Name: Ayush Jain
SAP ID: 60004200132
Div: B1
Computer Engineering

17/02/22 Data Structures

Solutions :

→ 4) 1) A priority queue is a data structure in which each element is assigned a priority. The priority of the elements will be used to determine the order in which the elements will be processed.

2) The general rules of processing the elements of a priority queue are:

a) An element with higher priority is processed before an element with a lower priority.

b) Two elements with the same priority are processed on a first-come-first-served basis.

3) A priority queue can be thought of as a modified queue in which when an element has to be removed from the queue, the one with the highest priority is retrieved first.

4) Code :

```
# include <stdio.h>
# include <malloc.h>
# include <conio.h>
Struct node
{
    int data;
    int Priority;
    struct node *next;
}
```

```
struct node *start = NULL;
struct node *insert (struct node*);
struct node *delete (struct node*);
void display (struct node*);

int main ()
{
    int option;
    do
    {
        printf ("\n *main menu*");
        printf ("\n 1. Insert");
        printf ("\n 2. Delete");
        printf ("\n 3. Display");
        printf ("\n 4. Exit");
        printf ("\n Enter your option : ");
        scanf ("%d", &option);
    switch(option)
     {
        case 1:
                start = insert (start);
                break;
        case 2:
                start = delete (start);
                break;
        case 3:
                display (start);
                break;
    }
```

```
    } while (option != 4);
}
Struct node *insert (Struct node *start)
{
    int val, pri;
    Struct node *ptr, *p;
    ptr = (struct node*)malloc (sizeof (Struct node));
    Printf("\n Enter the value and its priority : ");
    Scanf(" %d %d, &val, &pri);
    ptr -> data = val;
    ptr -> priority = pri;
    if (Start == NULL || pri < start -> Priority)
    {
        Ptr -> next = Start;
        Start = ptr;
    }
    else
    {
        P = Start;
        while (P -> next != NULL && P -> next -> Priority <= pri)
            P = P -> next;
        Ptr -> next = P -> next;
        P -> next = Ptr;
    }
    return start;
}
```

```
struct node * delete (struct node * start)
{
    struct node * ptr ;
    if (start == NULL)
    {
        Printf ("\n Underflow");
        return;
    }
    else
    {
        ptr = start ;
        Printf ("\n Deleted item is : %d", ptr -> data );
        Start = start -> next;
        free (ptr);
    } ?
    return start;
}
void display (struct node * start)
{
    Struct node * ptr;
    Ptr = start ;
    if (start = NULL)
            Printf ("\n Queue is empty");
    else
    {
        Printf ("\n Priority queue is : ");
        while ( ptr != NULL) {
            Printf("\t %d [priority = %d], ptr -> data, ptr -> priority );
            Ptr = ptr -> next ; }
    }
}
```

OUTPUT:

```
* Main Menu *
1. Insert
2. Delete
3. Display
4. Exit
Enter your option : 1
Enter the value and its priority : 6  2
Enter the value and its priority : 8  1
Enter your option : 3
Priority Queue is :
 8 [Priority = 1]  6 [Priority=2 ]
Enter your option : 4
```

→ 2) a) Insert a node.

i) Function to Insert at the front of linked list.

```
void insertAtfront()
{
   int data;
   Struct node * temp;
   temp = malloc (sizeof (struct node));
   Printf ("\n Enter numbers to be inserted");
   Scanf (" %d", & data);
    temp → info = data;
    temp → link = start;
    start = temp;
}
```

ii) Function to insert at the end of linked list.

```
void insertAtEnd()
{
   int data;
   Struct node * temp, * head;
   temp = malloc(sizeof (struct node));
   Printf(" \n Enter number to be inserted");
   scanf(" %d", & data);
   temp → link = 0;
   temp → info = data;
   head = start;
```

```
while (head->link != NULL) {
        head = head->link;
    }
    head->link = temp;
}
```

iii) Function to ~~delete~~ insert at any specified position

```
void insertAtPosition()
{
    struct node *temp, *newnode;
    int pos, data, i = 1;
    newnode = malloc(sizeof(struct node));
    Printf("\n Enter position and data");
    Scanf("%d %d", &pos, &data);
    temp = start;
    newnode->info = data;
    newnode->link = 0;
    while (i < pos - 1) {
        temp = temp->link;
        i++;
    }
    newnode->link = temp->link;
    temp->link = newnode;
}
```

b) Delete a node

```
void deletePosition ()
  {
    struct node *temp, *position;
    int i = 1, pos;
  if (start == NULL)
      printf ("List is empty");
  else {
      printf ("Enter index : " );
      scanf ("%d", &pos );
      position = malloc (size of (struct node) );
      temp = start;
      while (i < pos - 1) {
          temp = temp -> link;
          i++;
        }
    Position = temp -> link;
    temp -> link = position -> link;
    free (position);
  }
}
```

6000420013 2

→3) a) Code:

```
# include <stdio.h>
# include <stdlib.h>
# include <ctype.h>
# include <string.h>
# define SIZE 100
char stack [size];
int top = -1;

void push (char item)
 {
     if (top >= SIZE -1)
      {
        Printf ("\n Stack Overflow");
      }
     else {
          top = top + 1;
          Stack[top] = item;
      }
 }
char pop()
 {
    Char item ;
    if (top < 0)
     {
       Printf (" Stack underflow");
       getchar ();
       exit (1);
     }
```

```
else
{
   item = stack[top];
   top = top - 1;
   return(item);
  }
}
int is_operator(char symbol)
{
  if(symbol=='^'||symbol=='*'||symbol=='/'||symbol=='+'||
                              symbol=='-')
     {
        return 1;
     }
     else {
          return 0;
        }
}
int precedence(char symbol)
{
   if(symbol=='^')
   {
      return(3);
   }
   else if(symbol=='*'||symbol=='/'){
         return(2);
   }
   else if(symbol=='+'||symbol=='-'){
         return(1);
   }
```

```
else {
        return(0);
      }
}
void InfixToPostfix (char infix_exp[], char postfix_exp[])
{
    int i, j;
    char item;
    char x;
    Push('(');
    strcat (infix_exp, ")");
     i=0;
     j=0;
     item = infix_exp[i];
    while(item != '\0')
    {
       if (item == '(') {
          Push (item);
       }
      else if ( isdigit (item) || isalpha (itemp) )
       {
          Postfix_exp[j] = item;
            j++;
       }
      else if (is_operator (x) == 1) && Precedence (x) >= Precedence (item))
       {
       x = pop ();
```

```
while (is_operator(x) == 1 && precedence(x) >= precedence(item))
    {
      postfix_exp[j] = x;
       j++;
       x = pop();
    }
   push(x);
   push(item);
 }
else if(item == ')')
 {
   x = pop();
   while (x != '(')
   {
     postfix_exp[j] = x
      j++;
     x = pop();
   }
}
else {
   printf("\n a Invalid infix expression");
   getchar();
   exit(1);
}
i++;
item= infix_exp[i];
}
```

```
if (top > 0)
{
    Printf (" Invalid infix expression");
     getchar ();
    exit (1);
}
postfix-exp[j] = '\0';

}

int main ()
{
  char infix[SIZE] , postfix[SIZE];
  Printf (" \n Enter Infix expression");
  gets (infix);
  InfixToPostfix (infix, postfix);
  Printf (" Postfix Expression: " );
  Puts (postfix);

  return 0;

}
```

600042001132

→3 b) Application of stack: Reversing the ~~last~~ data.

i) A stack can be used to reverse the characters of string. This can be achieved by simply pushing one by one each character onto the stack, which later can be popped from the stack one by one.

2) Because of last in first out (LIFO) property of stack, the first character is on bottom of stack and the last character is on top of stack and after performing the pop operation, the stack returns the string in reverse order.

POP operation

EMOCLEW

TOP → 7	E	7	
6	M	6	
5	O	5	
4	E	4	
3	L	3	
2	E	2	
1	W	1	

→ 1 a) The main condition for fibonacci search is that the array elements should be in sorted order.

eg: if the array is

A	10	20	30	40	50	60
i	0	1	2	3	4	5

If the number to be searched = X = 50
Number of elements in array = N = 6

Fibonacci Sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21 .....
Closest fibonacci number which is greater than N = 8

So the initial condition becomes.

Fibm = 8
fibm1 = 5
fibm2 = 3
offset = -1

For each pass / iteration we do the following

i = min(offset + fibm2, N - 1)

and then check the following a[i] and do proper operations.

6000420015

Case 1:	Case 2:	Case 3:
if a[i] > x	if a[i] < x	value found.
fibm = fibm2	fibm = fibm1	
fibm1 = fibm1 - fibm	fibm1 = fibm2	
fibm2 = fibm - fibm1	fibm2 = fibm1 - fibm1	
	offset = i	

iteration ①

$i = \min(-1+3, 5)$

$i = \min(2, 5)$

$i = 2$

Here $a[i] = a[2] = 30$ and $x = 50$

So $a[i] < x$

∴ we update : $fibm = fibm_1 = 5$

$fibm_1 = fibm_2 = 3$

$fibm_2 = 5 - 3 = 2$

offset = 2

iteration ②

$i = \min(offset + fibm_2, n-1)$

$i = \min(2+2, 5)$

$i = \min(4, 5)$

$i = 4$

Here $a[i] = a[4] = 50$ and $x = 50$

∴ So, we found the value 50 at **indx = 4**

∴ By using fibonacci search, we found the value 50 in 2 iteration at index 4.

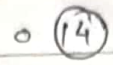
→1 b) Insertion Sort:

Code:

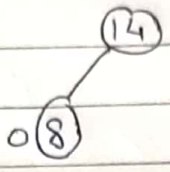
```
#include <stdio.h>
int main()
{
 int a[10],n,i;
 Printf(" Enter the number of elements in array : ");
 scanf("%d",&n);
 Printf(" Enter the elements of array : ");
for (i=0;i<n;i++)
 {
   scanf("%d",&a[i]);
 }
for(i=1; i<=n-1; i++)
 {
   int temp= a[i];
   int j = i-1;
   while(j>=0 && a[j]>temp)
    {
      a[j+1] = a[j];
      j--;
    }
   a[j+1] = temp;
 }
Printf(" The sorted array is : ");
for (i=0;i<n;i++)
 {  Printf ("%d ", a[i]);
 }
}
```

→5 a) Sol:

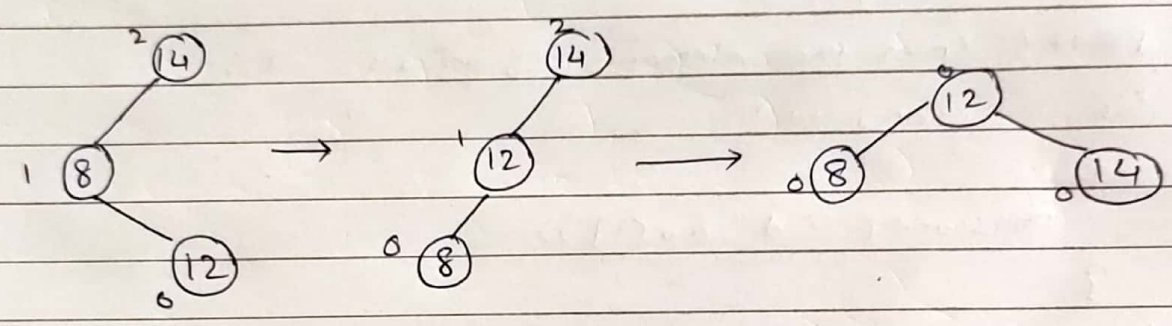
Insert (14)



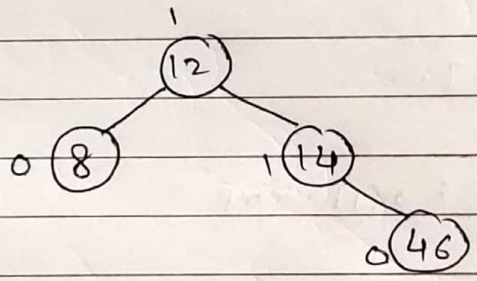
Insert (8)



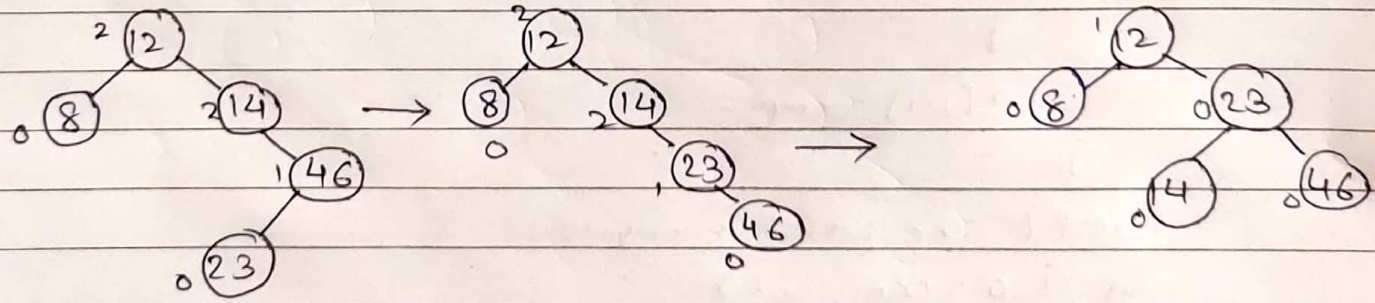
Insert (12)



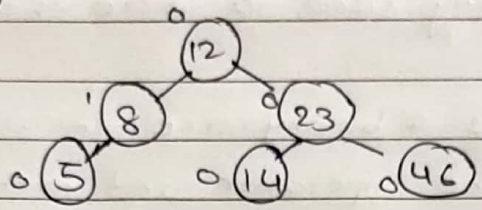
Insert (46)



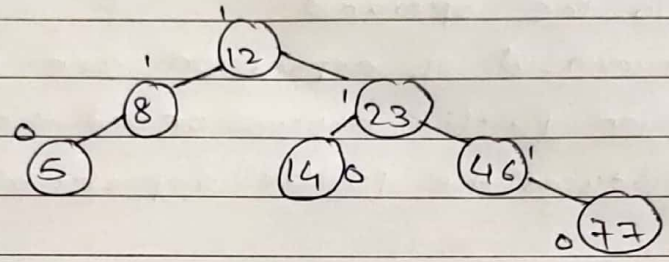
Insert (23)



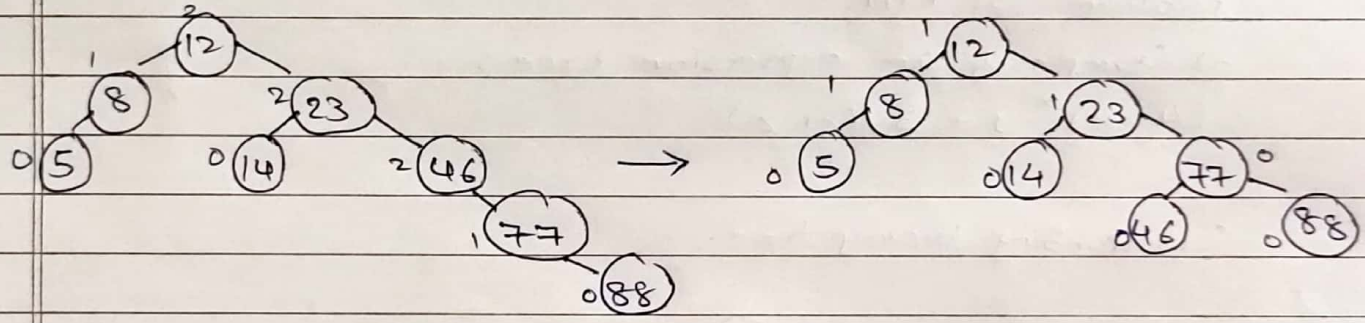
Insert (5)



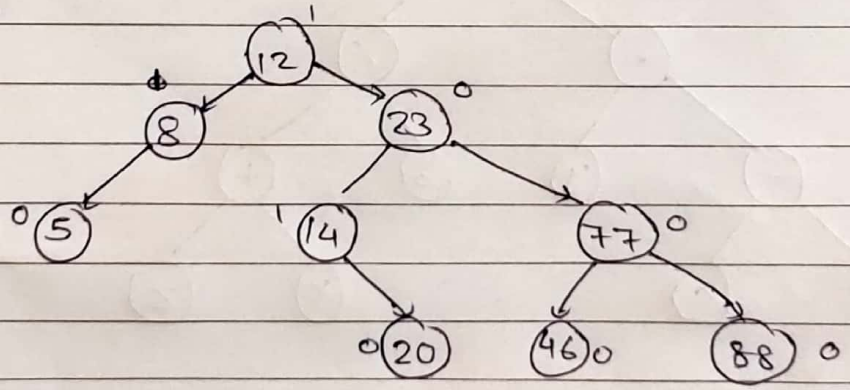
Insert (77)



Insert (88)



Insert (20)



→5 b) i) Binary Trees are widely used to store the algebraic expressions.

2) The Expression Tree is a binary tree in which each interval node corresponds to the operator and each leaf node corresponds to the operand.

3) In order traversal of expression tree produces infix version of given postfix expression and the post-order traversal produces the postfix-representation of the expression.

4) The postfix representation is also called reverse Polish notation or RPN.

5) Example of an expression tree:

(4 - x) * y + 3 / (x + z)

Corresponding binary tree: