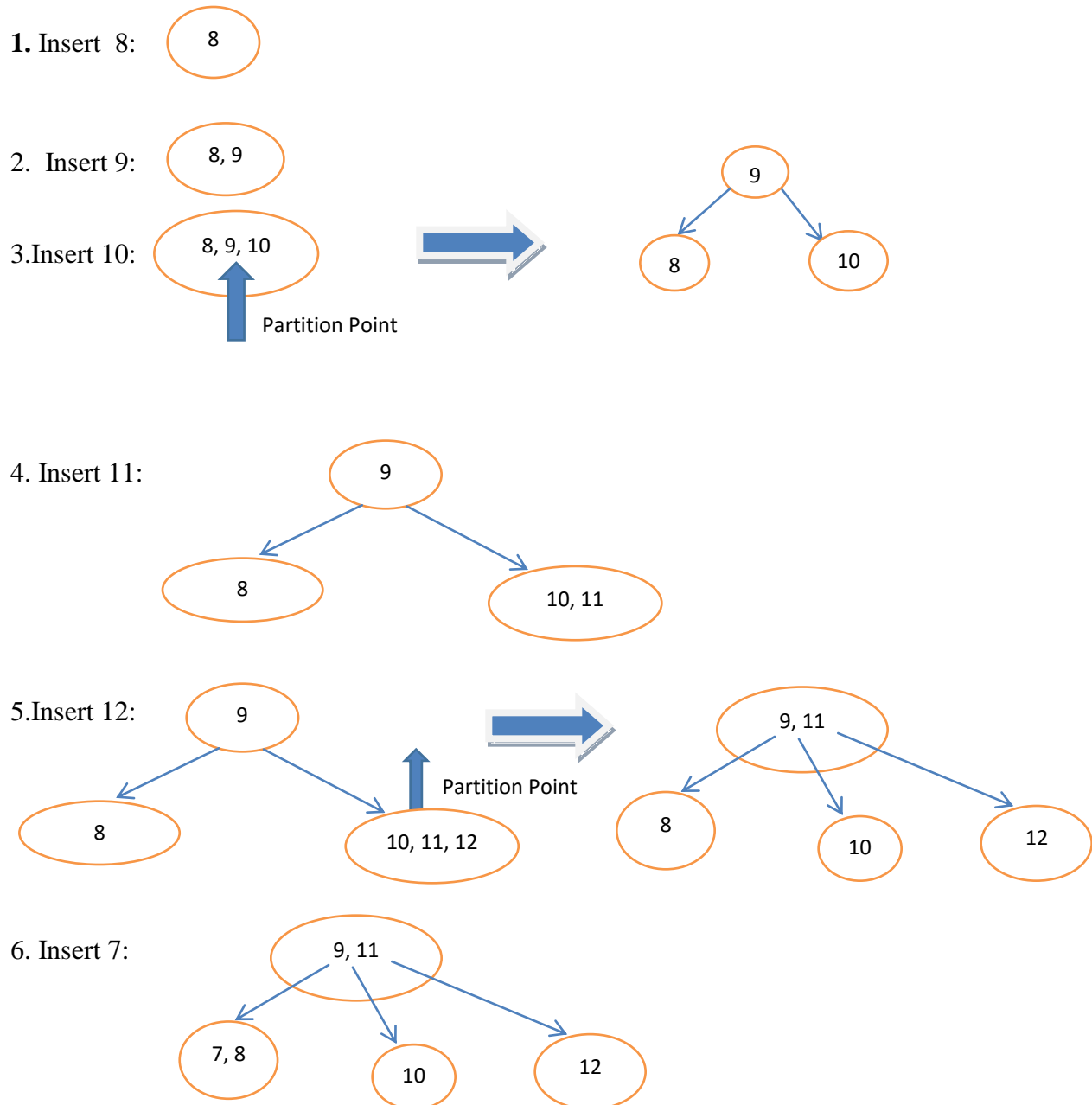
Properties of B-Tree of order m:

- 1) All leaves are at same level.
- 2) Every node except root must contain at least m-1 keys. Root may contain minimum 1 key.
- 3) All nodes (including root) may contain at most $2m - 1$ keys.
- 4) Number of children of a node is equal to the number of keys in it plus 1.
- 5) All keys of a node are sorted in increasing order. The child between two keys k_1 and k_2 contains all keys in range from k_1 and k_2 .

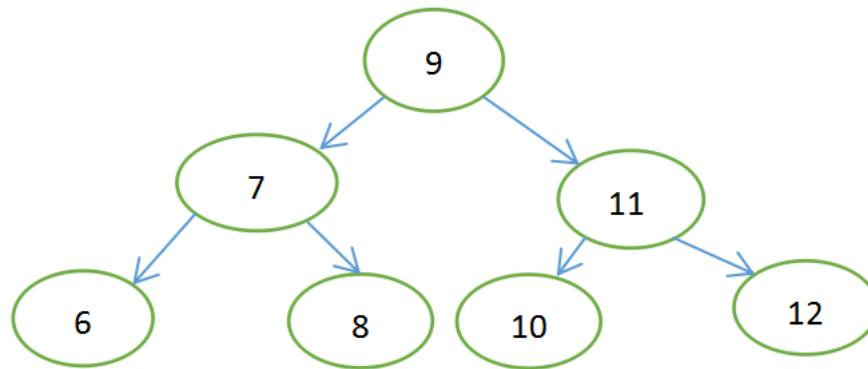
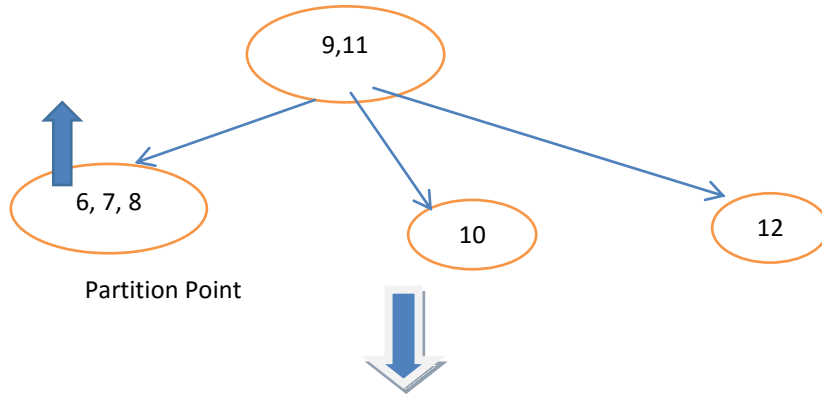
6) B-Tree grows and shrinks from root which is unlike Binary Search Tree. Binary Search Trees grow downward and also shrink from downward.

7) Like other balanced Binary Search Trees, time complexity to search, insert and delete is $O(\log n)$.

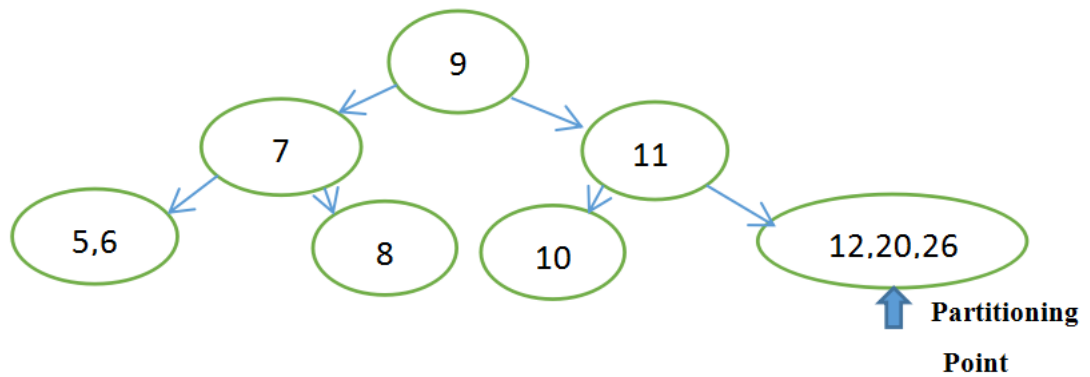
Following is an example B-Tree of minimum degree 3. Note that in practical B-Trees, the value of minimum degree is much more than 3.



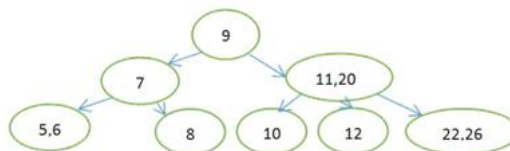
7. Insert 6:



8. Insert 5, 20 and 26



9. Insert 22



8 (b) What is threaded binary tree? What are its advantages? Explain with example of a in order threaded binary tree? [4 mark]

Evaluation Scheme:

Threaded binary tree definition -1 mark

Threaded binary tree advantages-1 mark

Explanation with proper example-2 marks

Solution:

Inorder traversal of a Binary tree is either being done using recursion or with the use of a auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).

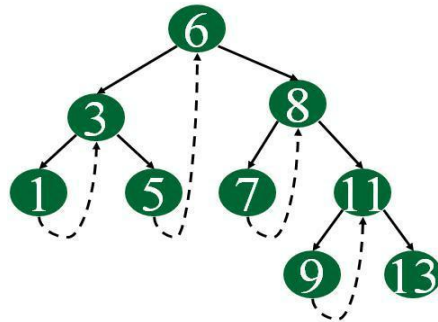
There are two types of threaded binary trees.

Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)

Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.

The threads are also useful for fast accessing ancestors of a node.

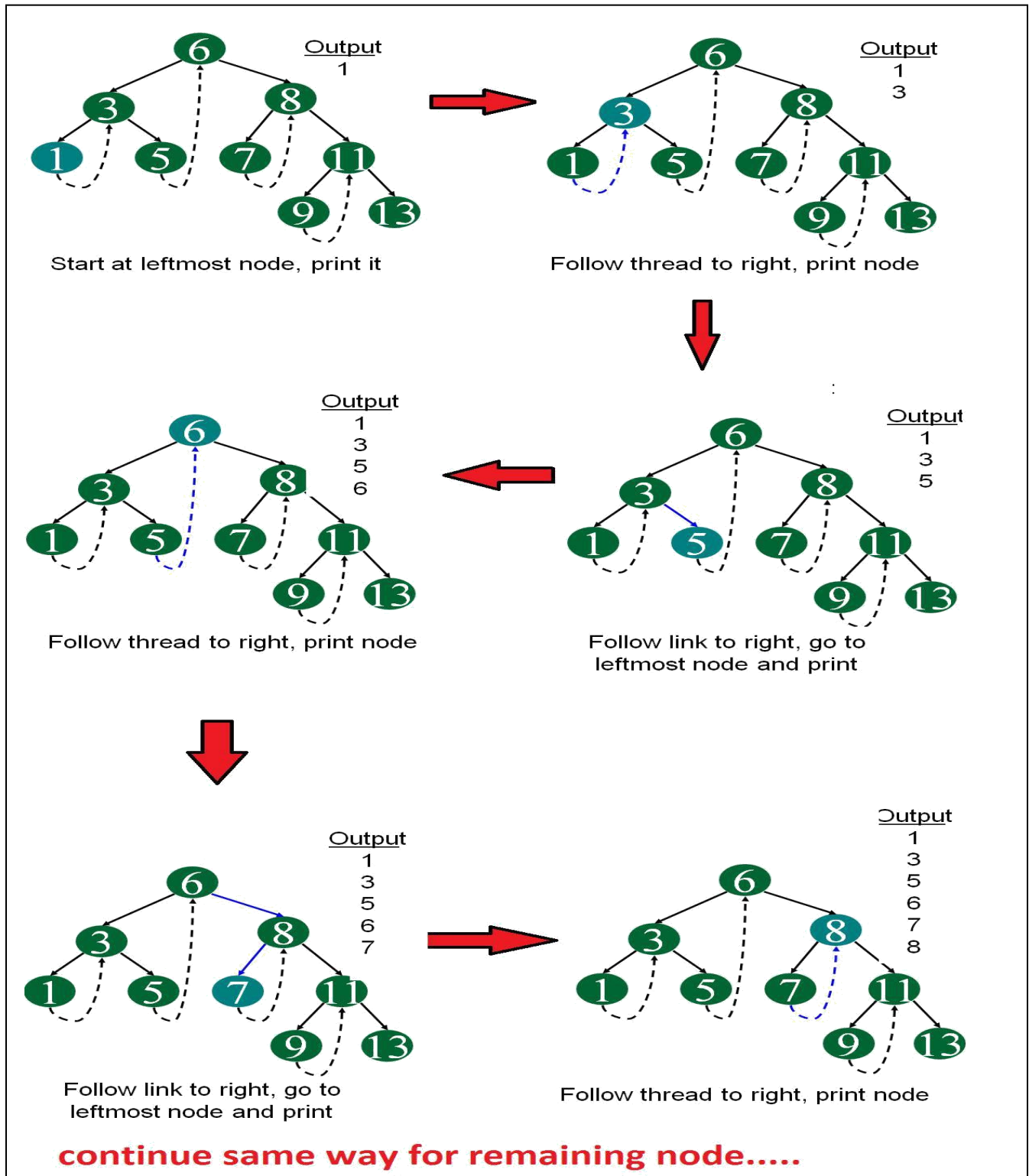
Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads.



Advantages of threaded binary tree:

- 1) By doing threading we avoid the recursive method of traversing a Tree, which makes use of stack and consumes a lot of memory and time.
- 2) The node can keep record of its root.

Explanation of in order traversal of threaded binary tree with example



In order traversal of the tree is:

1
3
5
6
7
8
9
11
13

The End