Name: Manish Shrestha
Roll No: 9

Date ....
Page No. ....

Q.1.
@
→ An abstract data type is an abstraction of a data structure that provides only the interface to which the data structure must adhere.

In other words, we can say that abstract data types are the entities that are definitions of data and operations but do not have implementation details. In this case, we know that data we are storing and the operations that can't be performed on data, but we don't know about the implementation details. The reason for not having implementation details is that every programming language has different implementation strategy. for example:- a C data structure is implemented using structures while a C++ data structure is implemented using objects & classes.

Abstraction data type model
→ Before knowing about the abstract data type model, we should know about abstraction & encapsulation

Abstraction:- It is a technique of hiding the internal details from the user and only showing the necessary details to the user.

Encapsulation :- It is a technique of combining data and the member function in a single unit is known as encapsulation.

1. (b) (P+Q*R/S) + T*U - (V*W +X-Y)

Scanned Token	Stack operator	Postfix output
(	(	
P	(	P
+	(+	P
Q	(+	PQ
*	(+*	PQ
R	(+*	PQR
/	(+/	PQR*
S	(+/	PQR*S
)		PQR*S/
+	+	PQR*S/+
T	+	PQR*S/+T
*	+*	PQR*S/+T
U	+*	PQR*S/+TU
-	-	PQR*S/+TU*+
(	-(	PQR*S/+TU*+
V	-(	PQR*S/+TU* +V
*	-(*	PQR*S/+TU* +V
W	-(*	PQR*S/+TU* +VW
+	-(+	PQR*S/+TU* +VW*
X	-(+	PQR*S/+TU*+VW*X
-	-(-	PQR*S/+TU* +VW*X+
Y	-(-	PQR*S/+TU*+VW*X+Y
)	-	PQR*S/+TU*+VW*X+Y-
		PQR*S/+TU*+VW*X+Y--

Postfix : PQR*S/+TU*+VW*X+Y--

Q.2.

(a) → Recursion is a programming technique where a function calls itself to solve a problem by breaking it down into smaller subproblems. It involves solving a problem by reducing it to a simpler version of the same problem.

Factorial of a Positive integer

```
#include <iostream>
using namespace std;
int add (int n)
int main() {
  int n;
 cout <<"Enter a positive integer:";
 cin >>n;
 cout <<"Sum = "<<add(n);
 return 0;
}
```

```
int add (int n) {
  if (n! = 0)
    return n + add (n-1);
  return 0;
}
```

Q.2 (b)

→ Enqueue operation

→ Steps of enque operations are :-

- First, we will check whether the Queue is full or not.
- Initially, the front & rear are set to -1. When we insert the first element in a Queue, front and rear both are set to 0.
- When we insert a new element, the rear gets incremented i.e. rear = rear + 1.

Algorithm to insert an element in a circular queue

Step 1: IF (REAR + 1) % MAX = FRONT
Write "OVERFLOW"
Goto step 4
[End of IF]

Step 2: IF FRONT = -1 and REAR = -1
SET FRONT = REAR = 0
ELSE IF REAR = MAX - 1 and FRONT ! = 0
SET REAR = 0
ELSE
SET REAR = (REAR + 1) % MAX
[END OF IF]

Step 3: SET QUEUE [REAR] = VAL

Step 4: EXIT

→ Dequeue Operation
→ Steps are:-

- First, we check whether the Queue is empty or not. If the queue is empty, we cannot perform the dequeue operation.
- When the element is deleted, the value of front gets decremented by 1.
- If there is only one element left which is to be deleted, then the front and rear to -1.

Algorithm to delete an element from the circular queue.

Step 1: IF FRONT = -1
Write "UNDERFLOW"
Go to Step 4
[END of IF]

Step 2: SET VAL = QUEUE [FRONT]

Step 3: IF FRONT = REAR
SET FRONT = REAR = -1
ELSE
FRONT = MAX = -1
SET FRONT = 0
ELSE
SET FRONT = FRONT + 1
[END OF IF]
[END OF IF]

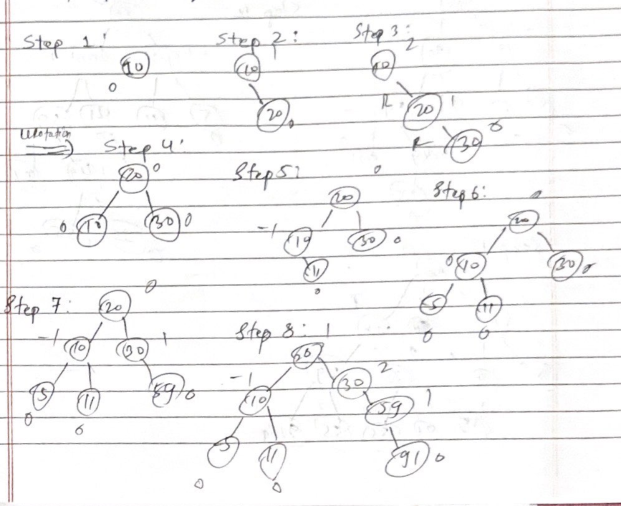
Step 4: EXIT

Q.3.
(b) Insert & Delete at the beginning of the singly linked list.

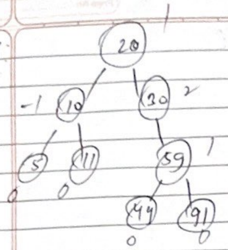
```
#include <iostream>
using namespace std;
void insertAtBeginning (int value) {
Node* newNode = new Node;
newNode → data = value;
newNode → next = head;
head = newNode;
}
```

```
void deletefromBeginning () {
  if (head == nullptr) {
  return;
  }
Node* temp = head;
head = head → next;
delete temp;
}
```

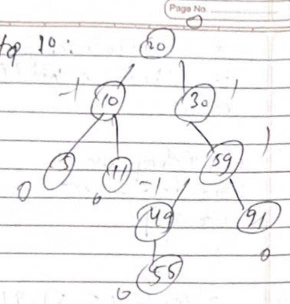
Q.4a 10, 20, 30, 11, 5, 59, 91, 49, 55, 21, 35, 15



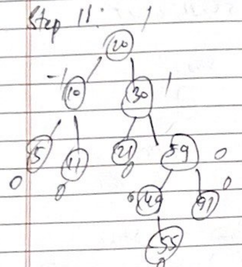
Step 9:



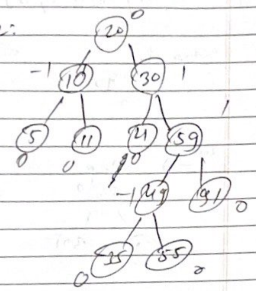
Step 10:



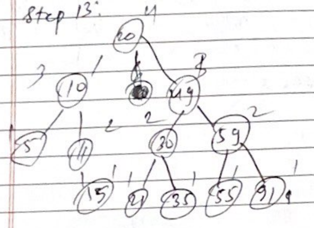
Step 11:



Step 12:



Step 13:



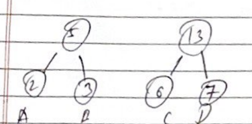
Q.4.
(b)

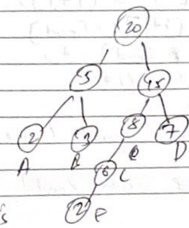
→ Huffman coding is an algorithm for compressing data with the aim of reducing its size without losing any of the details.

For example :- ABBCCDADDEBCCDDDDCCE

Scanned Token	Total no.
A	2
B	3
C	6
D	7
E	2

We calculate the total no. of values of variables and do in the way of BST as shown below.





→ In this way, Huffman is completed.

Q.5.

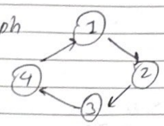
Implementation of quick sort.

```
# include <iostream>
using namespace std;
int partition (int a[ ], int start, int end)
{
int pivot = a [end ];
int i = (start-1);
for (int j= start; j <= end -1; j++)
{
 if (a[j] <pivot)
 {
 i++;
 int t = a [i];
 a[i] = a [j];
 a[j] = t;
 }
 }
int t= a [i+1];
a [i+1] = a [end];
a [end] = t;
return (i+1);
}
void quick (int a[], int start, int end)
{
 if (start <end)
 {
int p =partition (a,start, end);
quick (a, start, p-1);
quick (a, p+1, end);
}
}
void printArr (int a[], int n)
{
int i;
for (i=0 ; i<n; i++)
 cout <<a [i] <<" ";
}
int main()
{
int a[] ={23, 8, 28, 13, 18, 26};
int n=sizeof(a) /sizeof (a[]);
cout <<"Before sorting array elements are -\n";
print Arr (a,n);
quick (a,0,n-1);
cout <<"\nAfter sorting array elements are -\n";
printArr (a,n);
return 0;
}
```

Q.6.
(a)

→ Warshall's algorithm computes the transitive closure of a directed graph. It determines if there is a path from one node to another, considering all possible intermediate nodes. The algorithm maintains an adjacency matrix, initially filled with the graph's adjacency information.

(i) Digraph



(ii) where R=0

0	1	0	0
0	0	1	0
0	0	0	1
1	1	0	0

where R = 1

0	1	1	0
0	0	1	0
0	0	0	1
1	1	1	0

where R=2

0	1	1	1
0	0	1	1
0	0	0	1
1	1	1	1

where R = 3

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

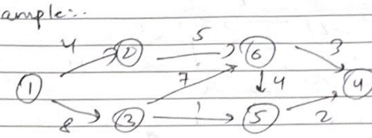
The transitive closure is:

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

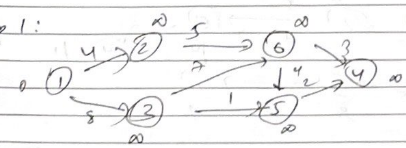
6.
(b)

→ Dijkstra's algorithm is a type of algorithm in which algorithm is completed in shortest distance by shortest value.

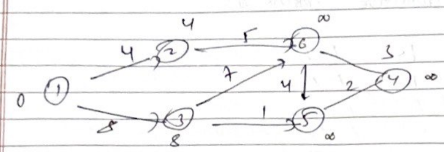
For example:-



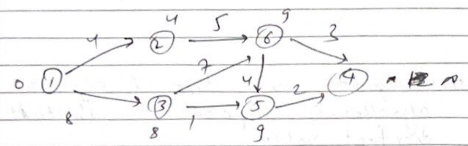
Step 1:



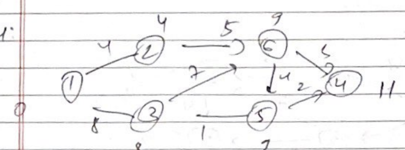
Step 2:



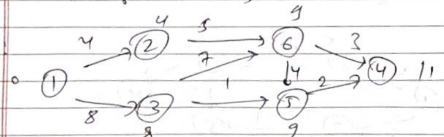
Step 3:



Step 4:



Step 5:



→ In this way, we can find the shortest path distance using Dijkstra's ~~short~~ algorithm.

Q.7

(a) Recursion vs Itieration

→ In recursion, a function calls itself to solve a problem where as in iteration, using loops to repeatedly execute code. Recursion solves problems with subproblems & Iteration solves using repetitive processes. or ~~to~~ simple loops. Recursion can make code more ~~consic~~ concise and elegant & in iteration, it may require more lines of code. Memory usage is less efficient in recursion & memory usage is more efficient in iteration. Recursion may have performance overhead due to function calls. Iteration often more performance for many tasks. Recursion may be trickier to debug in some cases & iteration is easier to debug.

(b) Binary Search

→ Binary Search is a divide & conquer algorithm for finding an element in a sorted array or list. It has a time complexity of O (logn)), making it highly efficient for large datasets.

Procedures ~~of~~ ~~BST~~ are :

1. Start with entire sorted array.
2. Compare the middle element to the target.
3. Adjust the search interval based ~~to~~ on the comparison (left or right half).
4. Repeat until ~~is~~ the target is found or the interval becomes empty.

→ The array must be sorted for binary search to work correctly. It's efficient and reduces ~~the~~ search space by half with each iteration. Binary Search only works on sorted data, so if the data isn't sorted, you'll need to sort it first, which can be time consuming. It is used in various fields including computer science, data structures and searching algorithms.