



AUTUMN END SEMESTER EXAMINATION-2018

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

1. Answer all the questions.

[1X10]

(a) Write a pseudo code to traverse a binary tree in in-order using user defined stack.

Evaluation scheme: Full mark will be awarded for the correct logic/method. Step-wise marks may be awarded judiciously depending on the partial correctness in the solution.

Solution:

```
void inOrder(struct tNode *root)
{
    struct tNode *current = root;
    struct sNode *s = NULL; //assume stack is implemented as linkedlist and sNode is pointing to top
    bool done = 0;

    while (!done)
    {
        if(current != NULL)
        {
            push(&s, current);
            current = current->left;
        }
        else{
            if (!isEmpty(s))
            {
                current = pop(&s);
                printf("%d ", current->data);
                current = current->right;
            }
            else
                done = 1;
        }
    }
}
```

(b) How many different binary search trees (BST) can be constructed from 4 different values?

Evaluation scheme: Full mark will be awarded for the correct logic/method. No partial mark to be awarded.

Solution: 14

(c) Write a postfix expression for the following prefix.

Prefix: $+a/-b^c*de-+fghj$

Evaluation scheme: exact answer - full mark. Step wise (correct) up-to some level - half mark.

Solution:

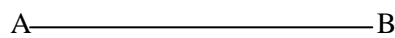
Postfix: $abcde*^{\wedge}-fg+h-/j/+$

(d) Define following terminologies in graph with suitable examples.

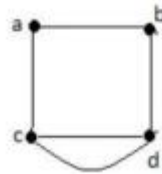
Evaluation scheme: each terminology (definition with example) carries 0.25 marks

Solution:

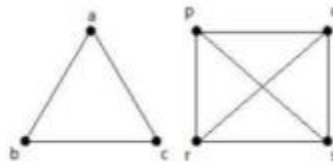
(i) Pendant vertex: A vertex of degree one is called a pendant vertex.



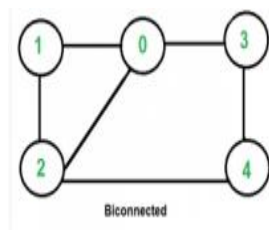
(ii) Multi-graph: A graph having parallel edges is known as a Multi-graph.



(iii) Complete graph: If a vertex is connected to all other vertices in a graph, then it is called a complete graph.



(iv) Bi-connected graph: A graph is bi-connected if and only if any vertex is deleted, the graph remains connected.



(e) Construct a binary tree with the following set of traversal sequences. The alphabet represents the node information.

Pre-order Traversal: ABCDEGKLHFIMNPOJ

In-order Traversal: ACKGLEHDMIPNOFIB

Evaluation scheme: if attempted, full mark to be awarded.

(f) Write a recursive function to find the height of the binary tree.

Evaluation scheme: Full mark will be awarded for the correct logic/method. No partial mark to be awarded.

Solution:

```
int maxDepth(struct node* node)
{
    if (node==NULL)
        return 0;
    else
    {
        /* compute the depth of each subtree */
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);

        /* use the larger one */
        if (lDepth > rDepth)
            return(lDepth+1);
        else return(rDepth+1);
    }
}
```

(g) Justify the best case and worst case time complexity of insertion sort.

Evaluation scheme: Full mark will be awarded for the correct logic/method. No partial mark to be awarded.

Solution: The best case input is an array that is already sorted. In this case insertion sort has a linear running time (i.e., $O(n)$). During each iteration, the first remaining element of the input is only compared with the right-most element of the sorted subsection of the array. The simplest worst case input is an array sorted in reverse order. The set of all worst case inputs consists of all arrays where each element is the smallest or second-smallest of the elements before it. In these cases every iteration of the inner loop will scan and shift the entire sorted subsection of the array before inserting the next element. This gives insertion sort a quadratic running time (i.e., $O(n^2)$).

h) Complexity of the code segment:

```
i=1;
while(i<n) {
    count = 0;
    for(j=n/i; j<=n; j=j+(n/i)) {
        count++;
    }
    i += count;
}
```

Evaluation scheme: Full mark will be awarded for the correct complexity. No partial mark to be awarded.

Solution: $O(n)$

(i) How data structures and abstract data types are related to each other? Justify that STACK can be an abstract data type?

Evaluation scheme: Full mark will be awarded for the correct logic/method. Step-wise marks may be awarded judiciously depending on the partial correctness in the solution.

Solution: A data structure is the implementation of an ADT. When we say the stack ADT, we refer to the stack data type which has set of defined operations, the operations constraints, and the possible values. One way of describing the stack is as a last in, first out (LIFO) abstract data type and linear data structure. A stack can have any abstract data type as an element, but is characterized by two fundamental operations, called push and pop (or pull).

(j) Write a modified bubble sort algorithm to minimize the number of comparison in case of best case.

Evaluation scheme: Full mark will be awarded for the correct logic/method. Step-wise marks may be awarded judiciously depending on the partial correctness in the solution.

Solution:

```
void modified_bubbleSort(int arr[], int n)
{
    int i, j, flag=1;
    for (i = 0; i < (n-1) && flag == 1; i++)
    {
        flag = -1;
        for (j = 0; j < (n-i-1); j++)
        {
            if (arr[j] > arr[j+1])
            {
                swap(arr[j], arr[j+1]);
                flag = 1;
            }
        }
    }
}
```

2. (a) Write a function to perform the following operations on the node of a binary search tree. [4]

- i.) If the degree of the node is two, delete it.**
- ii.) If the degree of the node is one, then add a child node with a value one more or less than the node depending on the position of the existing child node.**

Evaluation scheme: part (i) – 2 marks and part (ii) – 2 marks. Step-wise marks may be awarded judiciously depending on the partial correctness in the solution.

Solution:

```
void operation_BST(struct node *ptr)
{
    if(ptr->lchild != NULL && ptr->rchild != NULL)
    {
        struct node *p, *q = ptr->lchild ;
```

```

while(q->rchild != NULL)
{
    p = q;
    q = q->rchild;
}
ptr->info = q->info;
delete(q); // delete function to delete the node with one child or leaf node or use free(q)
} // end of if (Bit i)
else if(ptr->lchild != NULL && ptr->rchild == NULL) || (ptr->lchild == NULL && ptr->rchild !=
NULL)
{
    struct node *temp = (struct node *)malloc(sizeof*(struct node));
    temp->lchild = temp->rchild = NULL;
    if(ptr->rchild == NULL)
    {
        temp->info = ptr->info + 1;
        ptr->rchild = temp;
    }
    else
    {
        temp->info = ptr->info - 1;
        ptr->lchild = temp;
    }
} // end of else if (Bit ii)
}

```

2. (b) Consider two singly linked list *L1 and *L2. Write a function to multiply both the linked list as Cartesian product, so the final list *L3 must only contain unique elements. [4]

Evaluation scheme: Cartesian function - 3 Mark and search - 1 Mark. Step-wise marks may be awarded judiciously depending on the partial correctness in the solution.

Solution:

```

struct node * cartesian(struct node *L1, struct node *L2)
{
    struct node *p, *q, *L3, *t ;
    L3=NULL;
    int flag = 0 ;
    for(p = L1; p != NULL; p = p->next)
    {
        for(q = L2; q != NULL; q=q->next)
        {
            flag = search(L3, p->info * q->info);
            if (flag ==0)
            {
                struct node *temp = (struct node *)malloc(sizeof*(struct node));
                temp->next = NULL;
                temp->info = p->info * q->info;
                if(L3==NULL)
                    L3=temp;
            }
            else

```

```

{
    t=L3;
    while(t->next != NULL)
        t=t->next;
    t->next=temp;
}
} // end of if
} // end of second for
} // end of first for
return (L3);
} //end of function

```

```

int search(struct node *p, int val)
{
    struct node *t= p;
    if(P==NULL)
        return 0;
    else
    {
        while(t != NULL)
        {
            if(t->info == val) return 1;
            t = t->next;
        }
        return 0;
    }
} //end of function search

```

3. (a) Write a C function to display n^{th} element from the end of a single linked list. [4]

Evaluation scheme: Full mark for the correct answer. Step-wise marks may be awarded depending on the partial correctness in the solution.

Solution:

```

void printNthFromLast(struct node* head, int n)
{
    int len = 0, i;
    struct node *temp = head;

    // count the number of nodes in Linked List
    while (temp != NULL)
    {
        temp = temp->next;
        len++;
    }

    // check if value of n is not more than length of the linked list
    if (len < n)

```

```

return;

temp = head;

// get the (len-n+1)th node from the beginning
for (i = 1; i < len-n+1; i++)
temp = temp->next;

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

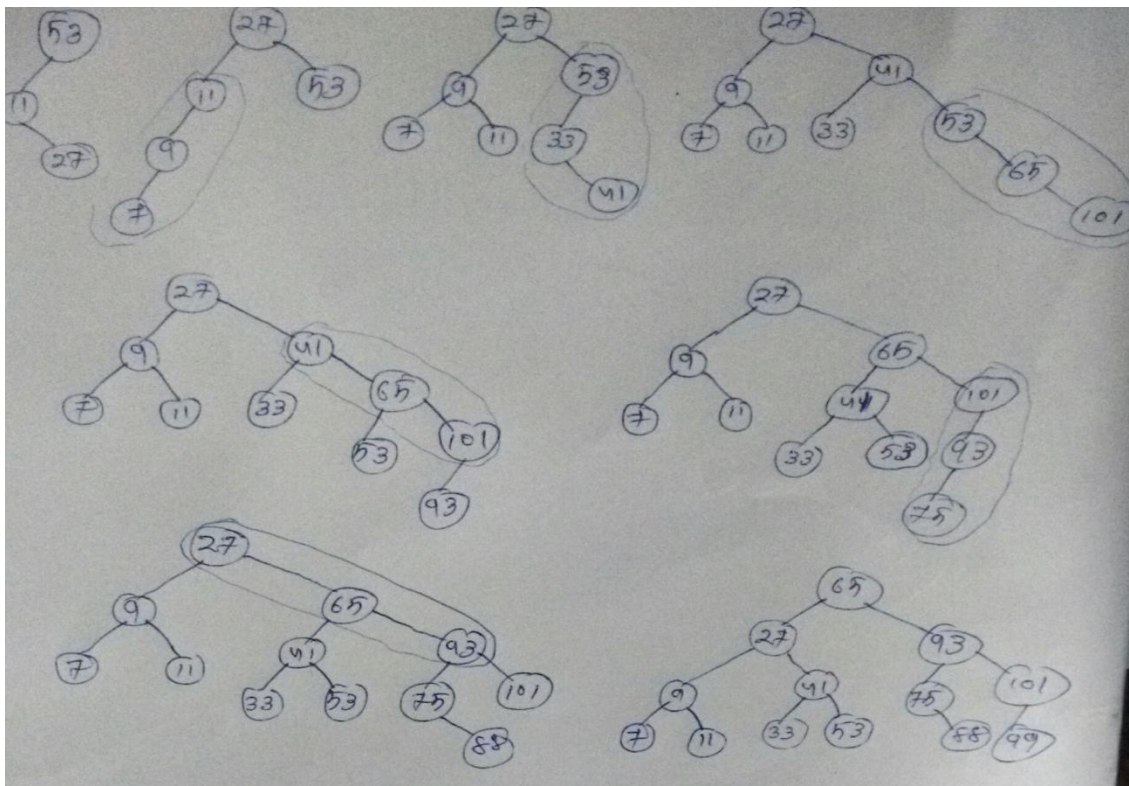
```

3. (b) What is a height balanced binary search tree known as? How it is different from binary search tree? Construct a height balanced binary search with the following elements where the elements are added to the tree one after the other in the given sequence. Mention the type of rotation applied wherever required. [4]

Evaluation scheme: What is a height balanced binary search tree known as - 0.5 mark. How it is different from binary search tree - 0.5 mark. Construction of height balanced binary search – 3 marks. Step-wise marks may be awarded only to the “Construction of height balanced binary search” depending on the partial correctness in the solution.

Solution:

A height balanced binary search tree is known as AVL tree after the inventors Adelson-Velskii and Landis. A node in a tree is height-balanced if the heights of its subtrees differ by no more than 1.



4. (a) Let a programming language does not support the structure kind of user defined data type where multiple types of elements can be stayed together. In order to store n number of students roll numbers and marks we prefer to store these information in a single integer array considering both the roll number and the marks as integer type. We store the information in the following manner, where 65,43, 54, 75 and 32 are the marks of the students with roll numbers 102, 194, 176, 201 and 146 respectively. [4]

102	65	194	43	176	54	201	75	146	32
-----	----	-----	----	-----	----	-----	----	-----	----

So, write an insertion sort function to sort the array based on the student mark.

Evaluation scheme: Full mark for the correct answer. Step-wise marks may be awarded judiciously depending on the partial correctness of the solution.

Solution:

```
void insertionSort(int arr[], int n)
{
    int roll[n/2];
    int mark[n/2];
    int x = 0;
    for(int i=0; i<n; i+=2) {
        roll[x] = arr[i];
        mark[x] = arr[i+1];
        x++;
    }
    int j;
    for (int i = 1; i < n/2; i++) {
        int key = mark[i];
        int rollno = roll[i];
        j = i-1;
        while (j >= 0 && mark[j] > key)
        {
            mark[j+1] = mark[j];
            roll[j+1] = roll[j];
            j = j-1;
        }
        roll[j+1] = rollno;
        mark[j+1] = key;
    }
    x=0;
    for(int i=0; i<n; i+=2)
    {
        arr[i] = roll[x];
        arr[i+1] = mark[x];
        x++;
    }
}
```

4. (b) Write PUSH and POP function for QUEUE using STACK data structure. [4]

Evaluation scheme: Full mark for the correct answer. Step-wise marks may be awarded judiciously depending on the partial correctness of the solution.

Solution:

```

int topa=-1,topb=-1,maxa,maxb;
int *stacka,*stackb;
stacka=(int*)malloc(maxa*sizeof(int));
stackb=(int*)malloc(maxb*sizeof(int));

```

```

void PUSH()
{
    int num;
    scanf("%d",&num);
    if(topa==maxa-1)
        printf("overflow\n");
    else
        stacka[++topa]=num;
}

```

```

void POP()
{
    if(topa== -1 && topb== -1)
        printf("underflow\n");
    else if(topb== -1)
    {
        while(topa!=0)
            stacka[topa--]=stackb[++topb];
        printf("element popped:%d\n",stacka[topa]);
        topa=-1;
    }
    else
    {
        printf("element popped:%d\n", stackb[topb--]);
    }
}

```

5. (a) Given a number $n = abc$ and with the hash function $f(n) = (a*3^0 + b*3^1 + c*3^2) \bmod 9$. Then generate the hash table by following linear and quadratic probing resolution separately for the following data set: 476,192,215,729,318,620,586,828,434 [4]

Evaluation Scheme: Correct key generation = 2 marks, Linear Probing = 1 mark, Quadratic Probing = 1 mark. Step-wise marks may be awarded judiciously depending on the partial correctness of the solution.

Solution:

Hash key for **476** = $(4*3^0 + 7*3^1 + 6*3^2) \bmod 9 = 79 \bmod 9 = 7$

							476	
0	1	2	3	4	5	6	7	8

For **192** = $(1*3^0 + 9*3^1 + 2*3^2) \bmod 9 = 46 \bmod 9 = 1$

	192						476	
0	1	2	3	4	5	6	7	8

For **215** = $(2*3^0 + 1*3^1 + 5*3^2) \bmod 9 = 50 \bmod 9 = 5$

	192				215		476	
0	1	2	3	4	5	6	7	8

For **729** = $(7*3^0 + 2*3^1 + 9*3^2) \bmod 9 = 94 \bmod 9 = 4$

	192			729	215		476	
0	1	2	3	4	5	6	7	8

For **318** = $(3*3^0 + 1*3^1 + 8*3^2) \bmod 9 = 78 \bmod 9 = 6$

	192			729	215	318	476	
0	1	2	3	4	5	6	7	8

For **620** = $(6*3^0 + 2*3^1 + 0*3^2) \bmod 9 = 12 \bmod 9 = 3$

	192		620	729	215	318	476	
0	1	2	3	4	5	6	7	8

For **586** = $(5*3^0 + 8*3^1 + 6*3^2) \bmod 9 = 83 \bmod 9 = 2$

	192	586	620	729	215	318	476	
0	1	2	3	4	5	6	7	8

For **828** = $(8*3^0 + 2*3^1 + 8*3^2) \bmod 9 = 86 \bmod 9 = 5$

Now there is a collision, it can be solved in following 2 methods:

- (i) Linear probe 2 : hash key = $(86 + 1) \bmod 9 = 6$
 Linear probe 3 : hash key = $(86 + 2) \bmod 9 = 7$
 Linear probe 4 : hash key = $(86 + 3) \bmod 9 = 8$

	192	586	620	729	215	318	476	828
0	1	2	3	4	5	6	7	8

- (ii) Quadratic Probing 1 : hash key = $(86 + 1*1) \bmod 9 = 6$

Quadratic Probing 2 : hash key = $(86 + 2*2) \bmod 9 = 0$

828	192	586	620	729	215	318	476	
0	1	2	3	4	5	6	7	8

For **434** = $(4*3^0 + 3*3^1 + 4*3^2) \bmod 9 = 49 \bmod 9 = 4$

Now again there is a collision, again following 2 methods can be employed:

- (i) Linear Probe 2 : hash key = $(49 + 1) \bmod 9 = 5$
 Linear probe 3 : hash key = $(49 + 2) \bmod 9 = 6$
 Linear Probe 4 : hash key = $(49 + 3) \bmod 9 = 7$
 Linear Probe 5 : hash key = $(49 + 4) \bmod 9 = 8$

Linear Probe 6 : hash key = $(49 + 5) \bmod 9 = 0$

434	192	586	620	729	215	318	476	828
0	1	2	3	4	5	6	7	8

- (ii) Quadratic probe 2 : hash key = $(49 + 1) \bmod 9 = 5$

Quadratic probe 3 : hash key = $(49 + 4) \bmod 9 = 8$

828	192	586	620	729	215	318	476	434
0	1	2	3	4	5	6	7	8

5. (b) "If the height of a tree is reduced and balanced, then the searching time also get reduced." (True/ False) Justify. Construct a B-tree of order three with the following set of elements where the elements are added to the tree one after the other in the given sequence.
23, 64, 48, 96, 101, 34, 55, 11, 22, 41, 89, 71, 78, 61, 83, 94, 8, 27, 35, 1 [4]

Evaluation Scheme: Justification - 1 mark, correct construction of B-tree - 3 marks. Step-wise marks may be awarded by looking at the correct approach even if some minor mistakes are present.

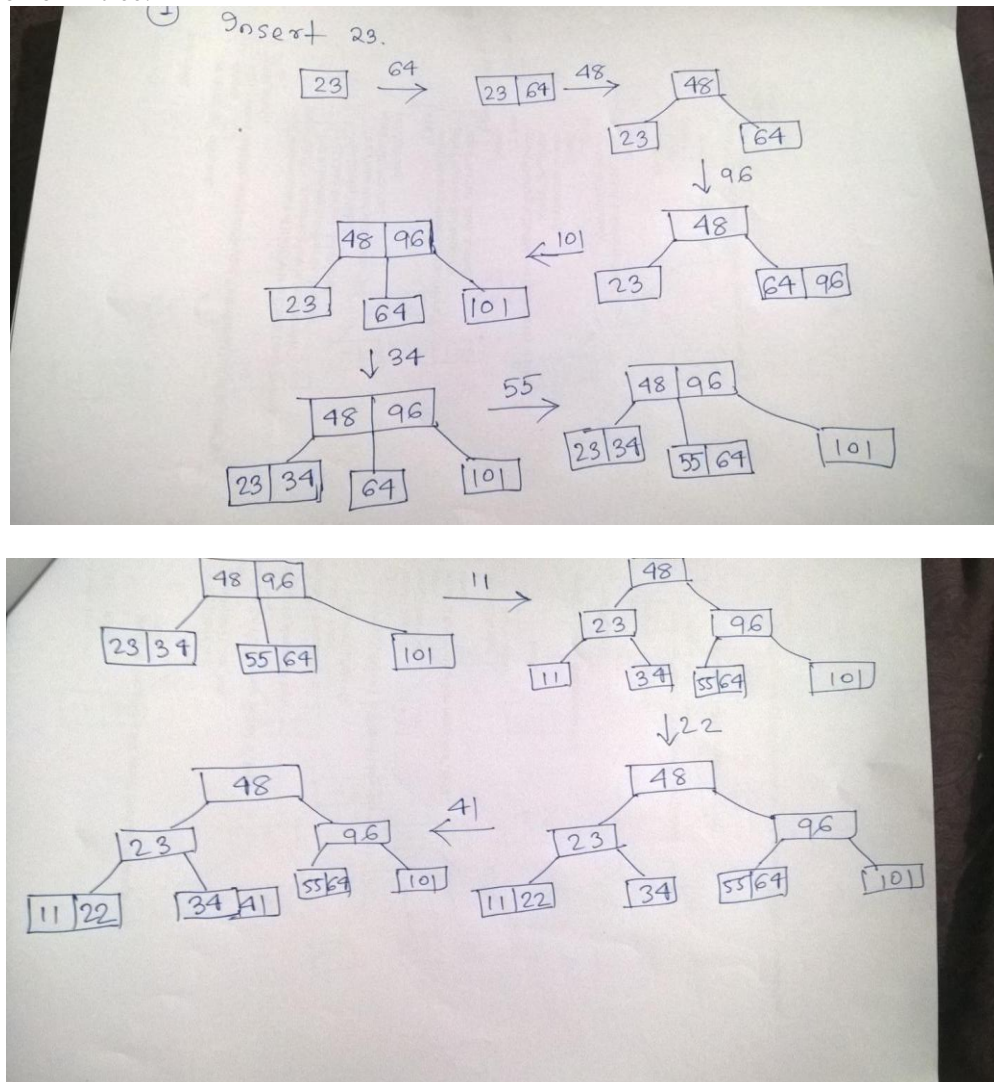
Solution:

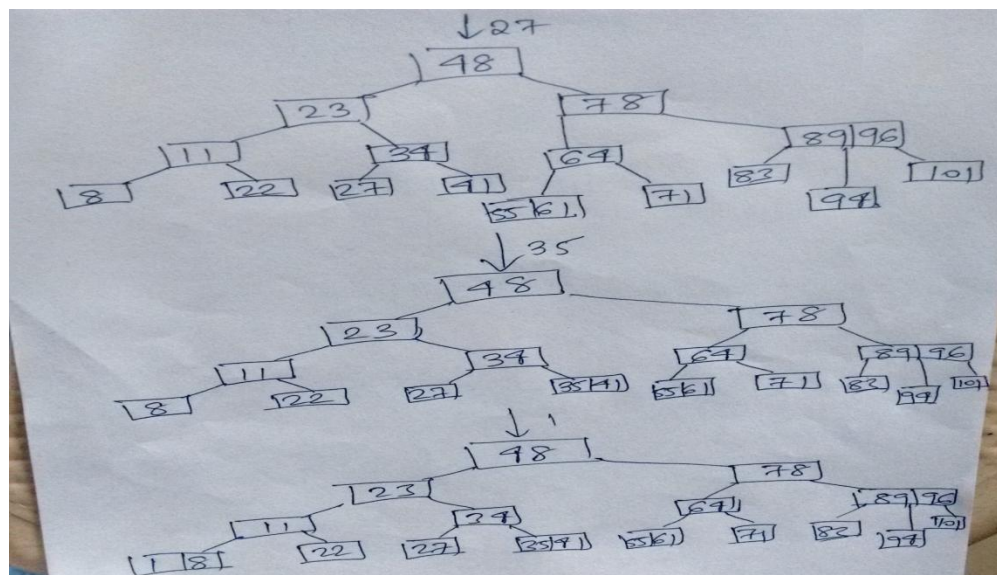
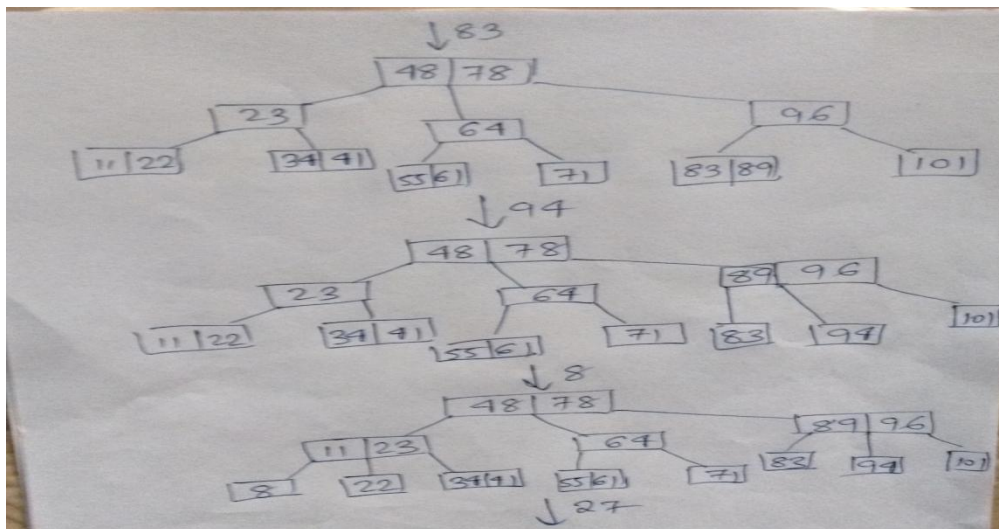
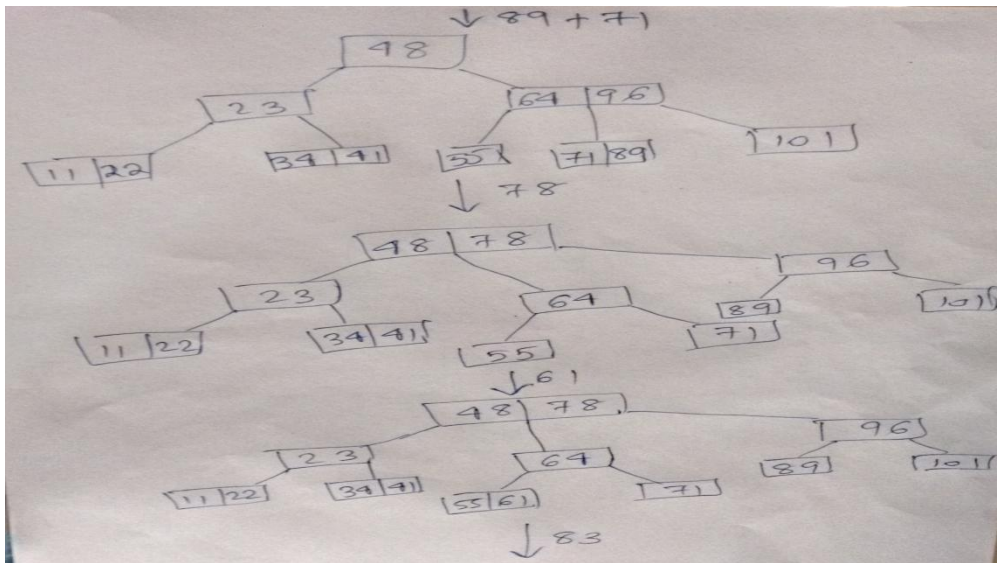
True. The worst case complexity for a single search in a balanced binary search tree is $O(\log(n))$.

The worst case complexity for a single search in an unbalanced binary search tree is $O(n)$.

In any binary search tree, the time complexity for searching is $O(h)$, where h is the height of the tree. In the worst case the maximum height h of the binary tree is $N-1$. This occurs in a case when every element that is added to the tree is greater than the previous element. So, in this case searching for an element would take $O(h) = O(N-1) = O(N)$ time complexity. The height of the Binary search tree can be reduced by balancing the tree. So the height h will become $\log(n)$. Hence time complexity for searching will become $O(\log(n))$.

Construction of B-tree:





6. (a) Consider a matrix of $M \times N$. Define two function for sorting each row of the matrix individually and sorting each column of the matrix individually. Write a C program to suggest the end users which order the sorting should be carried out so that we can get maximum total number of sorted rows and sorted columns. [4]

Evaluation Scheme:

Function to sort each row - 1 mark, Function to sort each column - 1 mark. Program to suggest the end user – 2 marks. Depending upon the steps, partial marks may be awarded judiciously.

Solution:

Let `sort_row()` and `sort_column()` are the two functions to sort rows and columns of the matrix respectively. `F1()` is the function to suggest the end users which order the sorting should be carried out so that we can get maximum total number of sorted rows and sorted columns. `nr` and `nc` represents total number of rows and total number of columns in the matrix. `A[][]` is the array which store the matrix.

```
void sort_row (int nr, int nc, int A[][])
{
    int i=nr;
    for (int j = 0; j < nc; j++) // loop for column of matrix
    {
        for (int k = 0; k < nc - j; k++) // loop for comparison and swapping
        {
            if (m[i][k] > m[i][k + 1])
            {
                // swapping of elements
                int t = m[i][k];
                m[i][k] = m[i][k + 1];
                m[i][k + 1] = t;
            }
        }
    }
}
```

```
void sort_column (int nr, int nc, int A[][])
{
    int i=nc;
    for (int j = 0; j < nr; j++) // loop for column of matrix
    {
        for (int k = 0; k < nr - j; k++) // loop for comparison and swapping
        {
            if (m[k][i] > m[k + 1][i])
            {
                // swapping of elements
                int t = m[k][i];
                m[k][i] = m[k+1][i];
                m[k+1][i] = t;
            }
        }
    }
}
```

```

void F1 (int nr, int nc, int A[][])
{
    if (nc<nr)
    {
        for (i=0; i<nc;i++)
        {
            sort_column (nr, i, A);
        }
        for (i=0; i<nr;i++)
        {
            sort_row (i, nc, A);
        }
    }
    else
    {
        for (i=0; i<nr; i++)
        {
            sort_row (i, nc, A);
        }
        for (i=0; i<nc; i++)
        {
            sort_column (nr, i, A);
        }
    }
}

```

6. (b) An arithmetic expression has different token including different parenthesis is like ‘(, ’, ‘{, ’, ‘[, ’]. Write a pseudo code to check the correct order and pairing of all the parenthesis? [4]

Evaluation Scheme:

4 marks for the correct answer. Depending upon the steps, partial marks may be awarded judiciously.

Solution:

- 1) Declare a character stack S.
- 2) Now traverse the expression string exp.
 - a) If the current character is a starting bracket (‘(’ or ‘{’ or ‘[’) then push it to stack.
 - b) If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack and if the popped character is the matching starting bracket then fine else parenthesis are not balanced.
- 3) After complete traversal, if there is some starting bracket left in stack then “not balanced”.

7. (a) How to represent a graph ADT in memory? Which data structures are used to implement breadth-first-search algorithm (BFS) and depth-first-search (DFS) algorithm? Briefly discuss the working of BFS algorithm with a suitable example. [4]

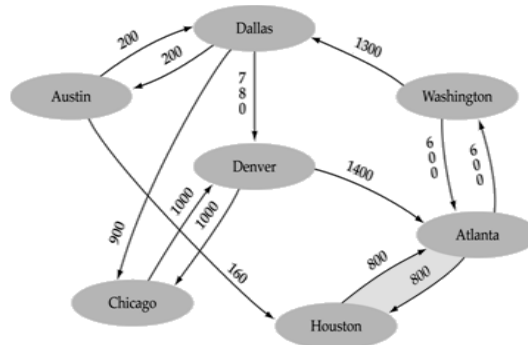
Evaluation Scheme:

Representation of graph ADT in memory – 1 mark. Data structure used in BFS and DFS – 0.5 mark each. Working of BFS with suitable example – 2 marks. Depending upon the steps, partial marks may be awarded judiciously.

Solution:

A graph can be represented by either adjacency list or adjacency matrix.

Weighted graph:

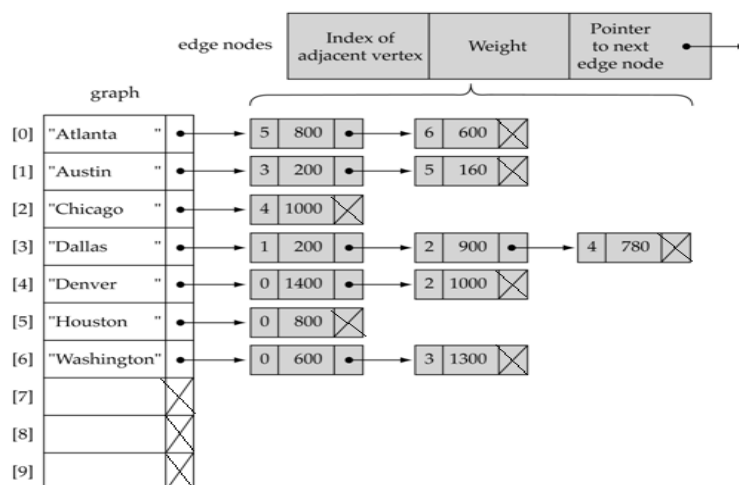


Adjacency matrix representation:

vertices		edges										
		[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	
[0]	"Atlanta"	0	0	0	0	0	800	600	*	*	*	
[1]	"Austin"	0	0	0	200	0	160	0	*	*	*	
[2]	"Chicago"	0	0	0	0	1000	0	0	*	*	*	
[3]	"Dallas"	0	200	900	0	780	0	0	*	*	*	
[4]	"Denver"	1400	0	1000	0	0	0	0	*	*	*	
[5]	"Houston"	800	0	0	0	0	0	0	*	*	*	
[6]	"Washington"	600	0	0	1300	0	0	0	*	*	*	
[7]		*	*	*	*	*	*	*	*	*	*	
[8]		*	*	*	*	*	*	*	*	*	*	
[9]		*	*	*	*	*	*	*	*	*	*	

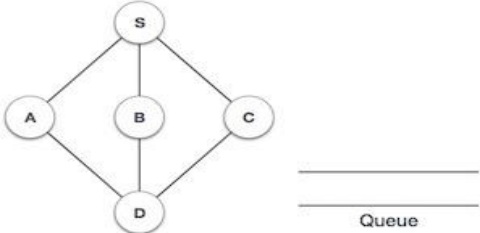
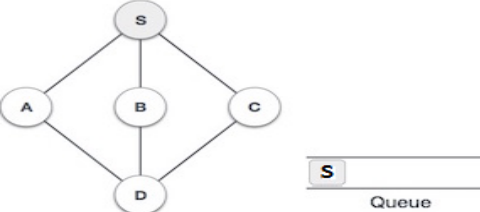
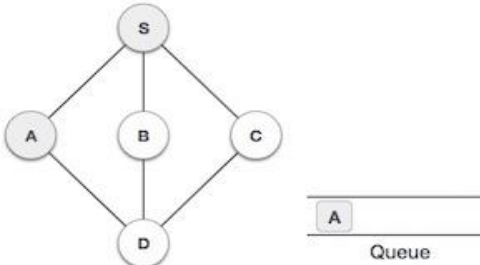
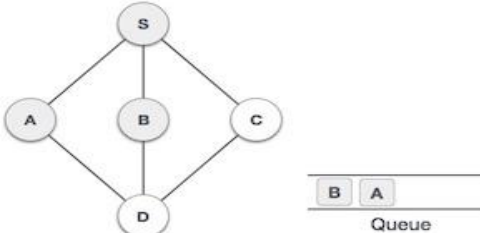
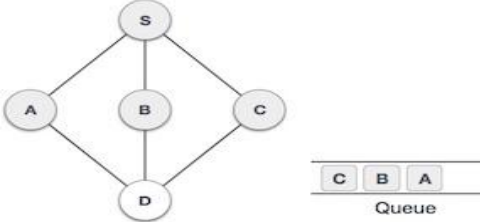
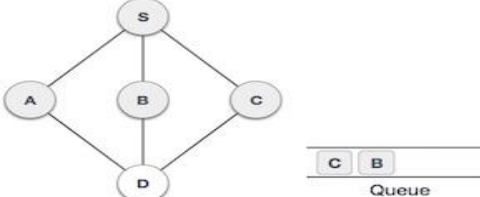
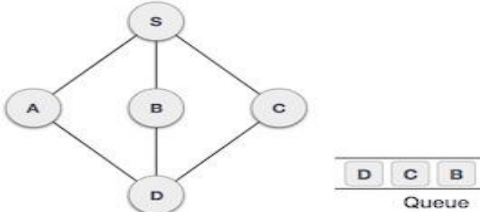
(Array positions marked '*' are undefined)

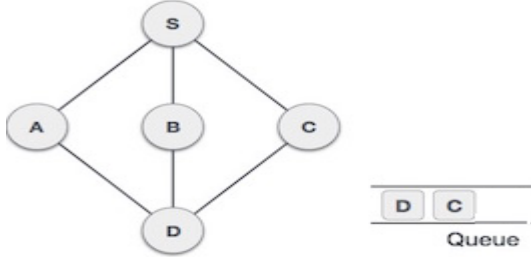
Adjacency list representation:



Queue data structure is used to implement BFS and stack is used to implement DFS.

BFS working with example:

Step #	Traversal	Description	O/P Seq
1		Initialize the Queue	--
2		We start from visiting S (starting node), and mark it visited and put it onto the queue	S
3		Queue is not empty and S is dequeued. We then see unvisited adjacent node from S. In this example, we have three nodes but alphabetically we choose A mark it visited and enqueue it.	S, A
4		Next unvisited adjacent node from S is B. We mark it visited and enqueue it.	S, A, B
5		Next unvisited adjacent node from S is C. We mark it visited and enqueue it.	S, A, B, C
6		Now S is left with no unvisited adjacent nodes. So we dequeue and find A.	S, A, B, C
7		From A we have D as unvisited adjacent node. We mark it visited and enqueue it.	S, A, B, C, D

8		At this stage we are left with no unmarked (unvisited) nodes. But as per algorithm we keep on dequeuing in order to get all unvisited nodes. When the queue gets emptied the algorithm is over.	S, A, B, C, D
---	---	---	---------------

Q7. (b) Given two sorted arrays A and B. Array A is full whereas array B is partially empty and number of empty slots are just enough to accommodate all elements of A. Give an algorithm to merge the two sorted arrays to fill the array B. [4]

Evaluation Scheme: 4 marks for the correct answer. Depending upon the steps, partial marks may be awarded judiciously.

Solution:

Input: A and B two sorted arrays. N is the number of elements present in A and M is number of elements present in B.

Output: Sorted array B

void sortedMerge(int A[], int B[], int N, int M)

```
{
    int i = N - 1;
    int j = M - 1;
    int lastIndex = N + M - 1;
    /* Merge A and B, starting from last element
    in each */
    while (i >= 0 && j >= 0) {
        /* End of a is greater than end of b */
        if (a[i] > b[j]) {
            B[lastIndex] = A[i];      // Copy Element
            i--;
        } else {
            B[lastIndex] = B[j];      // Copy Element
            j--;
        }
        lastIndex--; // Move indices
    }
    while (i >= 0) {
        B[lastIndex] = A[i];
        i--;
        lastIndex--; // Move indices
    }
    while (j >= 0) {
        B[lastIndex] = B[j];
        j--;
        lastIndex--; // Move indices
    }
}
```

8. Write pseudo code (Any Two)

[4 X 2]

- a) Implementation of one way threaded binary tree.**
- b) Finding a loop in the singly linked list.**
- c) Priority queue using Heap.**

Evaluation Scheme:

4 marks for the correct answer. Depending upon the steps, partial marks may be awarded judiciously.

Solution:

- a) Implementation of one way threaded binary tree.**

```
typedef struct threadedbinarytree
```

```
{
    int info;
    struct threadedbinarytree *left,*right;
    int rlink;
} TBT;
```

```
TBT *create()
```

```
{
    TBT *temp=(TBT *)malloc(sizeof(TBT));
    if(temp==NULL)
    {
        printf("Memory Allocation Error!");
        exit(1);
    }
    return temp;
}
```

```
TBT *makenode(int x) {
```

```
    TBT *temp=create();
    temp->info=x;
    temp->left=NULL;
    temp->right=NULL;
    temp->rlink=1;    /*1 for thread,0 for link*/
    return temp;
}
```

```
TBT *insert(TBT *root,TBT *newnode)
```

```
{
    TBT *p,*temp=root;
    if(root==NULL)
    {
        root=newnode;
        return root;
    }
}
```

```

while(temp!=NULL)
{
if(newnode->info<temp->info)
{
if(temp->left)
temp=temp->left;
else
{
temp->left=newnode;
newnode->left=NULL;
newnode->right=temp;
newnode->rlink=1; /*set during making node*/
break;
}
}
else
{
if(temp->right)
{
if(temp->rlink==1)
{
newnode->right=temp->right;
temp->right=newnode;
temp->rlink=0;
newnode->rlink=1; /*set during making node*/
break;
}
}
else
temp=temp->right;
}
else {
p=temp->right;
temp->right=newnode;
temp->rlink=0;
newnode->left=NULL;
newnode->right=p;
newnode->rlink=1;
break;
}
}
}
return root;
}

```

b) Finding a loop in the singly linked list.

```
int detect_loop(struct node *head)
{
    struct node *slowptr=head;
    struct node *fastptr=head;
    int loop_exist=0;
    while ((slowptr && fastptr && fastptr->next)!=NULL) /*this will find if there exists a loop or not*/
    {
        slowptr = slowptr->next;
        fastptr = fastptr->next->next;
        if(slowptr == fastptr)
        {
            loop_exist = 1;
            break;
        }
    }
    return loop_exist;
}
```

c) Priority queue using a heap.

```
struct node
{
    int priority;
    int info;
    struct node *next;
}*start=NULL,*q,*temp,*new;
```

```
typedef struct node N;
```

```
void insert()
{
    int item, itprio;
    new=(N*)malloc(sizeof(N));
    printf("ENTER THE ELT.TO BE INSERTED :\t");
    scanf("%d",&item);
    printf("ENTER ITS PRIORITY :\t");
    scanf("%d",&itprio);
    new->info=item;
    new->priority=itprio;
    new->next=NULL;
    if(start==NULL )
    {
        //new->next=start;
```

```

    start=new;
}
else if(start!=NULL&&itprio<=start->priority)
{
    new->next=start;
    start=new;
}
else
{
    q=start;
    while(q->next != NULL && q->next->priority<=itprio)
    {q=q->next;}
    new->next=q->next;
    q->next=new;
}
}
void display()
{
    temp=start;
    if(start==NULL)
        printf("QUEUE IS EMPTY\n");
    else
    {
        printf("QUEUE IS:\n");
        if(temp!=NULL)
            for(temp=start;temp!=NULL;temp=temp->next)
            {
                printf("\n%d priority =%d\n",temp->info,temp->priority);
            }
    }
}

```

NOTE: Solution can be written in different ways.



Faculty Consent

SL NO	Name	Signature
1	Dr. Aniya Ranjan Panda	Arpanda
2	Ms. Nishita Kundo	Nishita Kundo
3	Mr. Gananath Bhuyan	Cirish
4	Dr. Arup A. Acharya	Arup Acharya
5	Arundyn Ratan Suman	Arundyn
6	N Be Raja Isaac	N Be Raja
7	Lipika Mohanty	Lipika
8	Rajat Kumar Behara	Rajat Behara

Rajat Behara
5/12/18
Rajat Kumar Behara
(Course Coordinator)