09/10/2021

Data Structure and Algorithms

Applied Prep Course

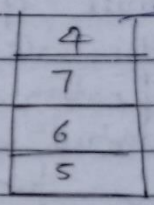
SATYAM SETH

PART-2

Stack (LIFO)

Page No.: 104
Date: 2/11/20

Last in First out



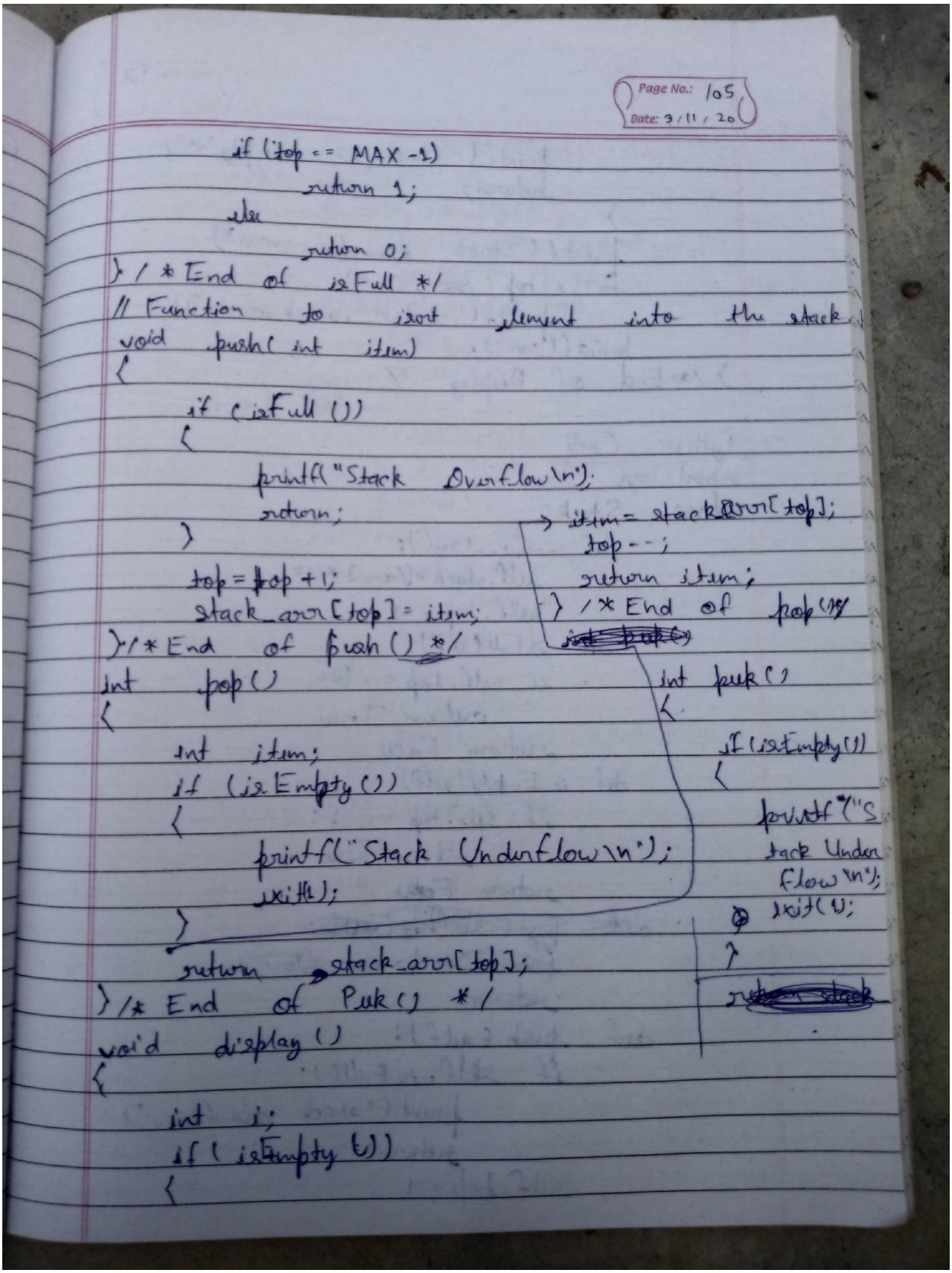
→ Applications -

i) Expression Evaluation
ii) Parenthesis Check
iii) In Recursion etc.

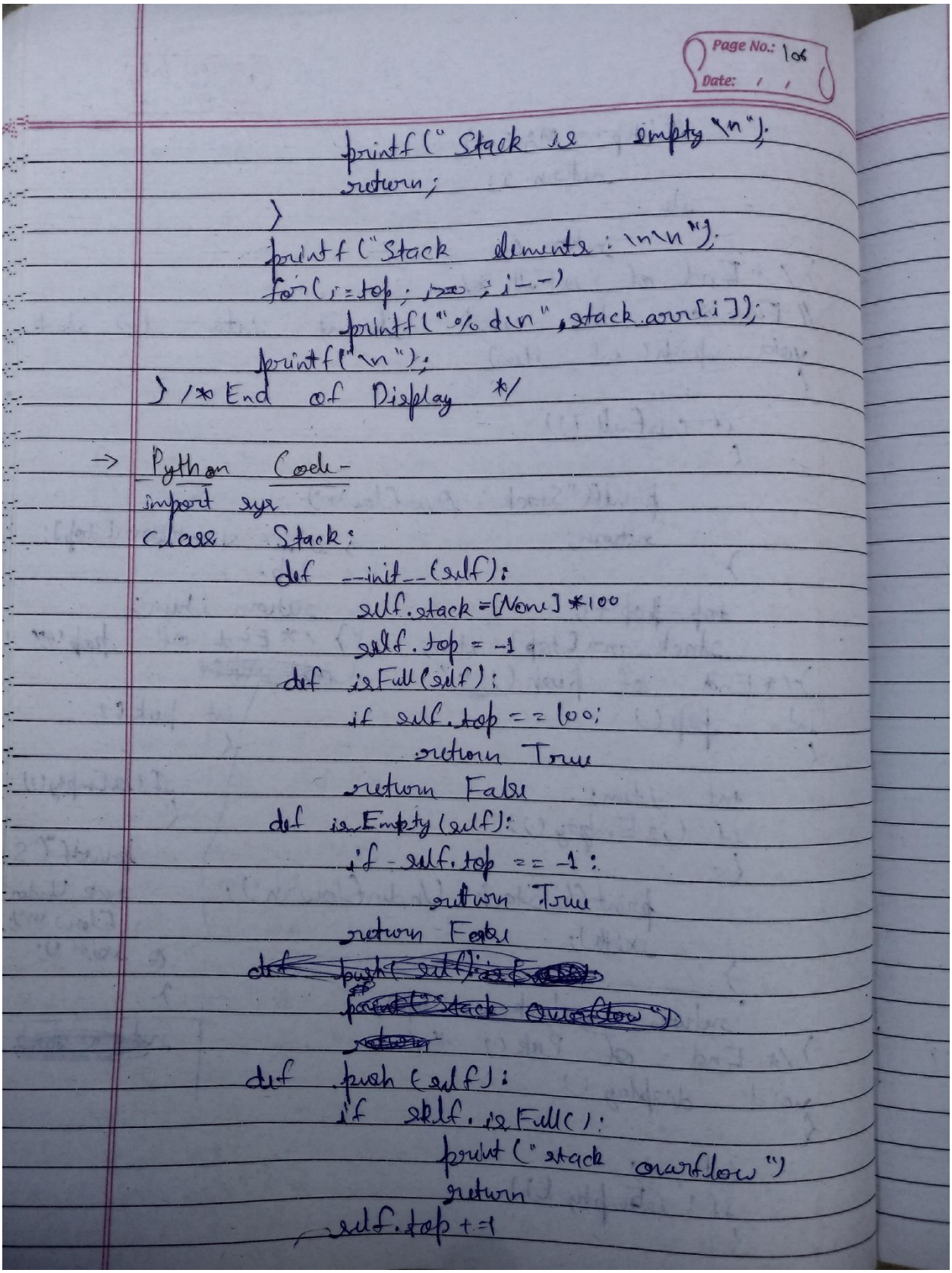
Stack using Array -

→ C Code -

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
int stack_arr[MAX];
int top = -1;
int isEmpty()
{
    if (top == -1)
        return 1;
    else
        return 0;
} /* End of isEmpty */
int isFull()
{
```



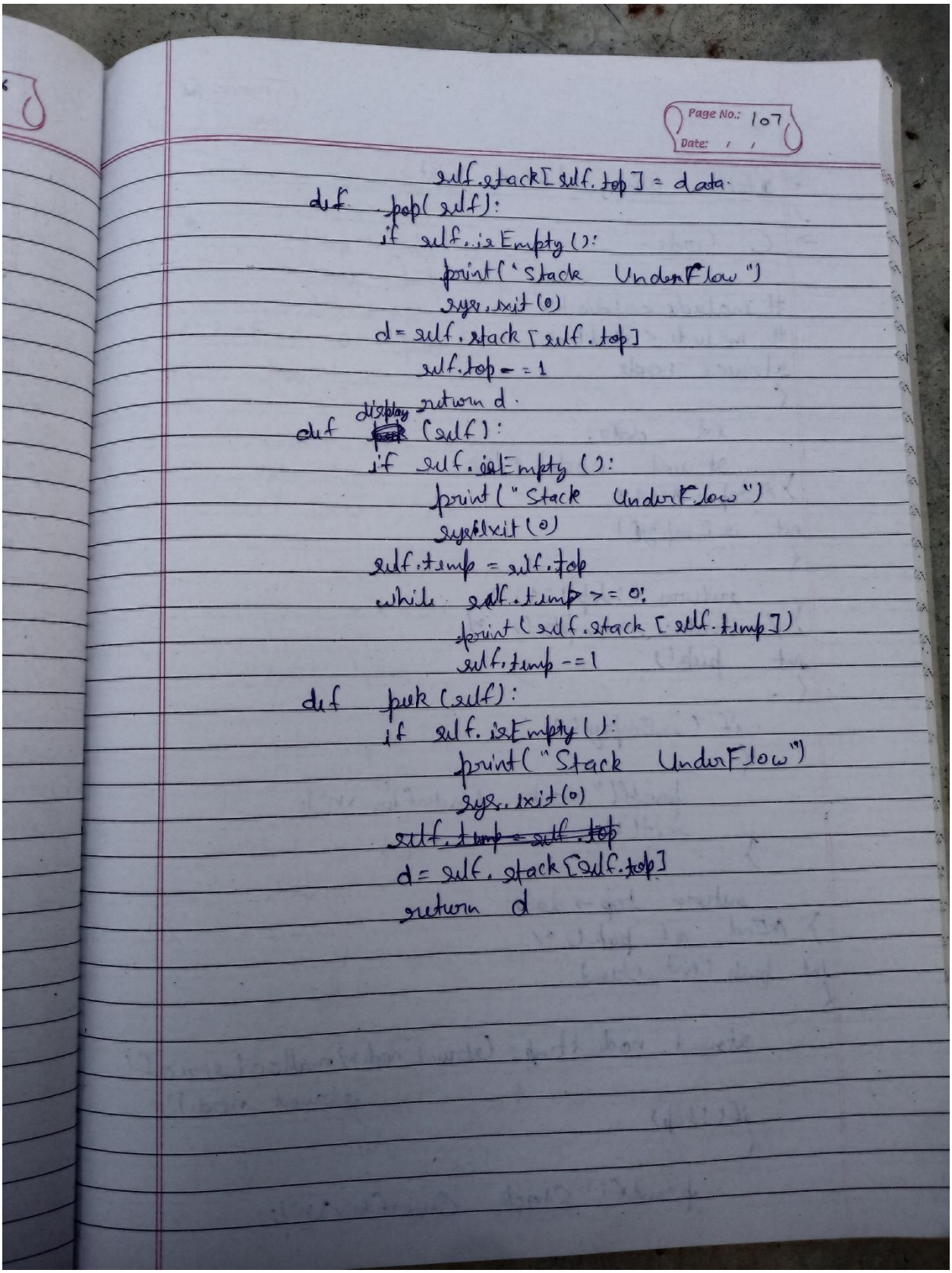
```
Page No.: 105
Date: 3/11/20

    if (top == MAX-1)
        return 1;
    else
        return 0;
}/* End of isFull */
// Function to insert element into the stack
void push(int item)
{
    if (isFull())
    {
        printf("Stack Overflow\n");
        return;
    }
    top = top + 1;
    stack_arr[top] = item;
}/* End of push() */
int pop()
{
    int item;
    if (isEmpty())
    {
        printf("Stack Underflow\n");
        exit(1);
    }
    return stack_arr[top];
}/* End of Pop() */
void display()
{
    int i;
    if (isEmpty())
    {

item = stack_arr[top];
top--;
return item;
} /* End of pop() */

int peek()
{
if (isEmpty())
{
printf("Stack Underflow\n");
exit(1);
}
return stack
```



```
            printf(" Stack is empty \n");
            return;
        }
        printf("Stack elements : \n\n");
        for(i=top; i>=0; i--)
            printf("%d\n", stack.arr[i]);
        printf("\n");
} /* End of Display */
```

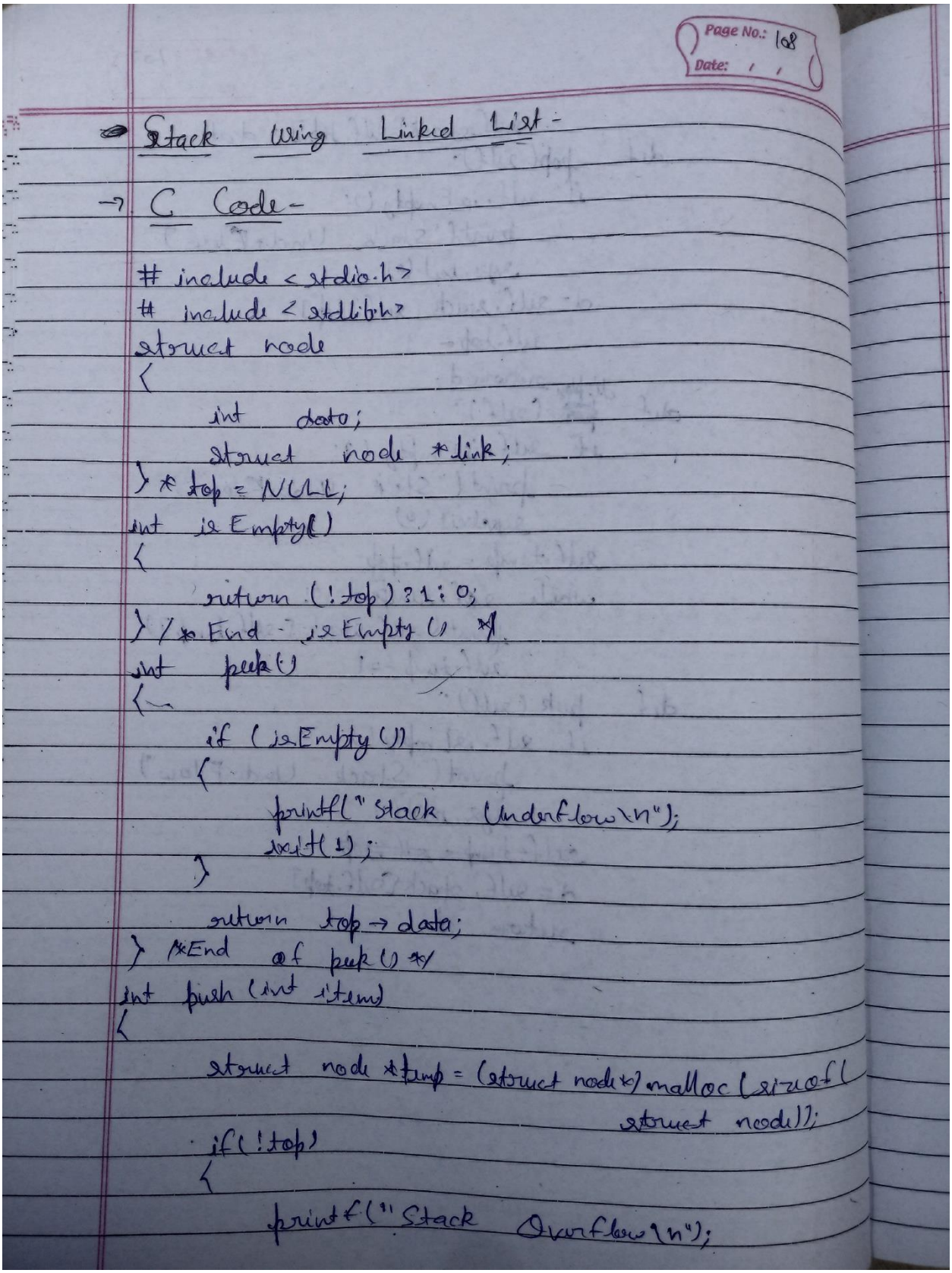
→ Python Code-

```
import sys
class Stack:
    def __init__(self):
        self.stack = [None] * 100
        self.top = -1
    def isFull(self):
        if self.top == 100:
            return True
        return False
    def isEmpty(self):
        if self.top == -1:
            return True
        return False
    def push(self):
        if self.isFull():
            print("stack overflow")
            return
        self.top += 1
```



Page No.: 107
Date: / /

```
            self.stack[self.top] = data
    def pop(self):
        if self.isEmpty():
            print("Stack UnderFlow")
            sys.exit(0)
        d = self.stack[self.top]
        self.top -= 1
        return d
    def display(self):
        if self.isEmpty():
            print("Stack UnderFlow")
            sys.exit(0)
        self.temp = self.top
        while self.temp >= 0:
            print(self.stack[self.temp])
            self.temp -= 1
    def peek(self):
        if self.isEmpty():
            print("Stack UnderFlow")
            sys.exit(0)
        d = self.stack[self.top]
        return d
```

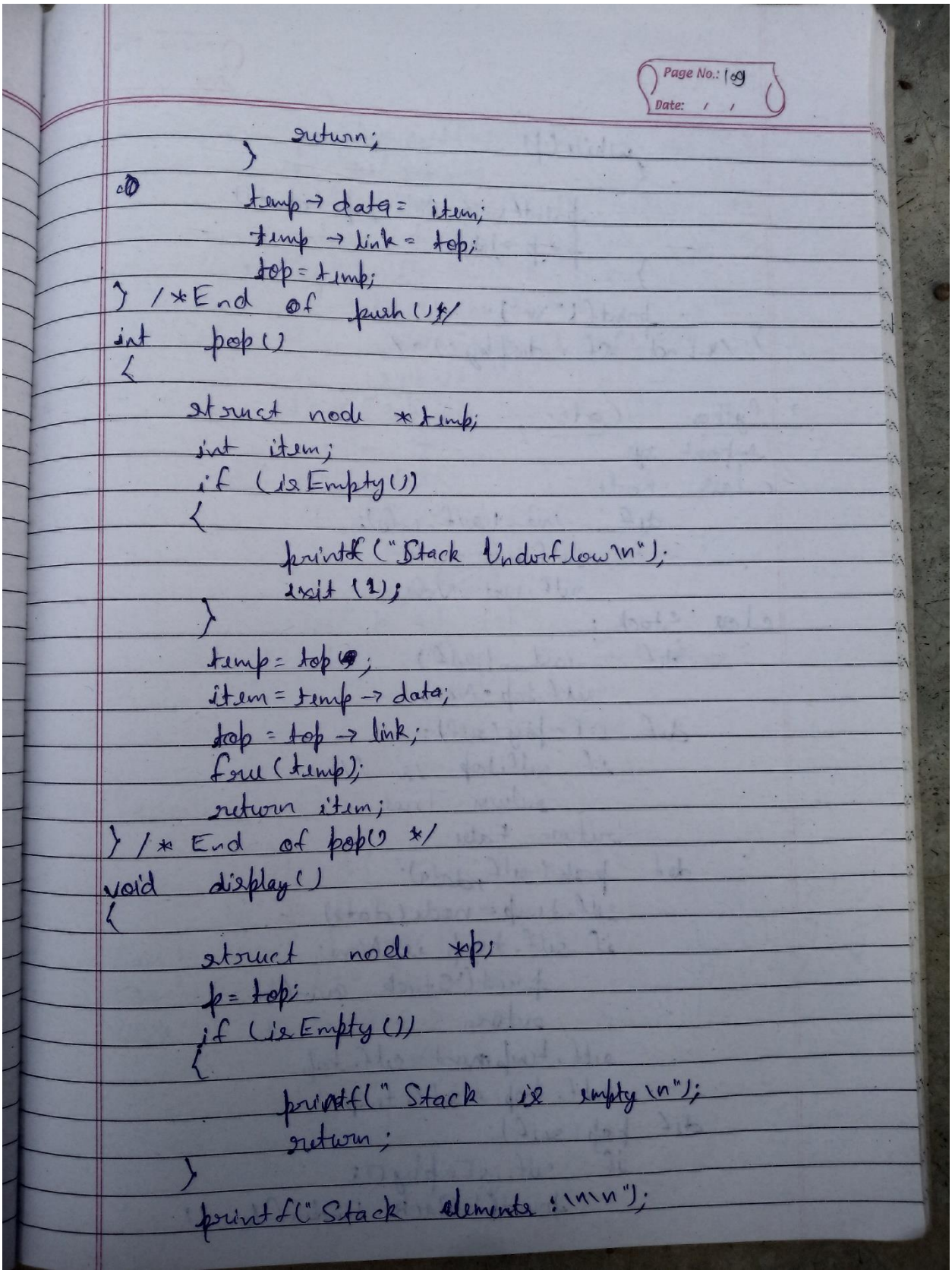


Page No.: 108
Date: / /

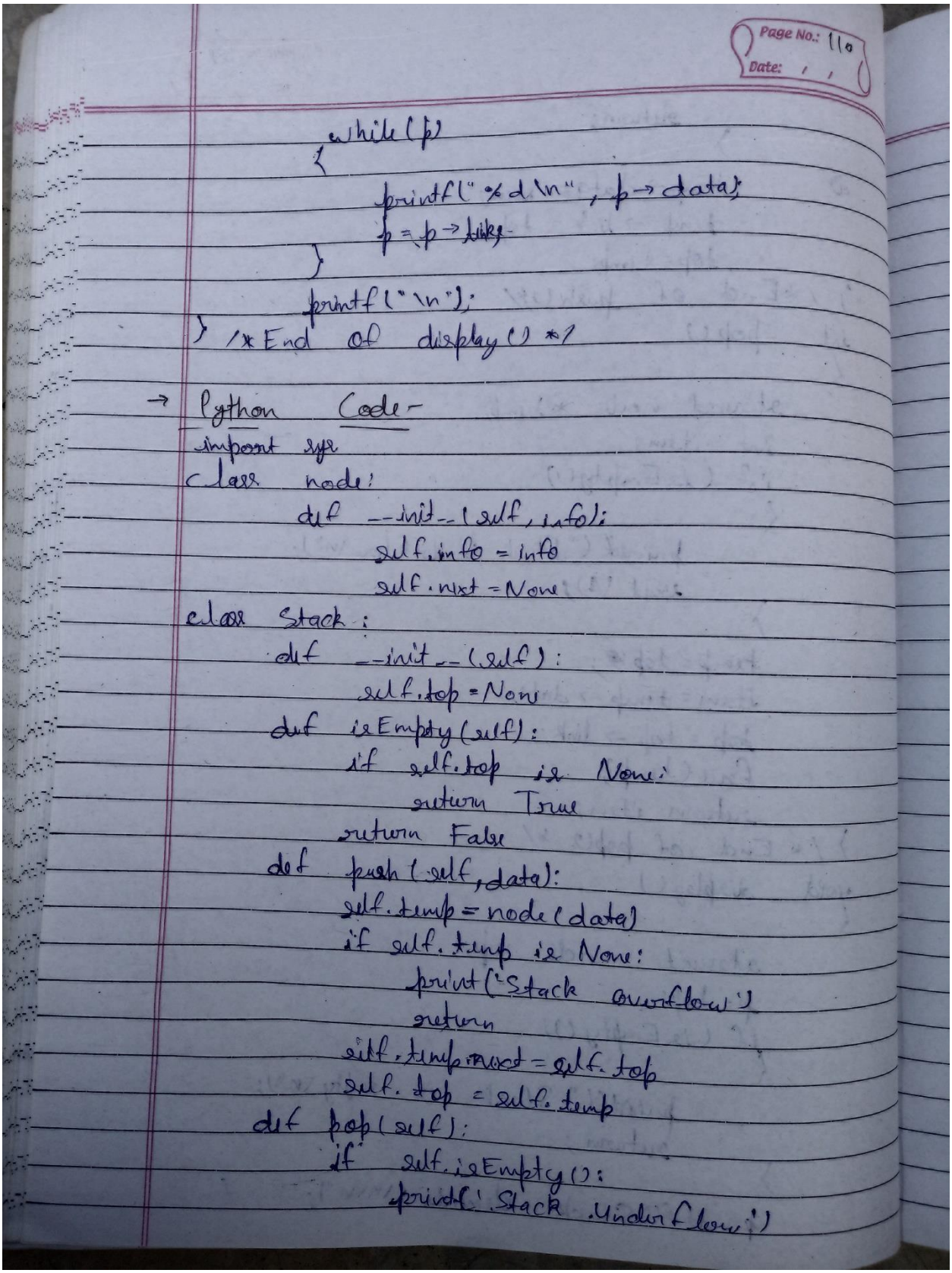
Stack using Linked List -

→ C Code -

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *link;
} *top = NULL;
int isEmpty()
{
    return (!top) ? 1 : 0;
} /* End of isEmpty() */
int peek()
{
    if (isEmpty())
    {
        printf("Stack Underflow\n");
        exit(1);
    }
    return top->data;
} /* End of peek() */
int push(int item)
{
    struct node *temp = (struct node*) malloc(sizeof(
                                struct node));
    if(!top)
    {
        printf("Stack Overflow\n");
```



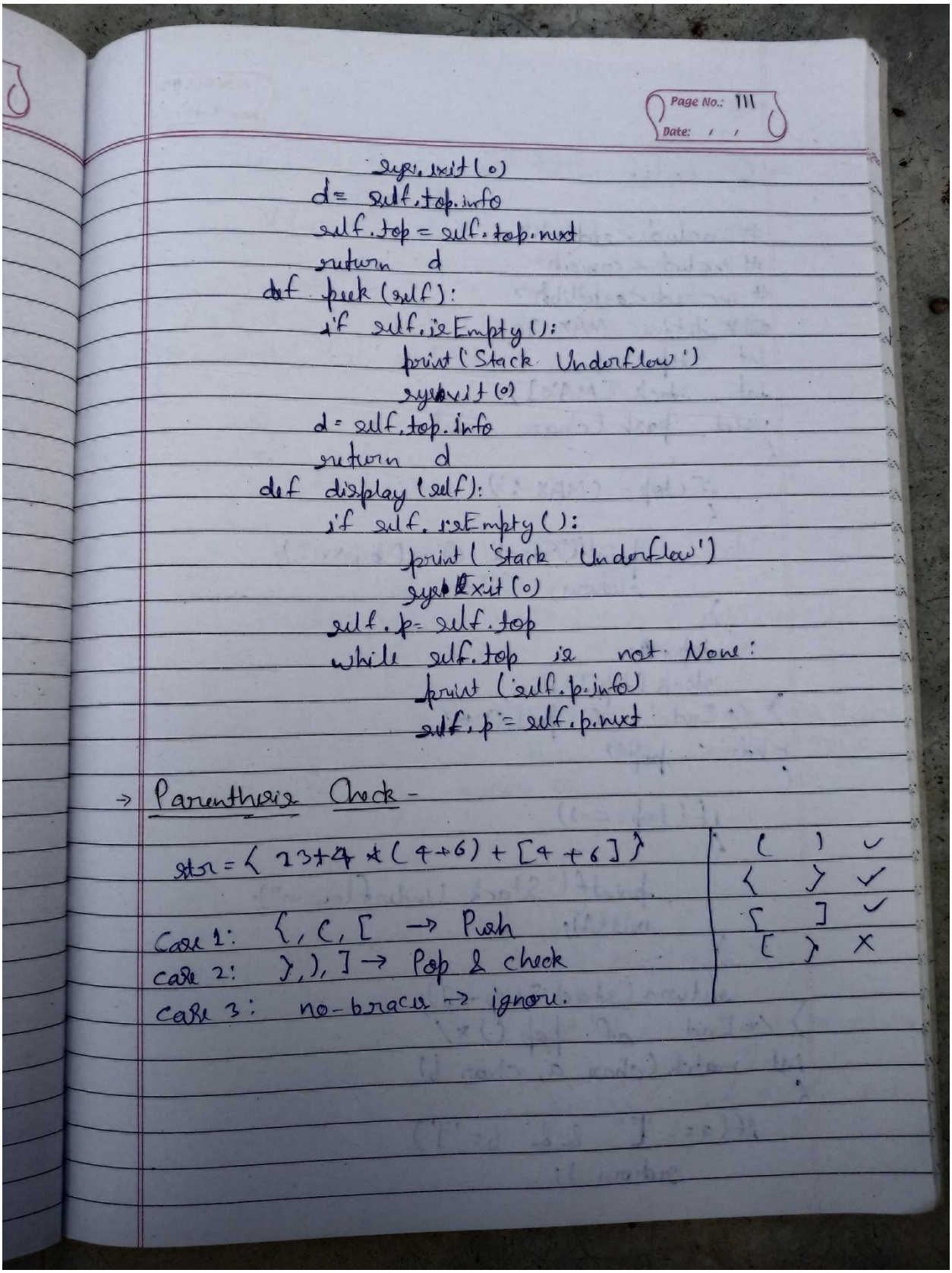
```
            return;
        }
        temp -> data = item;
        temp -> link = top;
        top = temp;
} /*End of push()*/
int pop()
{
    struct node *temp;
    int item;
    if (isEmpty())
    {
        printf("Stack Underflow\n");
        exit(1);
    }
    temp = top;
    item = temp -> data;
    top = top -> link;
    free(temp);
    return item;
} /* End of pop() */
void display()
{
    struct node *p;
    p = top;
    if (isEmpty())
    {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack elements :\n\n");
```



```
        while(p)
        {
            printf("%d\n", p->data);
            p = p->link;
        }
        printf("\n");
} /* End of display() */
```

→ Python Code -

```
import sys
class node:
    def __init__(self, info):
        self.info = info
        self.next = None
class Stack:
    def __init__(self):
        self.top = None
    def isEmpty(self):
        if self.top is None:
            return True
        return False
    def push(self, data):
        self.temp = node(data)
        if self.temp is None:
            print('Stack overflow')
            return
        self.temp.next = self.top
        self.top = self.temp
    def pop(self):
        if self.isEmpty():
            print('Stack Underflow')
```



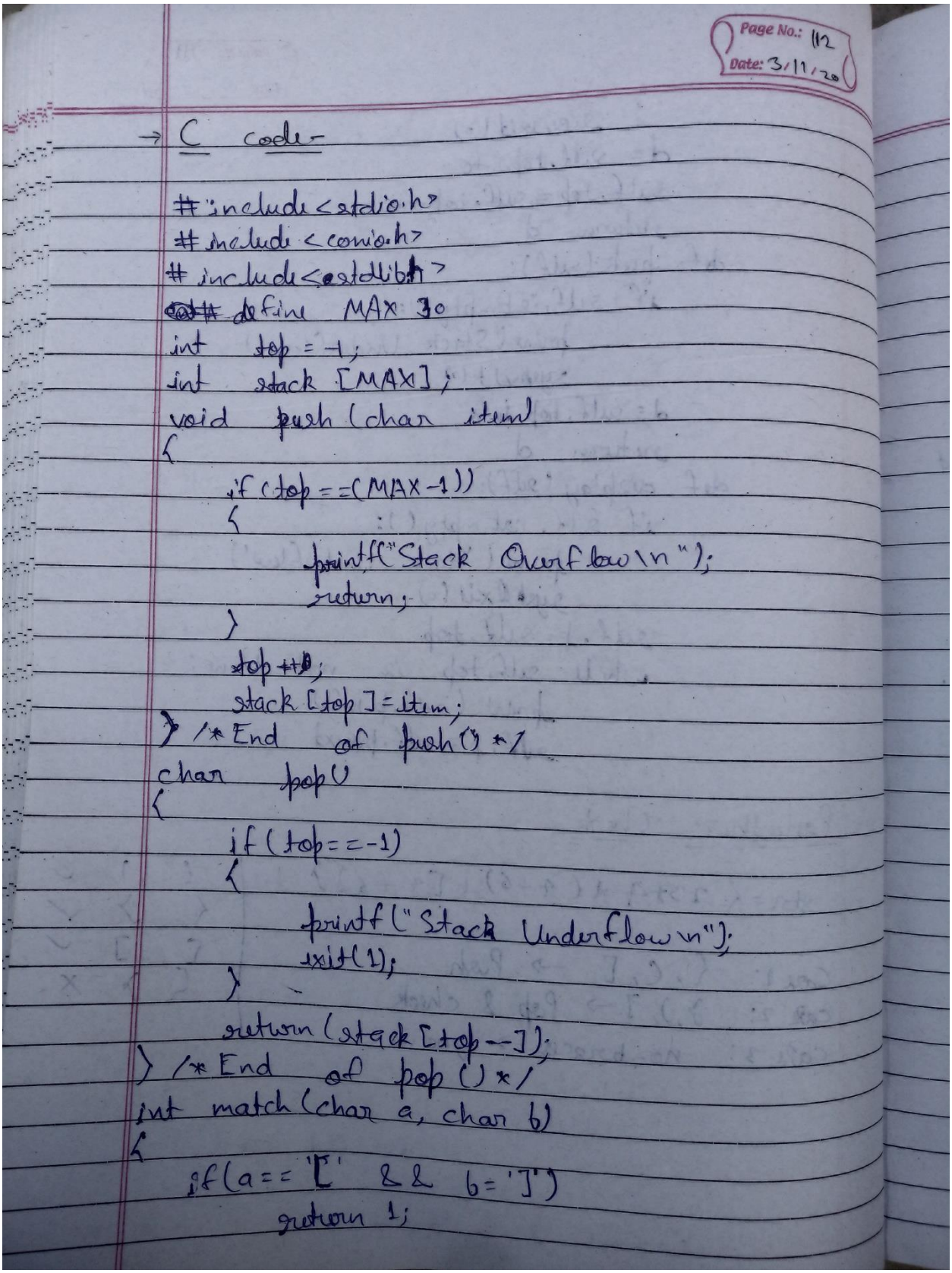
```
            sys.exit(0)
        d = self.top.info
        self.top = self.top.next
        return d
    def peek(self):
        if self.isEmpty():
            print('Stack Underflow')
            sys.exit(0)
        d = self.top.info
        return d
    def display(self):
        if self.isEmpty():
            print('Stack Underflow')
            sys.exit(0)
        self.p = self.top
        while self.top is not None:
            print(self.p.info)
            self.p = self.p.next
```

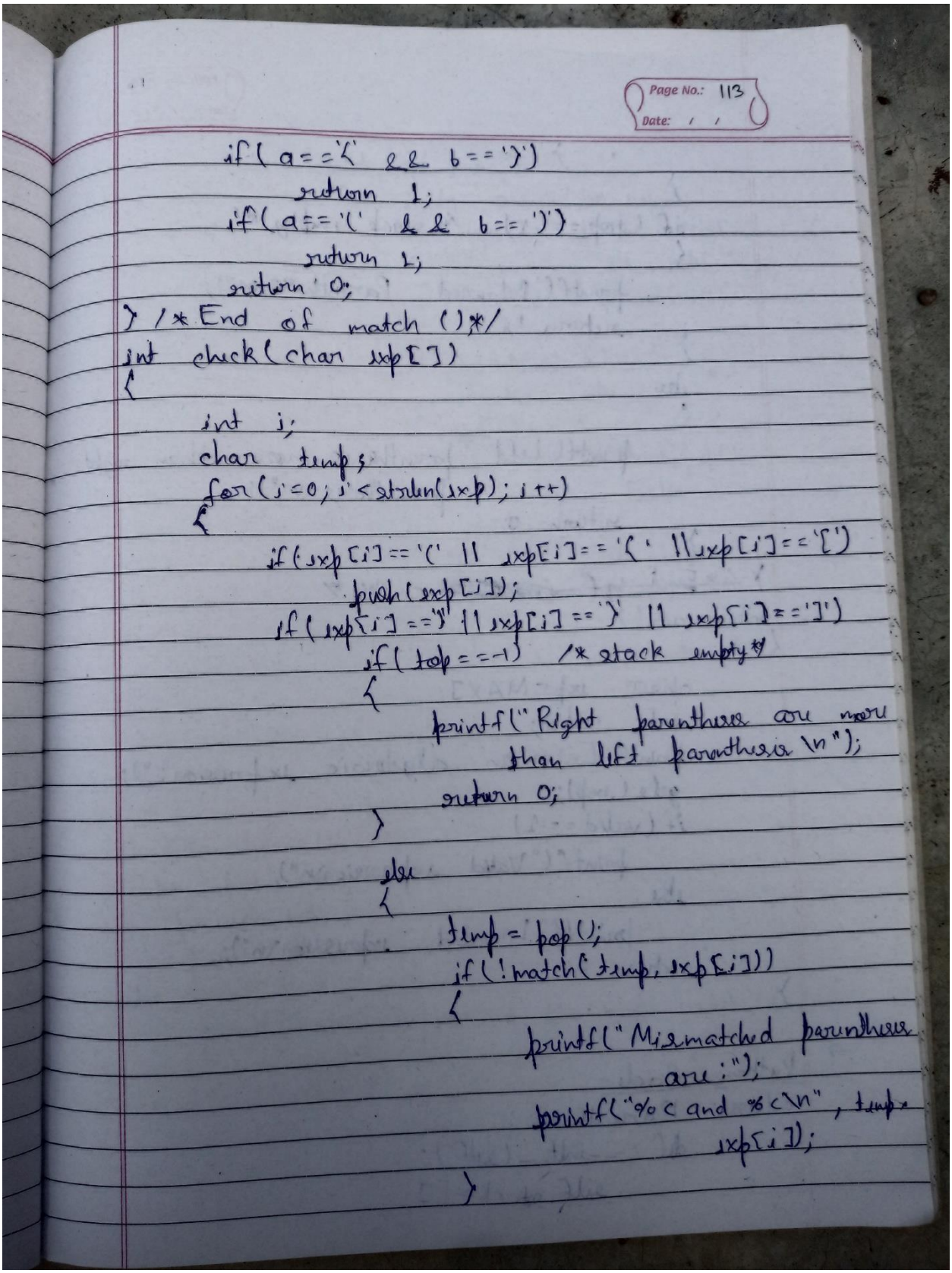
→ Parenthesis Check -

str = { 23+4 * (4+6) + [4 + 6] }

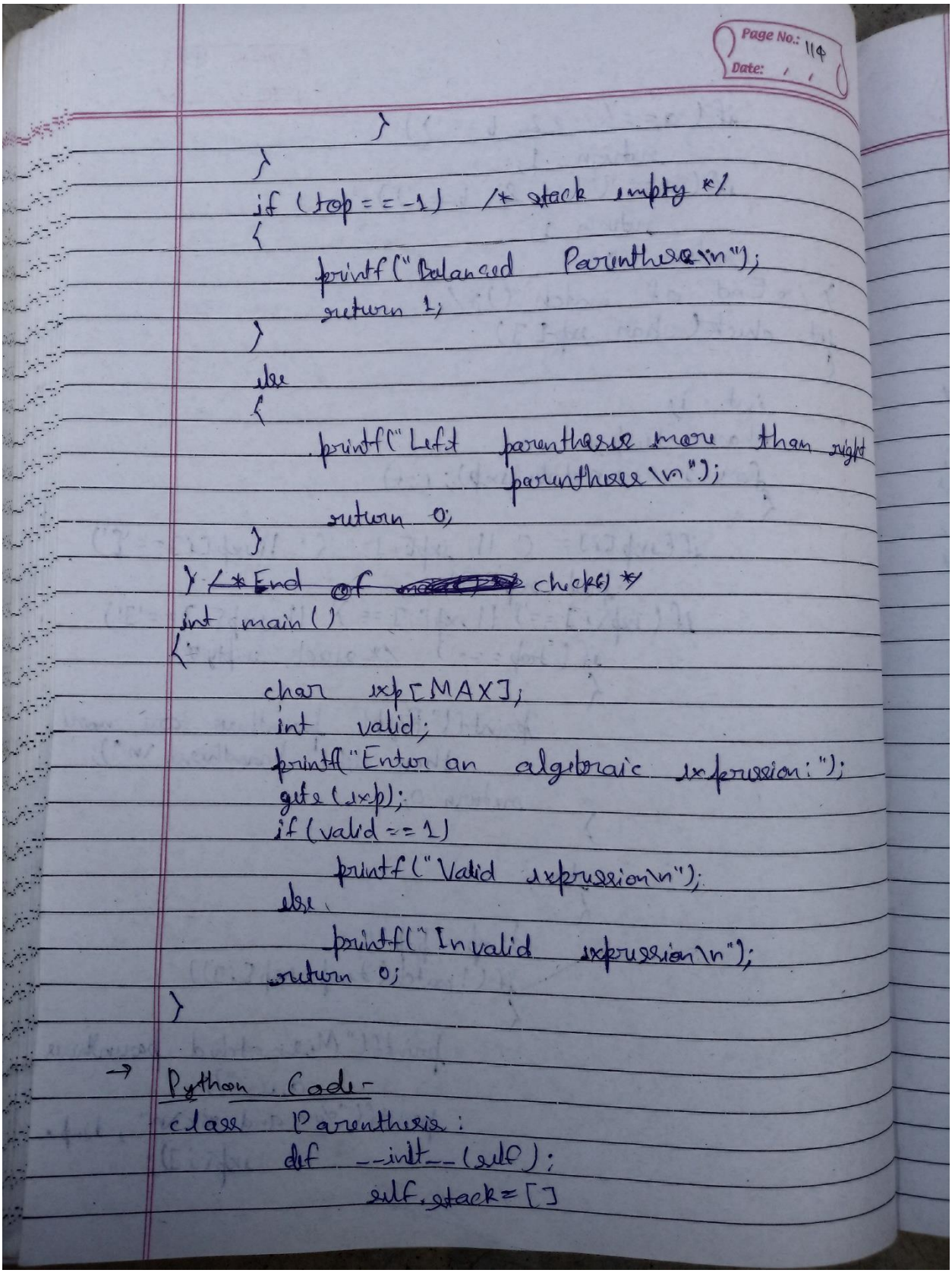
Case 1: {, (, [→ Push
Case 2: },),] → Pop & check
Case 3: no-braces → ignore.

(	)	✓
{	}	✓
[	]	✓
[	)	✗


Page No.: 112
Date: 3/11/20
→ C code-
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 30
int top = -1;
int stack[MAX];
void push(char item)
{
if(top==(MAX-1))
{
printf("Stack Overflow\n");
return;
}
top++;
stack[top]=item;
} /* End of push() */
char pop()
{
if(top==-1)
{
printf("Stack Underflow\n");
exit(1);
}
return(stack[top--]);
} /* End of pop() */
int match(char a, char b)
{
if(a=='[' && b==']')
return 1;



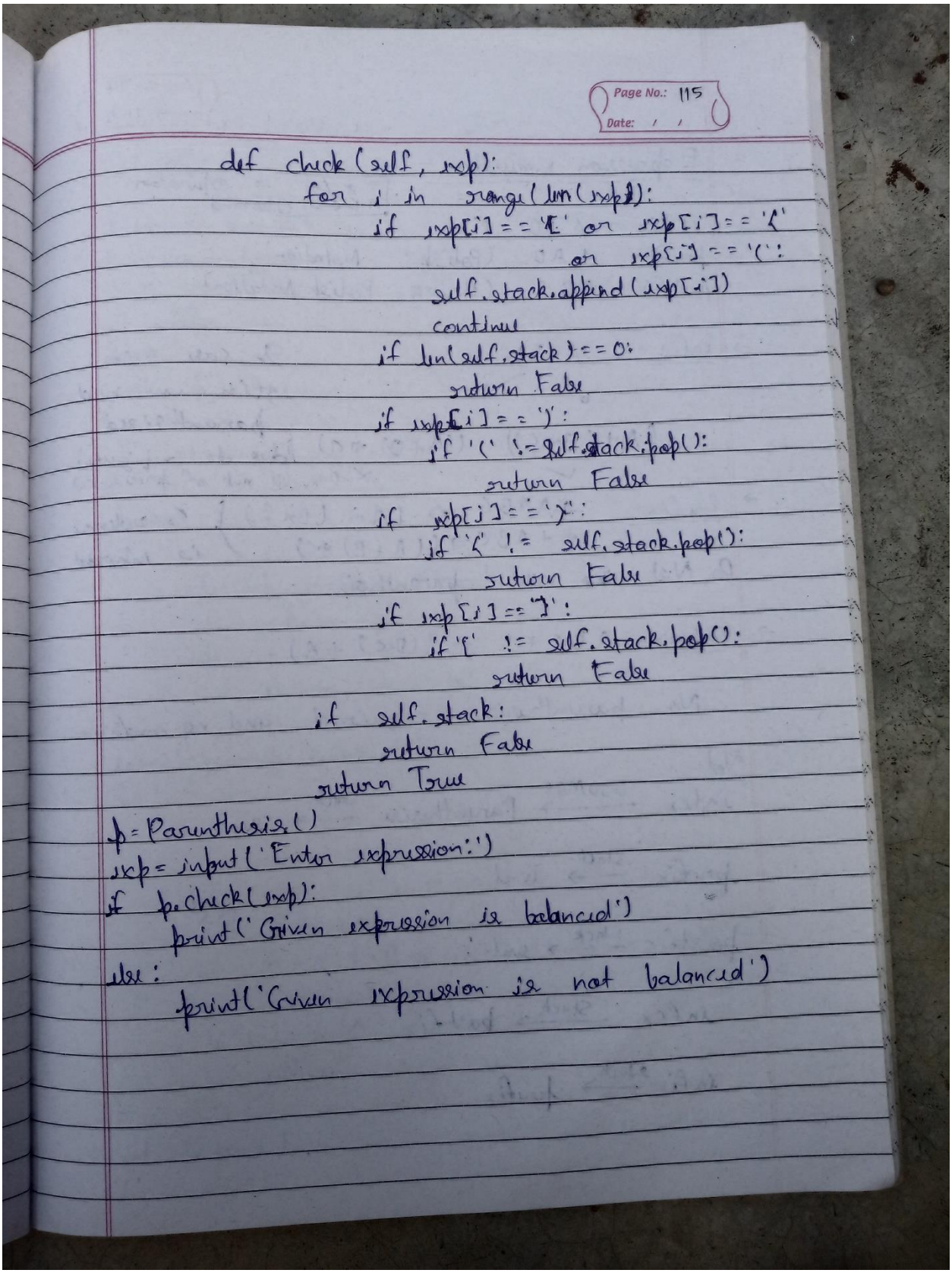
```
Page No.: 113
Date: / /

if( a=='{' && b=='}')
    return 1;
if( a=='(' && b==')')
    return 1;
return 0;
} /* End of match ()*/
int check(char sxp[])
{
    int i;
    char temp;
    for(i=0; i<strlen(sxp); i++)
    {
        if(sxp[i]=='(' || sxp[i]=='{' ||sxp[i]=='[')
            push(sxp[i]);
        if(sxp[i]==')' ||sxp[i]=='}' ||sxp[i]==']')
            if(top==-1)  /* stack empty*/
            {
                printf("Right parentheses are more
                    than left parenthesis\n");
                return 0;
            }
            else
            {
                temp=pop();
                if(!match(temp,sxp[i]))
                {
                    printf("Mismatched parenthesis
                        are:");
                    printf("%c and %c\n", temp,
                        sxp[i]);
                }
```



```
        }
    }
    if (top == -1)  /* stack empty */
    {
        printf("Balanced Parenthesis\n");
        return 1;
    }
    else
    {
        printf("Left parenthesis more than right
                parenthesis\n");
        return 0;
    }
} /* End of check() */
int main()
{
    char exp[MAX];
    int valid;
    printf("Enter an algebraic expression:");
    gets(exp);
    if (valid == 1)
        printf("Valid expression\n");
    else
        printf("Invalid expression\n");
    return 0;
}
```

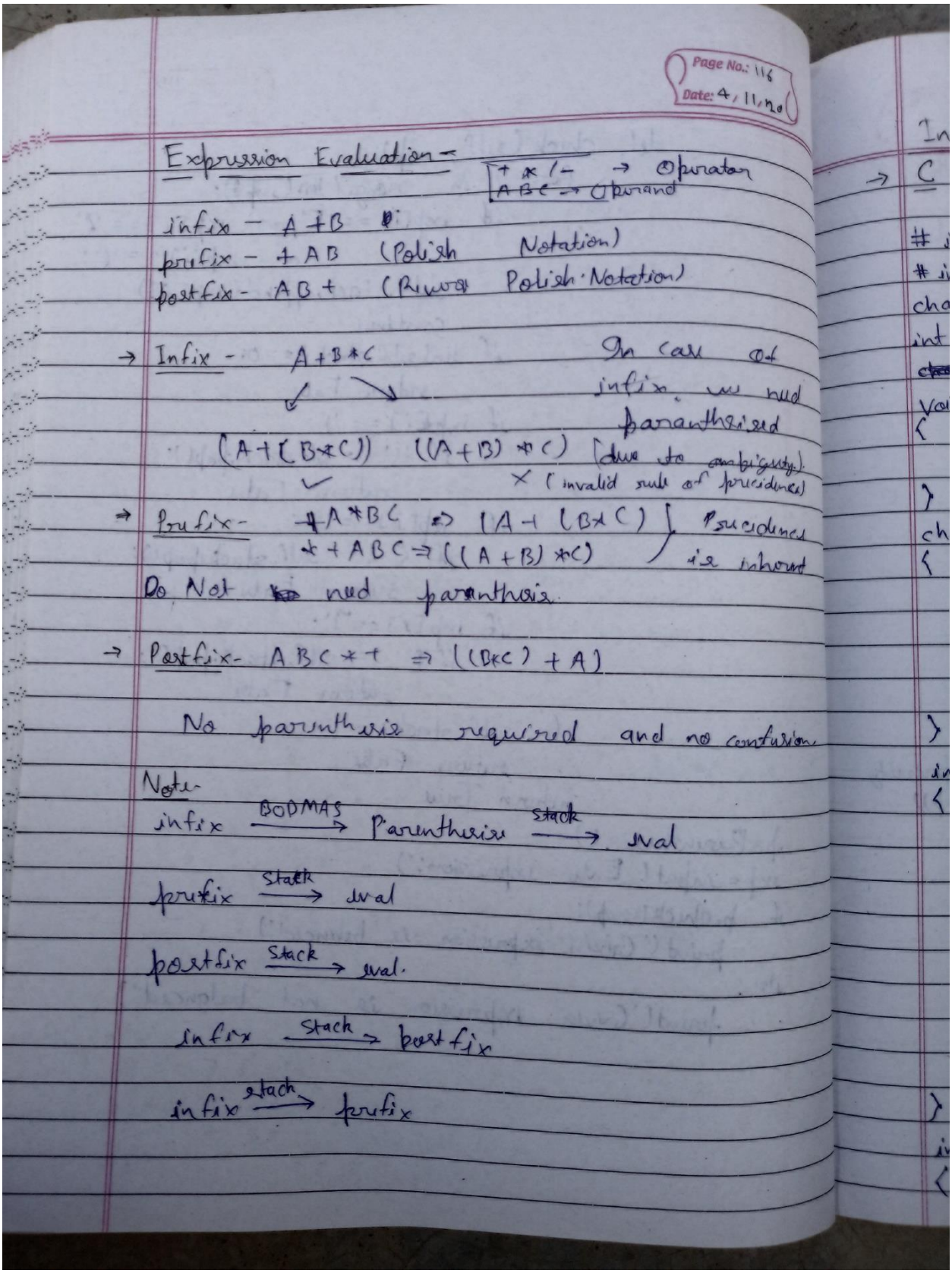
→ Python Code -

```
class Parenthesis:
    def __init__(self):
        self.stack = []
```



Page No.: 115
Date: / /

```
def check(self, exp):
    for i in range(len(exp)):
        if exp[i] == '[' or exp[i] == '{'
                or exp[i] == '(':
            self.stack.append(exp[i])
            continue
        if len(self.stack) == 0:
            return False
        if exp[i] == ')':
            if '(' != self.stack.pop():
                return False
        if exp[i] == '}':
            if '{' != self.stack.pop():
                return False
        if exp[i] == ']':
            if '[' != self.stack.pop():
                return False
    if self.stack:
        return False
    return True
p = Parenthesis()
exp = input('Enter expression:')
if p.check(exp):
    print('Given expression is balanced')
else:
    print('Given expression is not balanced')
```



Page No.: 116
Date: 4/11/20

Expression Evaluation -

+ * / - → Operator
A B C → Operand

infix - A + B
prefix - + A B (Polish Notation)
postfix - A B + (Reverse Polish Notation)

→ Infix - A + B * C

(A + (B * C)) ✓ ((A + B) * C) ✗ (invalid due to ambiguity / invalid rule of precedence)

In case of infix, we need parenthesized

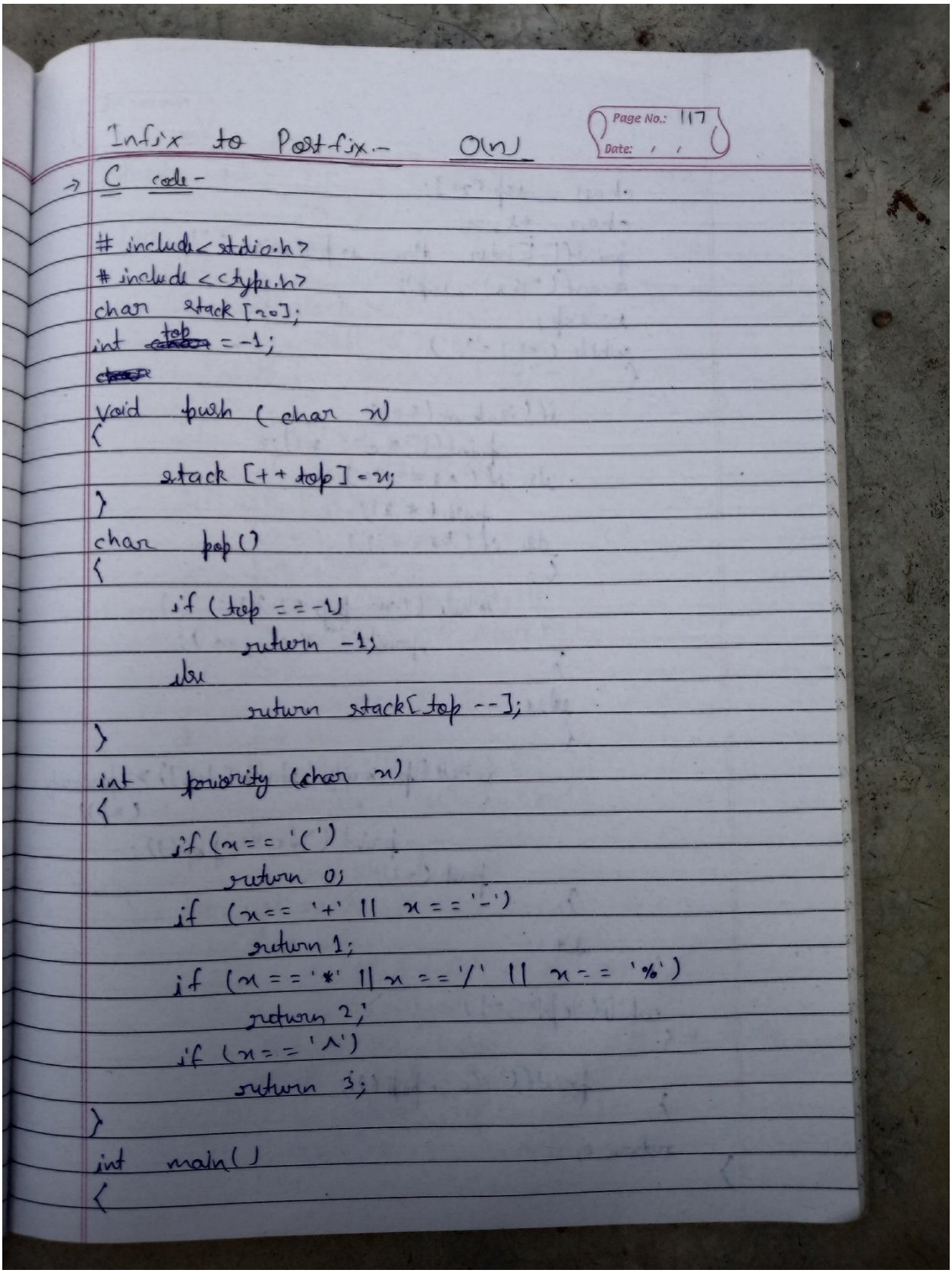
→ Prefix - + A * B C ⇒ (A + (B * C)) precedence
* + A B C ⇒ ((A + B) * C) i.e. inward

Do Not need parenthesis

→ Postfix - A B C * + ⇒ ((B * C) + A)

No parenthesis required and no confusion.

Note -
infix —BODMAS→ Parenthesis —stack→ eval
prefix —stack→ eval
postfix —stack→ eval.
infix —stack→ postfix
infix —stack→ prefix

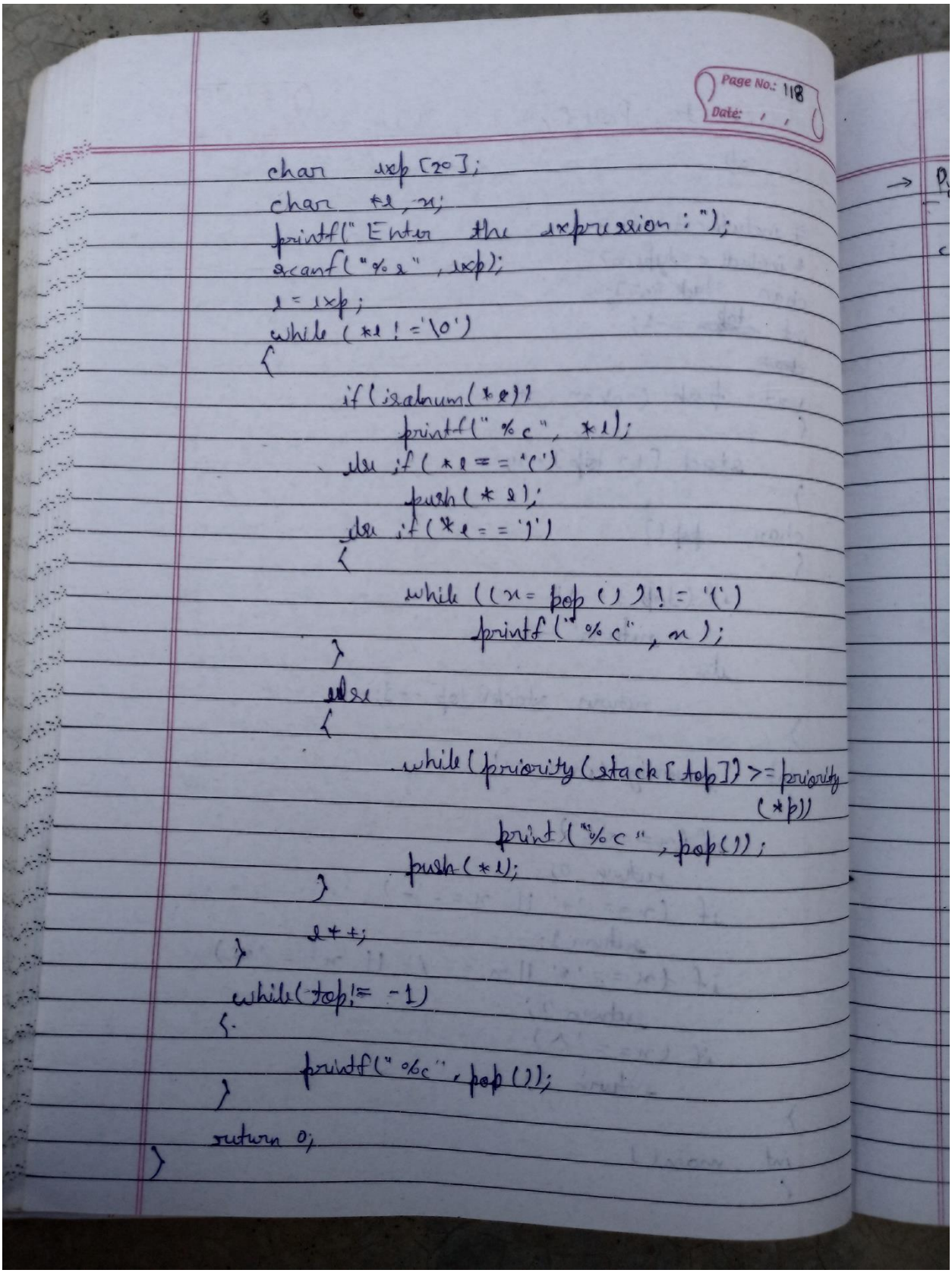


Infix to Postfix - O(n)

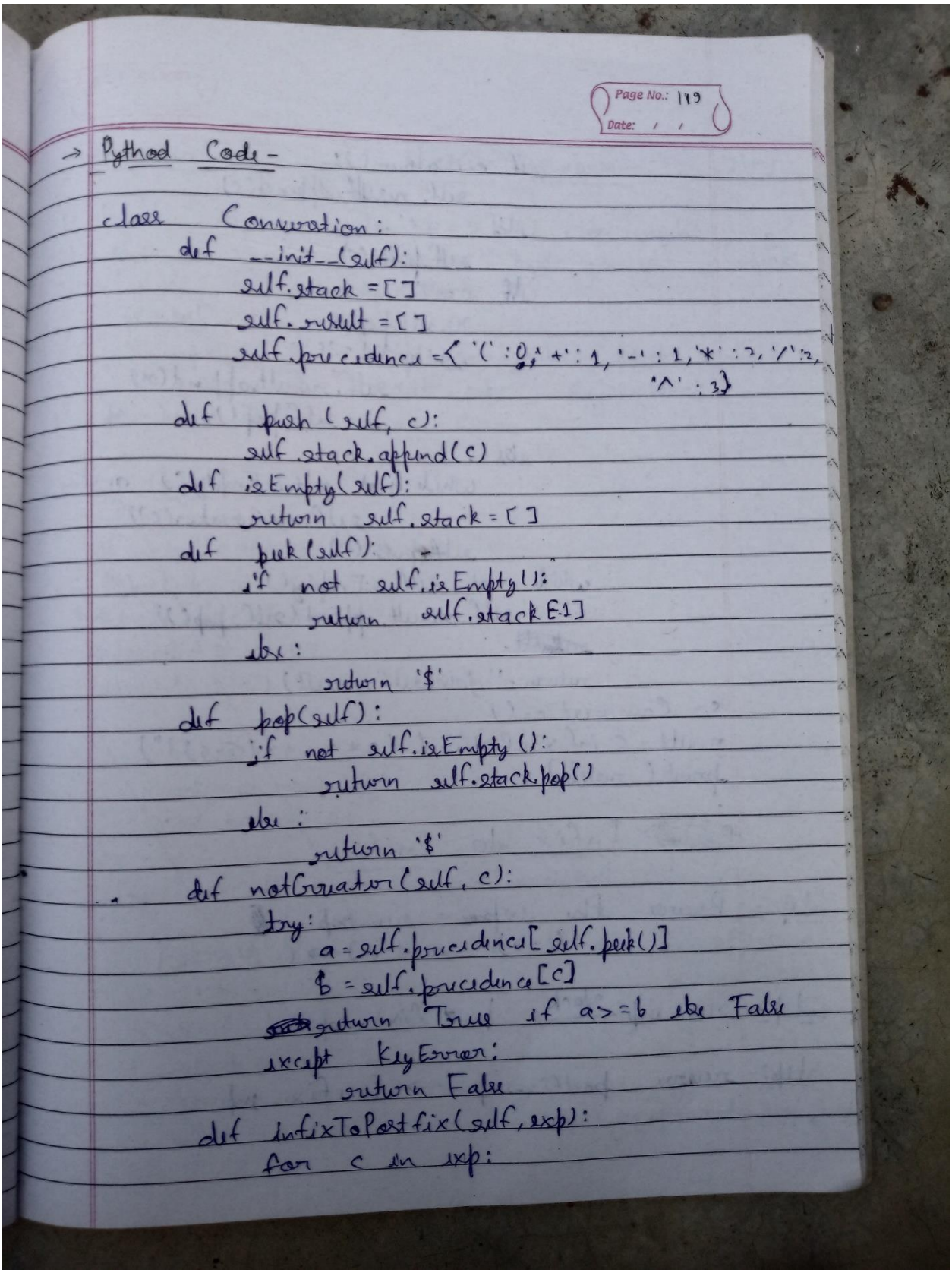
Page No.: 117
Date: / /

→ C code -

```
#include<stdio.h>
#include<ctype.h>
char stack[20];
int top = -1;

void push(char x)
{
    stack[++top] = x;
}
char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}
int priority(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/' || x == '%')
        return 2;
    if(x == '^')
        return 3;
}
int main()
{
```



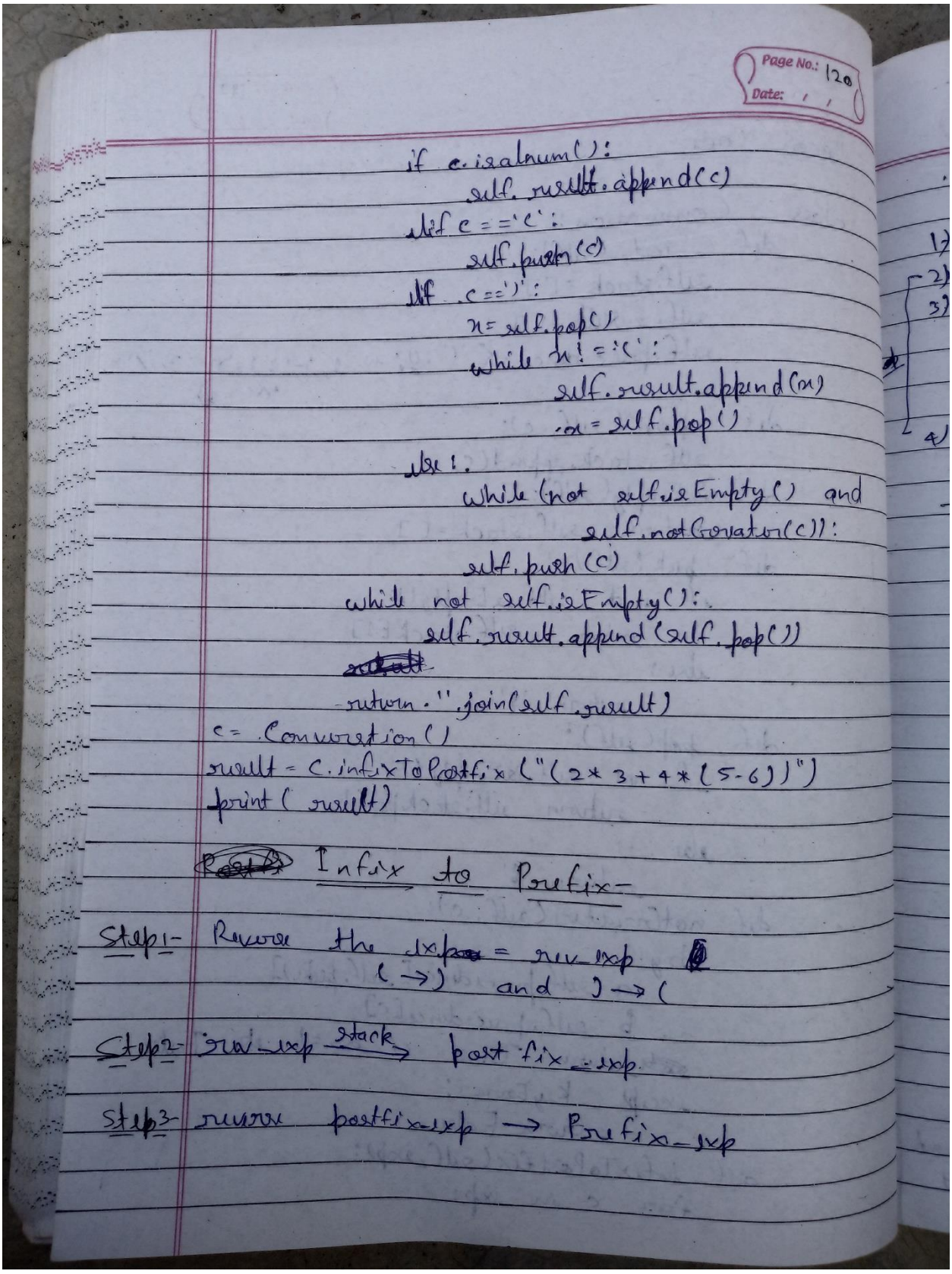
```
char exp[20];
char *e, x;
printf("Enter the expression:");
scanf("%s", exp);
e = exp;
while (*e != '\0')
{
    if(isalnum(*e))
        printf("%c", *e);
    else if(*e == '(')
        push(*e);
    else if(*e == ')')
    {
        while((x = pop()) != '(')
            printf("%c", x);
    }
    else
    {
        while(priority(stack[top]) >= priority(*p))
            print("%c", pop());
        push(*e);
    }
    e++;
}
while(top != -1)
{
    printf("%c", pop());
}
return 0;
}
```



Page No.: 119
Date: / /

→ Python Code -

```
class Conversion:
    def __init__(self):
        self.stack = []
        self.result = []
        self.precedence = {'(':0, '+':1, '-':1, '*':2, '/':2,
                           '^':3}

    def push(self, c):
        self.stack.append(c)
    def isEmpty(self):
        return self.stack == []
    def peek(self):
        if not self.isEmpty():
            return self.stack[-1]
        else:
            return '$'
    def pop(self):
        if not self.isEmpty():
            return self.stack.pop()
        else:
            return '$'
    def notOperator(self, c):
        try:
            a = self.precedence[self.peek()]
            b = self.precedence[c]
            return True if a >= b else False
        except KeyError:
            return False
    def infixToPostfix(self, exp):
        for c in exp:
```



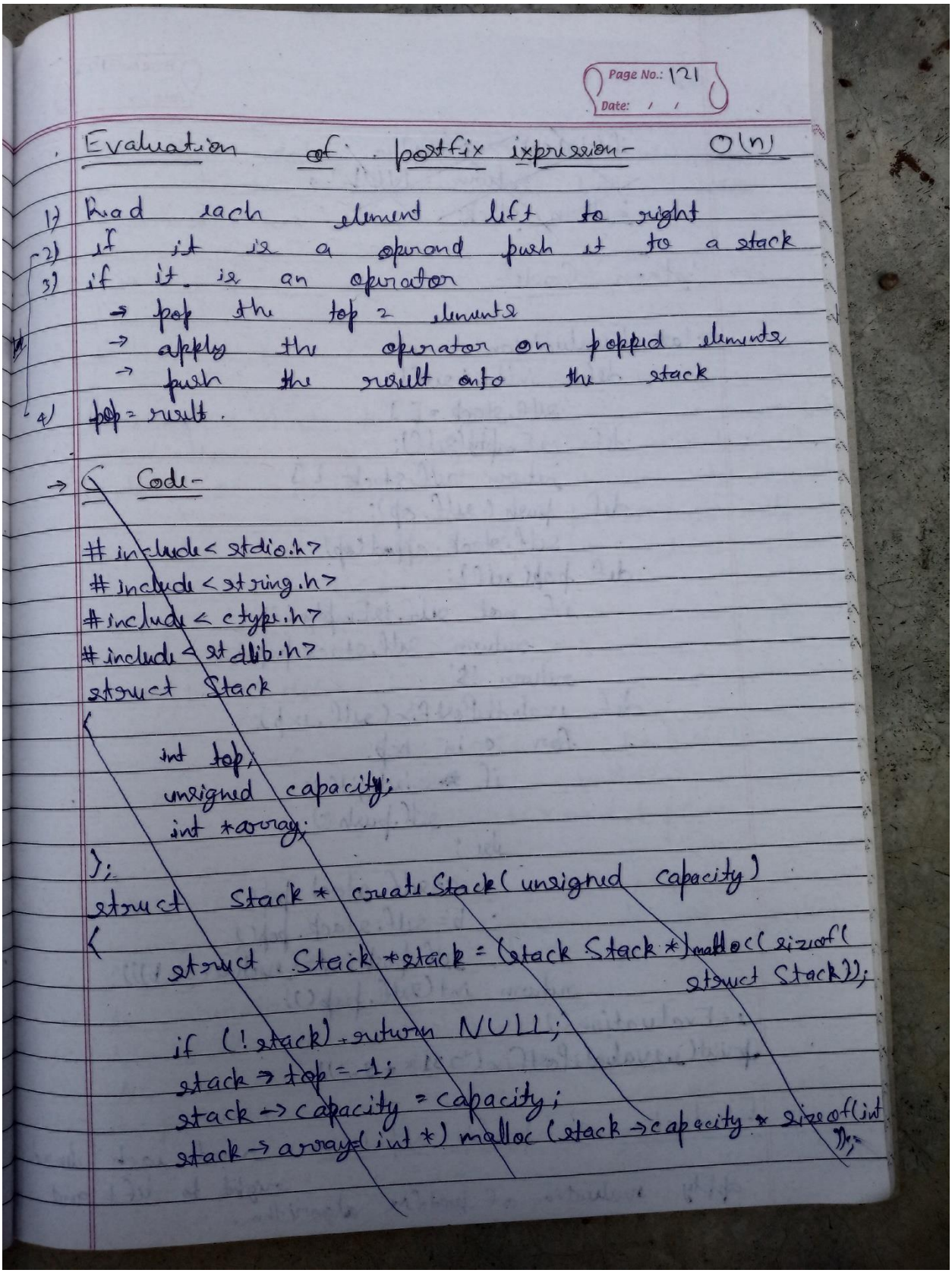
```
if c.isalnum():
    self.result.append(c)
if c == '(':
    self.push(c)
if c == ')':
    x = self.pop()
    while x != '(':
        self.result.append(x)
        x = self.pop()
else:
    while (not self.isEmpty() and
           self.notGreater(c)):
        self.push(c)
while not self.isEmpty():
    self.result.append(self.pop())
return "".join(self.result)
c = Conversion()
result = c.infixToPostfix("(2*3+4*(5-6))")
print(result)
```

Infix to Prefix-

Step 1- Reverse the exp = rev exp
(→) and) → (

Step 2- rev_exp $\xrightarrow{\text{stack}}$ post fix_exp

Step 3- reverse postfix_exp → Prefix_exp



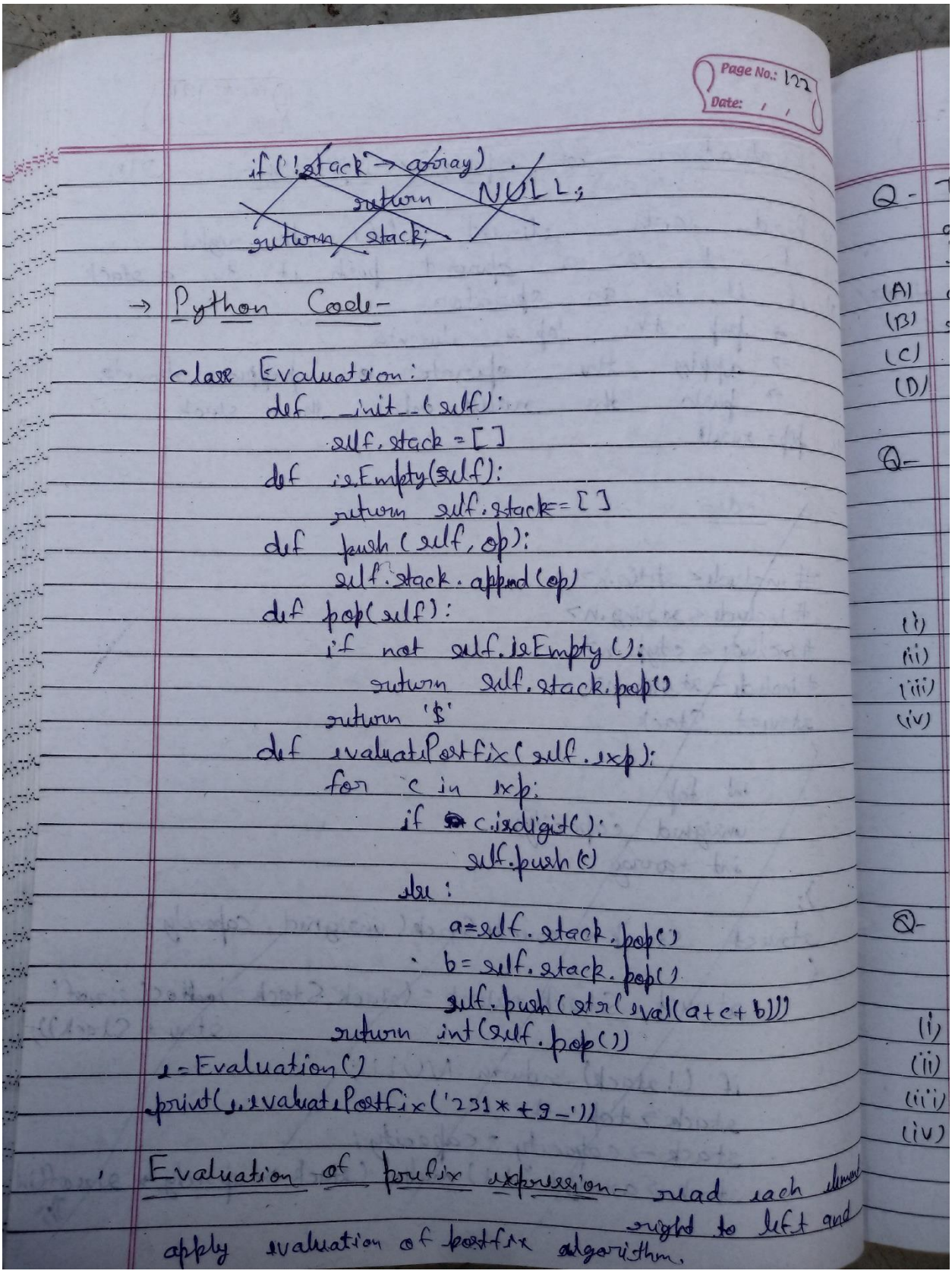
Page No.: 121
Date: / /

Evaluation of postfix expression- O(n)

1) Read each element left to right
2) if it is a operand push it to a stack
3) if it is an operator
   → pop the top 2 elements
   → apply the operator on popped elements
   → push the result onto the stack
4) pop = result.

→ Code-

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
struct Stack
{
    int top;
    unsigned capacity;
    int *array;
};
struct Stack * createStack(unsigned capacity)
{
    struct Stack *stack = (struct Stack *)malloc(sizeof(
                                    struct Stack));
    if (!stack) return NULL;
    stack->top = -1;
    stack->capacity = capacity;
    stack->array(int *) malloc (stack->capacity * sizeof(int
                                    ));
```



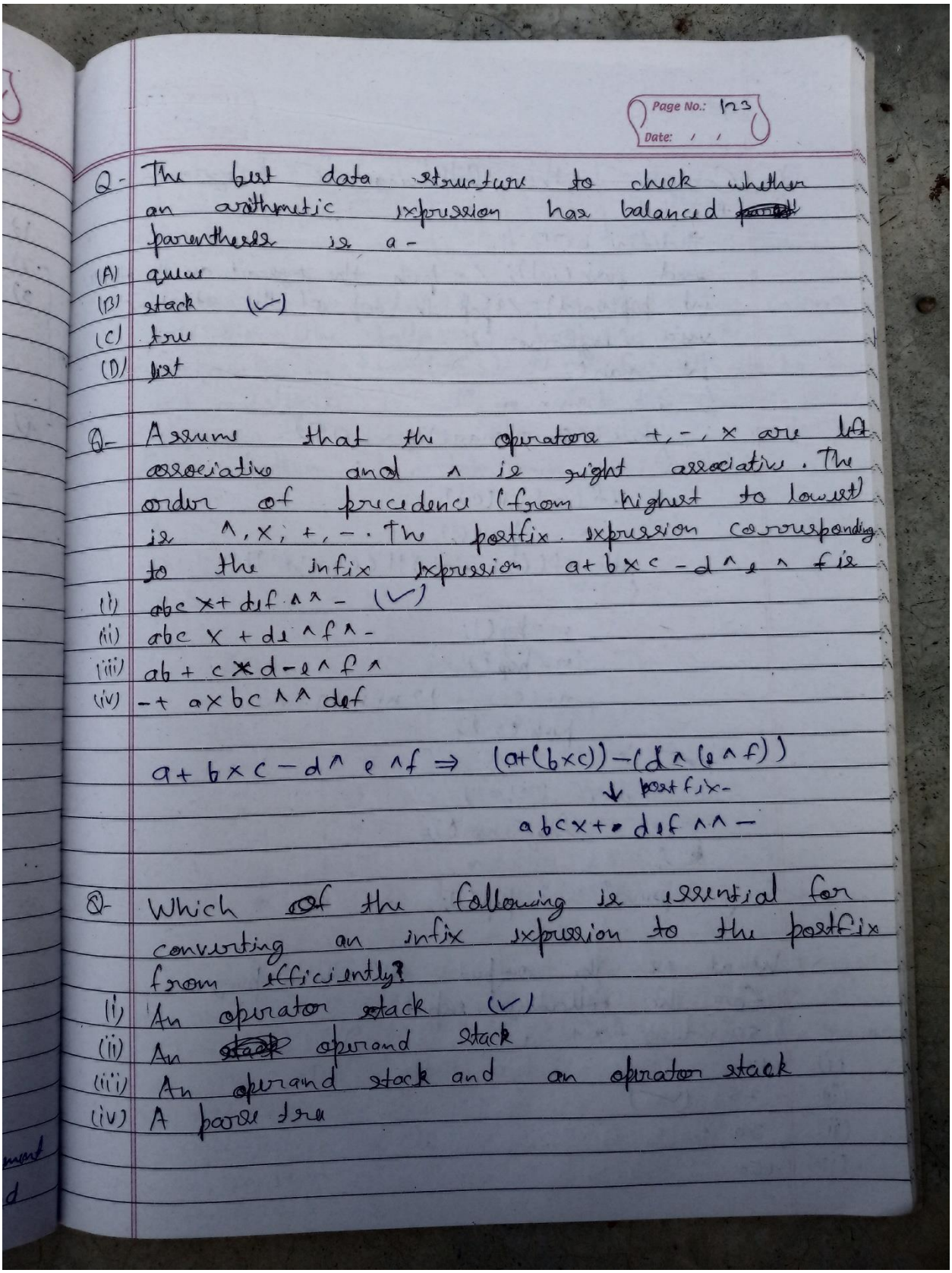
Page No.: 122
Date: / /

~~if(!stack->array)~~
~~return NULL;~~
~~return stack;~~

→ Python Code-

```
class Evaluation:
    def __init__(self):
        self.stack = []
    def isEmpty(self):
        return self.stack == []
    def push(self, op):
        self.stack.append(op)
    def pop(self):
        if not self.isEmpty():
            return self.stack.pop()
        return '$'
    def evaluatePostfix(self, exp):
        for c in exp:
            if c.isdigit():
                self.push(c)
            else:
                a = self.stack.pop()
                b = self.stack.pop()
                self.push(str(eval(a+c+b)))
        return int(self.pop())
e = Evaluation()
print(e.evaluatePostfix('231*+9-'))
```

Evaluation of postfix expression- read each element right to left and apply evaluation of postfix algorithm.



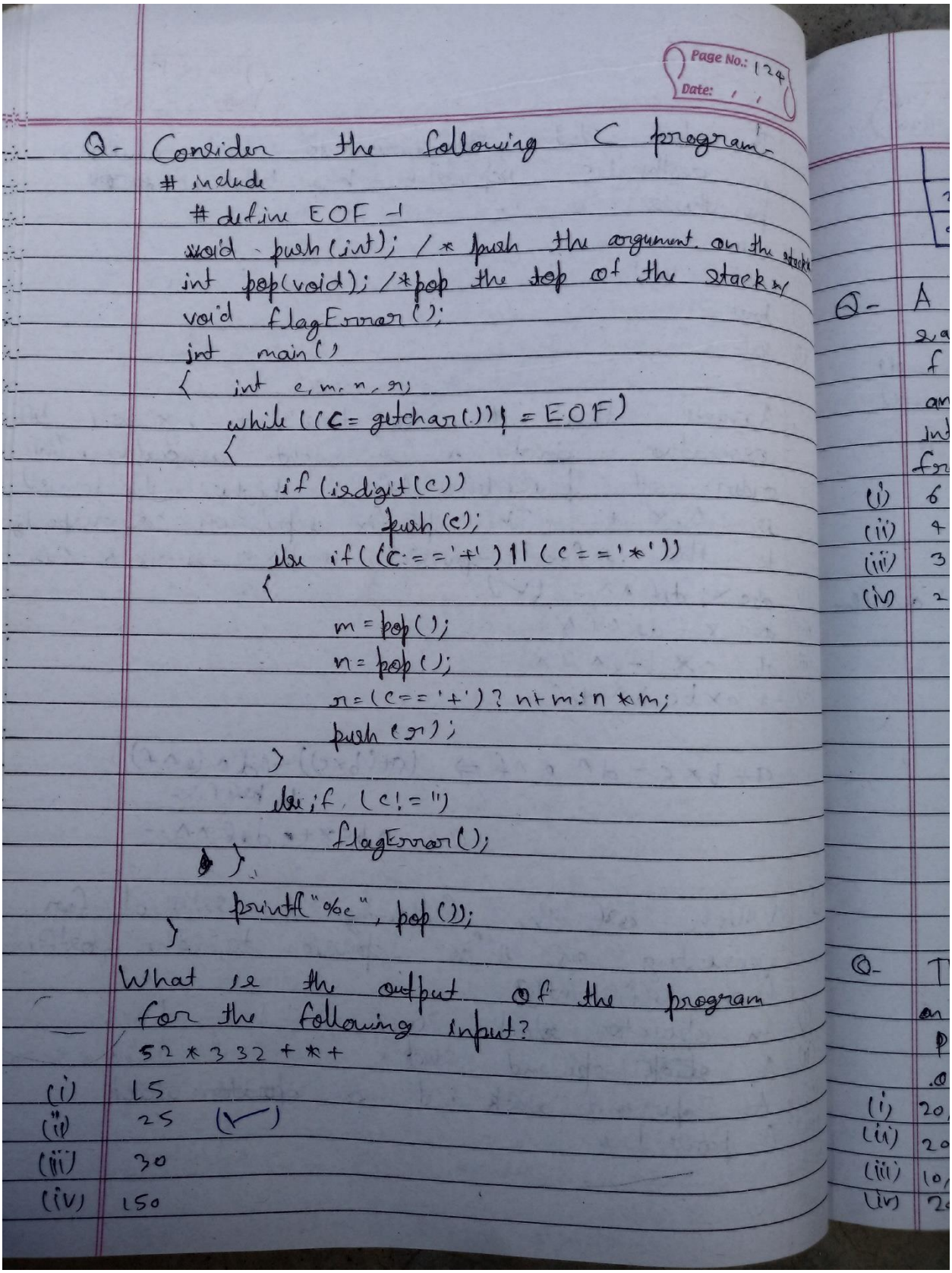
Page No.: 123
Date: / /

Q- The best data structure to check whether an arithmetic expression has balanced parentheses is a -
(A) queue
(B) stack (✓)
(C) tree
(D) list

Q- Assume that the operators +, -, × are left associative and ^ is right associative. The order of precedence (from highest to lowest) is ^, ×, +, -. The postfix expression corresponding to the infix expression a+b×c-d^e^f is
(i) abc×+def^^- (✓)
(ii) abc×+de^f^-
(iii) ab+c*d-e^f^
(iv) -+a×bc^^def

a+b×c-d^e^f ⇒ (a+(b×c))-(d^(e^f))
↓ postfix-
abc×+def^^-

Q- Which of the following is essential for converting an infix expression to the postfix form efficiently?
(i) An operator stack (✓)
(ii) An operand stack
(iii) An operand stack and an operator stack
(iv) A parse tree



Page No.: 124
Date: / /

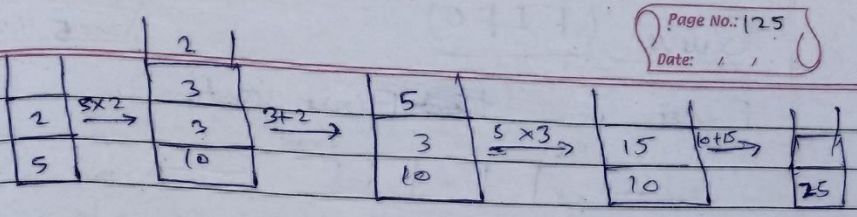
Q- Consider the following C program-

```
#include
#define EOF -1
void push(int); /* push the argument on the stack */
int pop(void); /* pop the top of the stack */
void flagError();
int main()
{ int c, m, n, r;
  while ((c = getchar()) != EOF)
  {
    if (isdigit(c))
        push(c);
    else if ((c == '+') || (c == '*'))
    {
        m = pop();
        n = pop();
        r = (c == '+') ? n+m : n*m;
        push(r);
    }
    else if (c != '')
        flagError();
  }
  printf("%c", pop());
}
```

What is the output of the program for the following input?

52*332+**+

(i) 15
(ii) 25 (✓)
(iii) 30
(iv) 150



Page No.: 125
Date: / /

Q- A function f defined on stacks of integers satisfies the following properties. $f(\phi) = 0$ and $f(push(S, i)) = \max(f(S), 0) + i$ for all stacks S and integers i. If a stack S contains the integers the integers 2, -3, 2, -1, 2, in order from bottom to top, what is f(S)?

(i) 6
(ii) 4
(iii) 3 (✓)
(iv) 2

$f(S) = 0$ - $i = 2, -3, 2, -1, 2$.

f(2)	i	max(f(2),0) + i
0	2	max(0,0)+2=0 +2 = 2
2	-3	max(2,0)-3 = 2-3 = -1
-1	2	max(-1,0)+2 = 0+2=2
2	-1	max(2,0)-1 = 2-1 = 1
1	2	max(1,0)+2 = 1+2 = 3 Ans

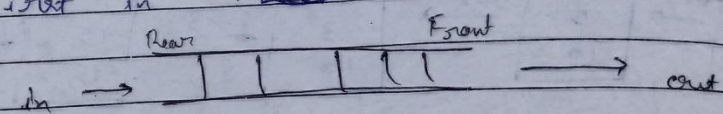
Q- The following sequence of operations is performed on stack:- PUSH(10), PUSH(20), POP, PUSH(10), PUSH(20), POP, POP, POP, PUSH(10), POP, The sequence of the value popped out is -

(i) 20, 10, 20, 10, 20
(ii) 20, 20, 10, 10, 20 (✓)
(iii) 10, 20, 20, 10, 20
(iv) 20, 20, 10, 20, 10

Queue (FIFO)

Page No.: 126
Date: 5/11/20

First in First out.

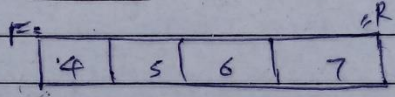


Operations:-

1- Enqueue - insert new element (at rear end)
2- Dequeue - delete element (at front end)

Linear Vs. Circular Queue -

→ Linear Queue -

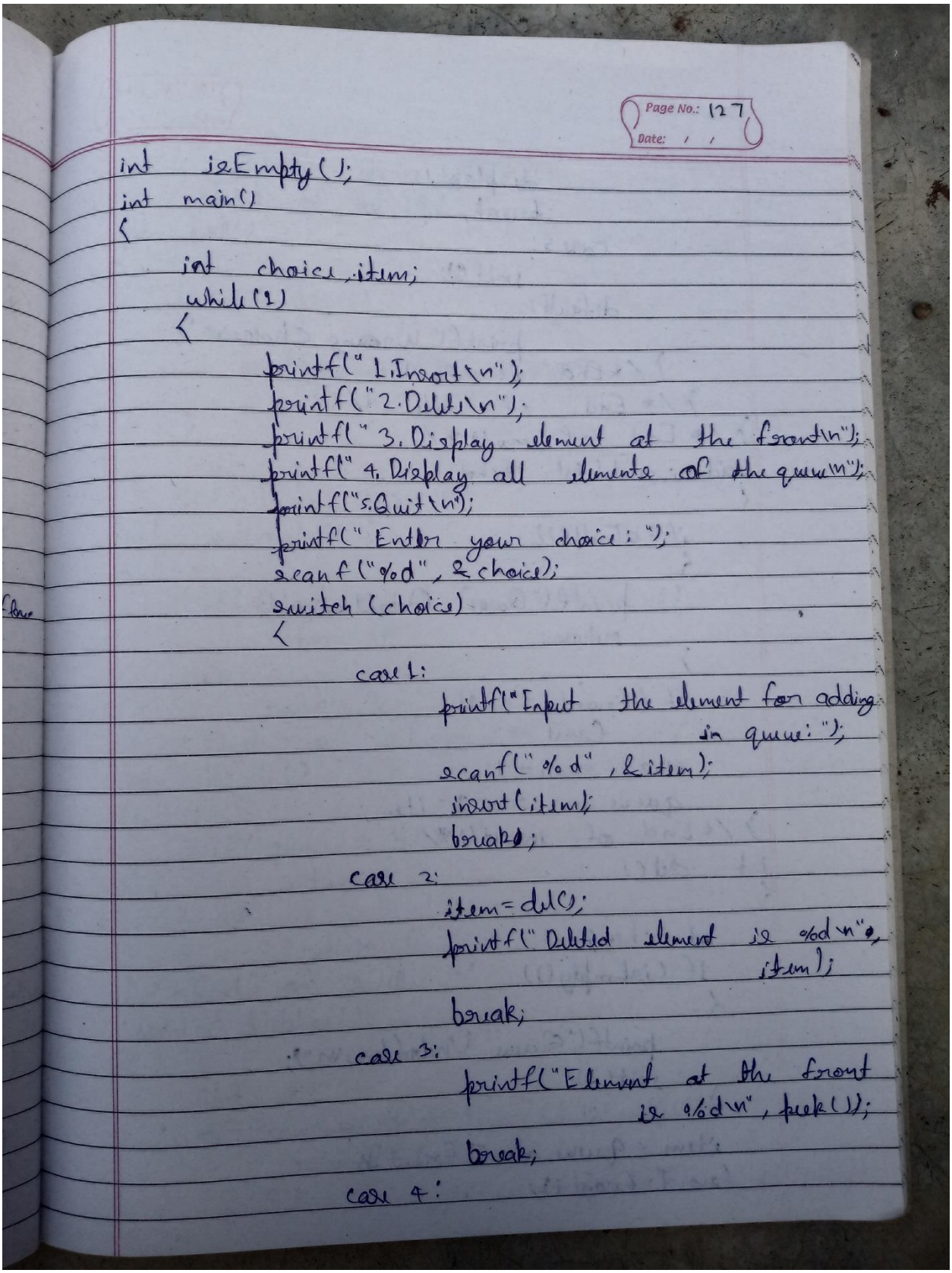


if R = MAX-1 overflow

Queue Using Array -

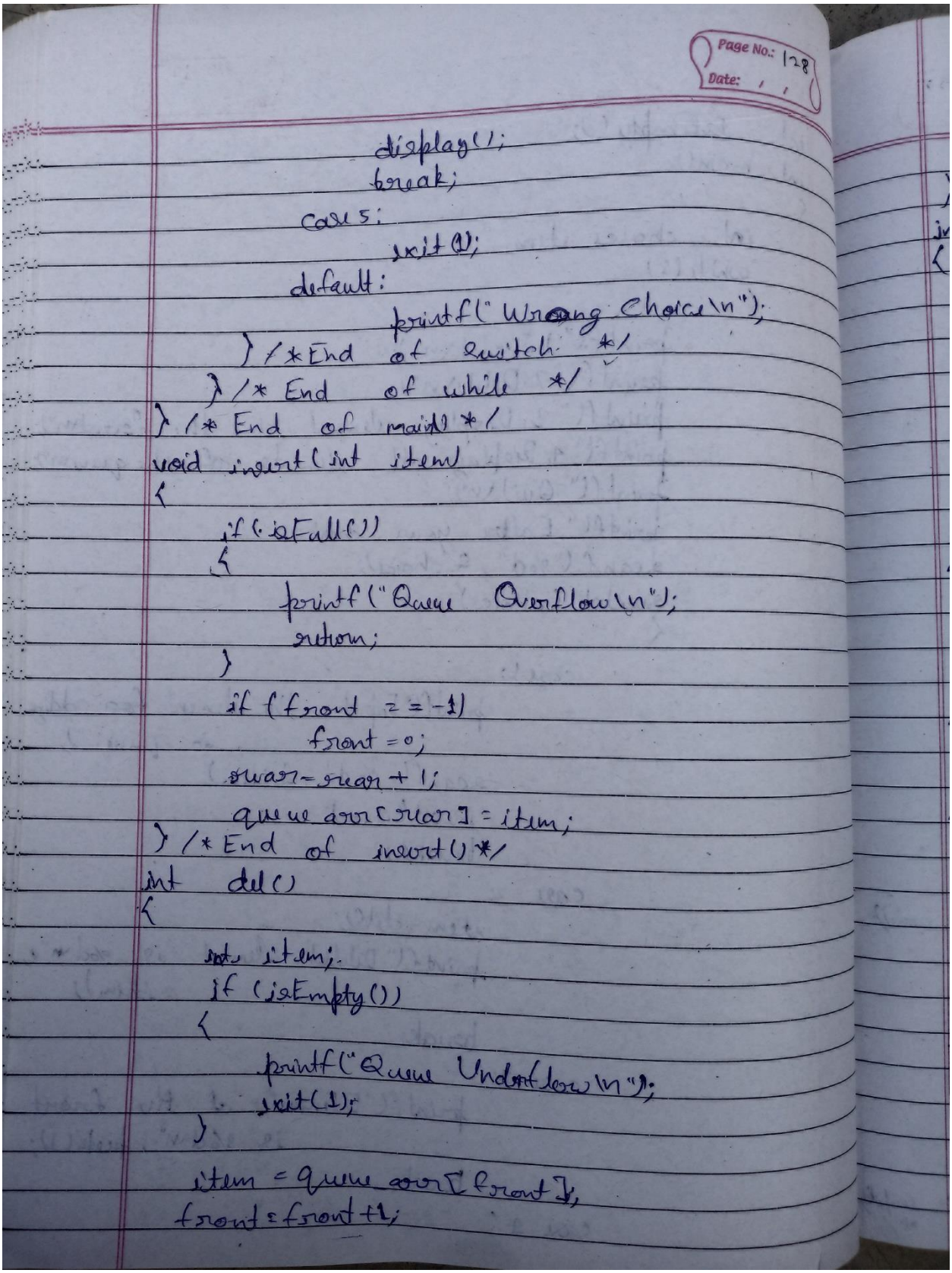
→ C Code -

```
#include <stdio.h>
#include<stdlib.h>
#define MAX 10
int queue_arr[MAX];
int rear = -1;
int front = -1;
void insert(int item);
int del();
int peek();
void display();
int isFull();
```

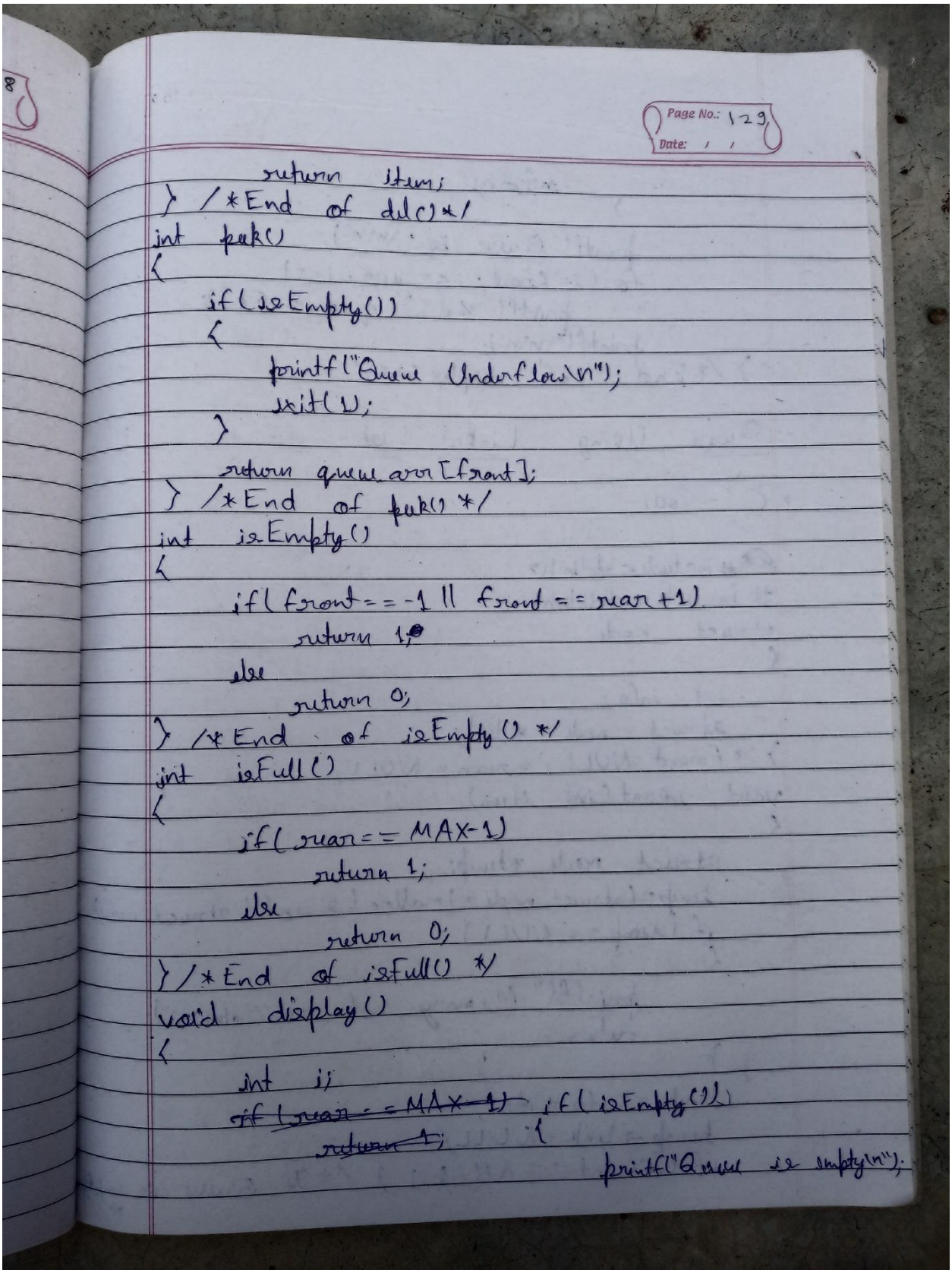


Page No.: 127
Date: / /

```
int isEmpty();
int main()
{
    int choice, item;
    while(1)
    {
        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Display element at the front\n");
        printf("4.Display all elements of the queue\n");
        printf("5.Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                printf("Input the element for adding
                        in queue: ");
                scanf("%d", &item);
                insert(item);
                break;
            case 2:
                item = del();
                printf("Deleted element is %d\n",
                        item);
                break;
            case 3:
                printf("Element at the front
                        is %d\n", peek());
                break;
            case 4:
```

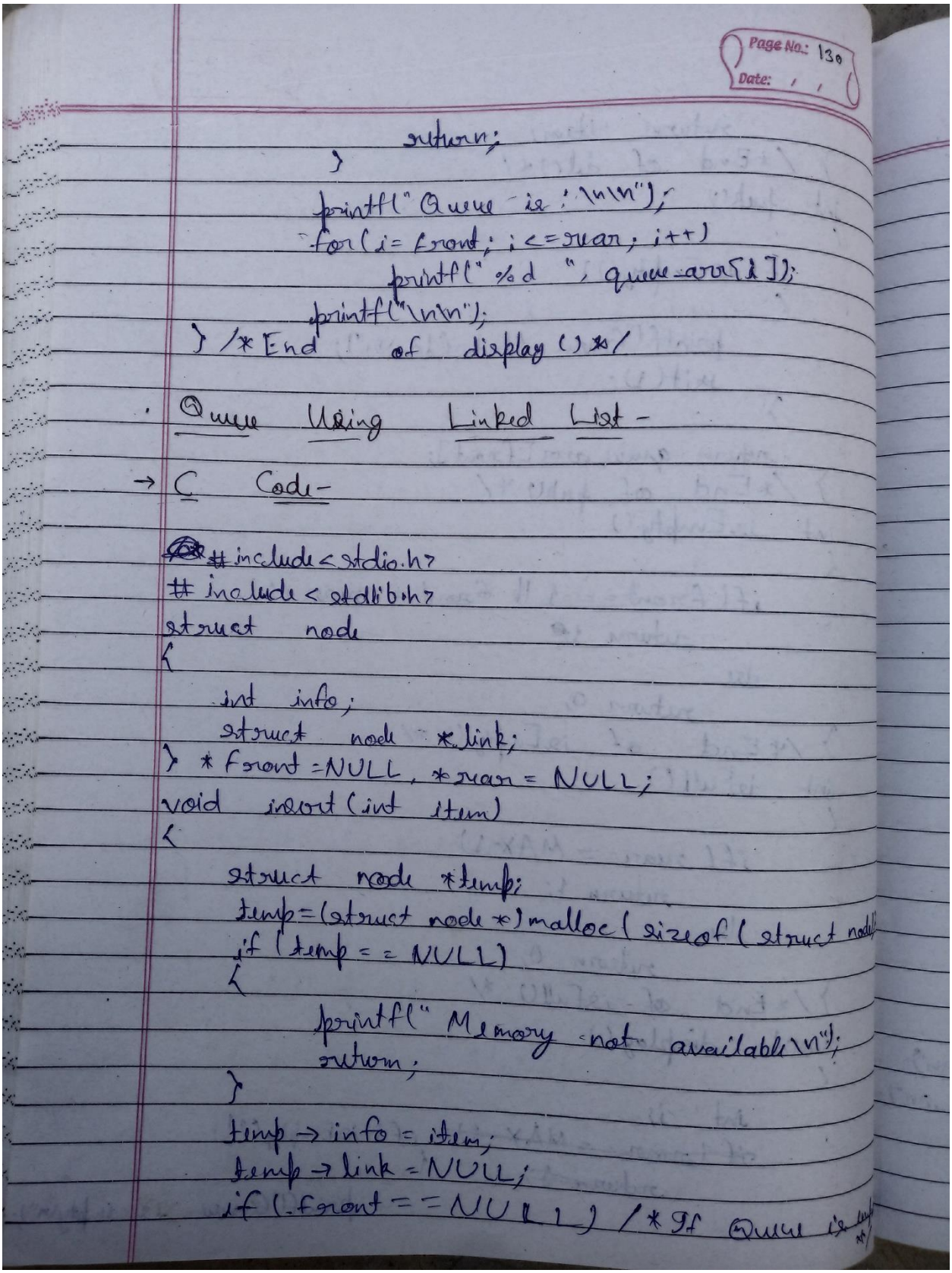


```
                display();
                break;
            case 5:
                exit(1);
            default:
                printf("Wrong Choice\n");
        } /* End of switch */
    } /* End of while */
} /* End of main() */
void insert(int item)
{
    if (isFull())
    {
        printf("Queue Overflow\n");
        return;
    }
    if (front == -1)
        front = 0;
    rear = rear + 1;
    queue_arr[rear] = item;
} /* End of insert() */
int del()
{
    int item;
    if (isEmpty())
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    item = queue_arr[front];
    front = front + 1;
```



Page No.: 129
Date: / /

```
        return item;
} /*End of del()*/
int peek()
{
    if(isEmpty())
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    return queue_arr[front];
} /*End of peek() */
int isEmpty()
{
    if(front==-1 || front==rear+1)
        return 1;
    else
        return 0;
} /*End of isEmpty() */
int isFull()
{
    if(rear==MAX-1)
        return 1;
    else
        return 0;
}/*End of isFull() */
void display()
{
    int i;
    if(rear==MAX-1)  if(isEmpty())
        return 1;        {
                         printf("Queue is empty\n");
```

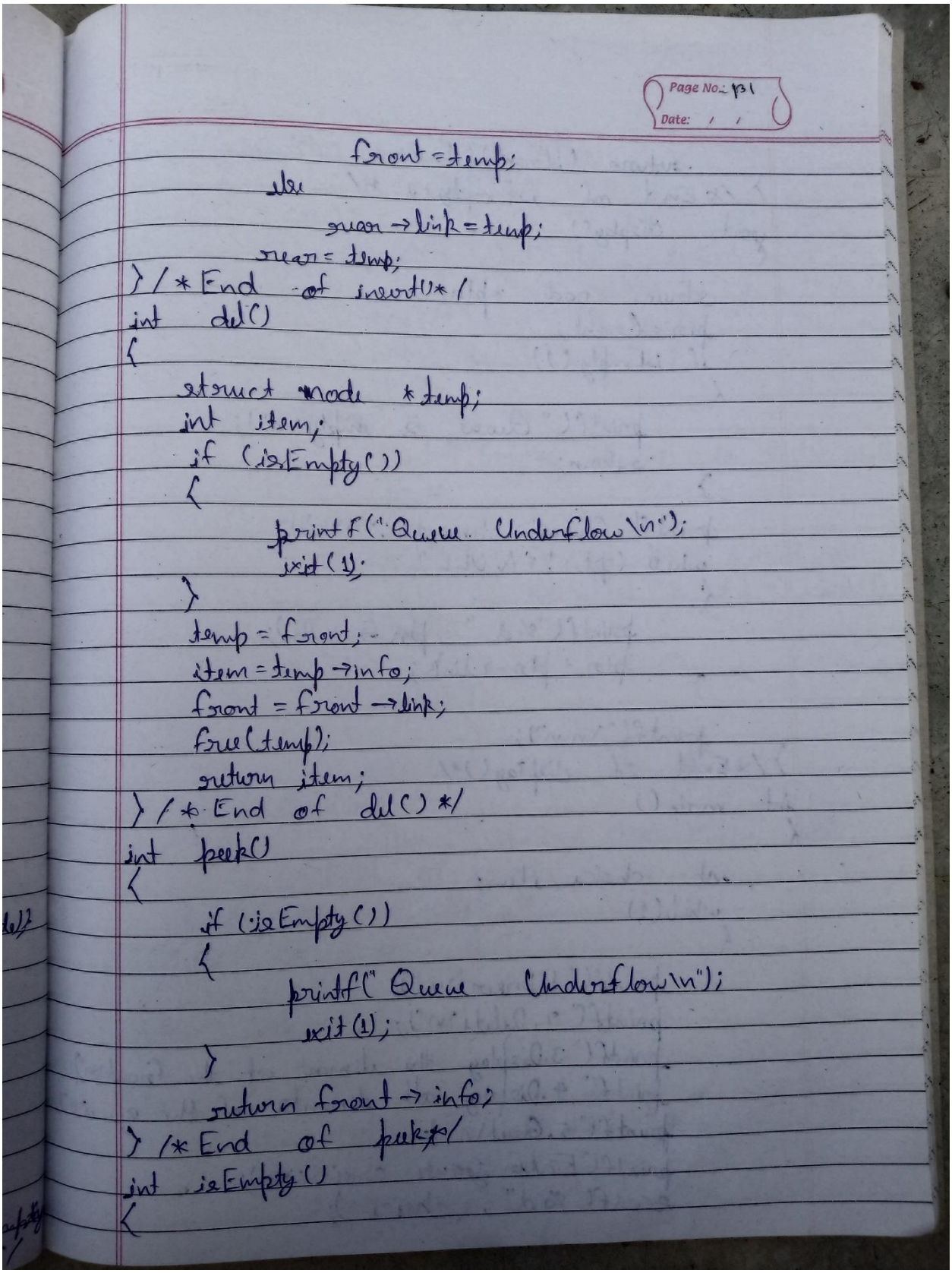


```
        return;
    }
    printf("Queue is :\n\n");
    for(i=front; i<=rear; i++)
        printf("%d ", queue_arr[i]);
    printf("\n\n");
} /* End of display() */
```

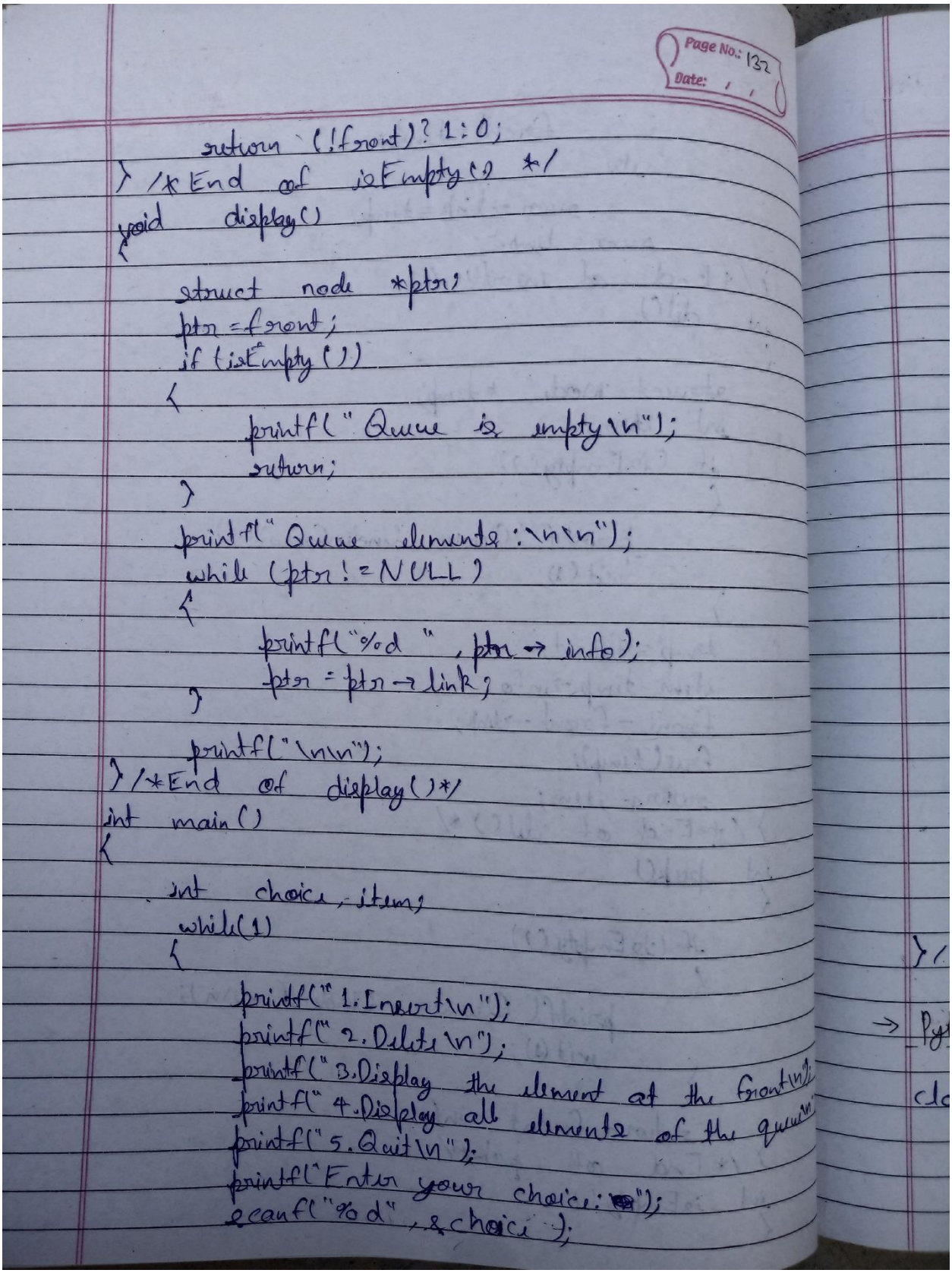
Queue Using Linked List -

→ C Code-

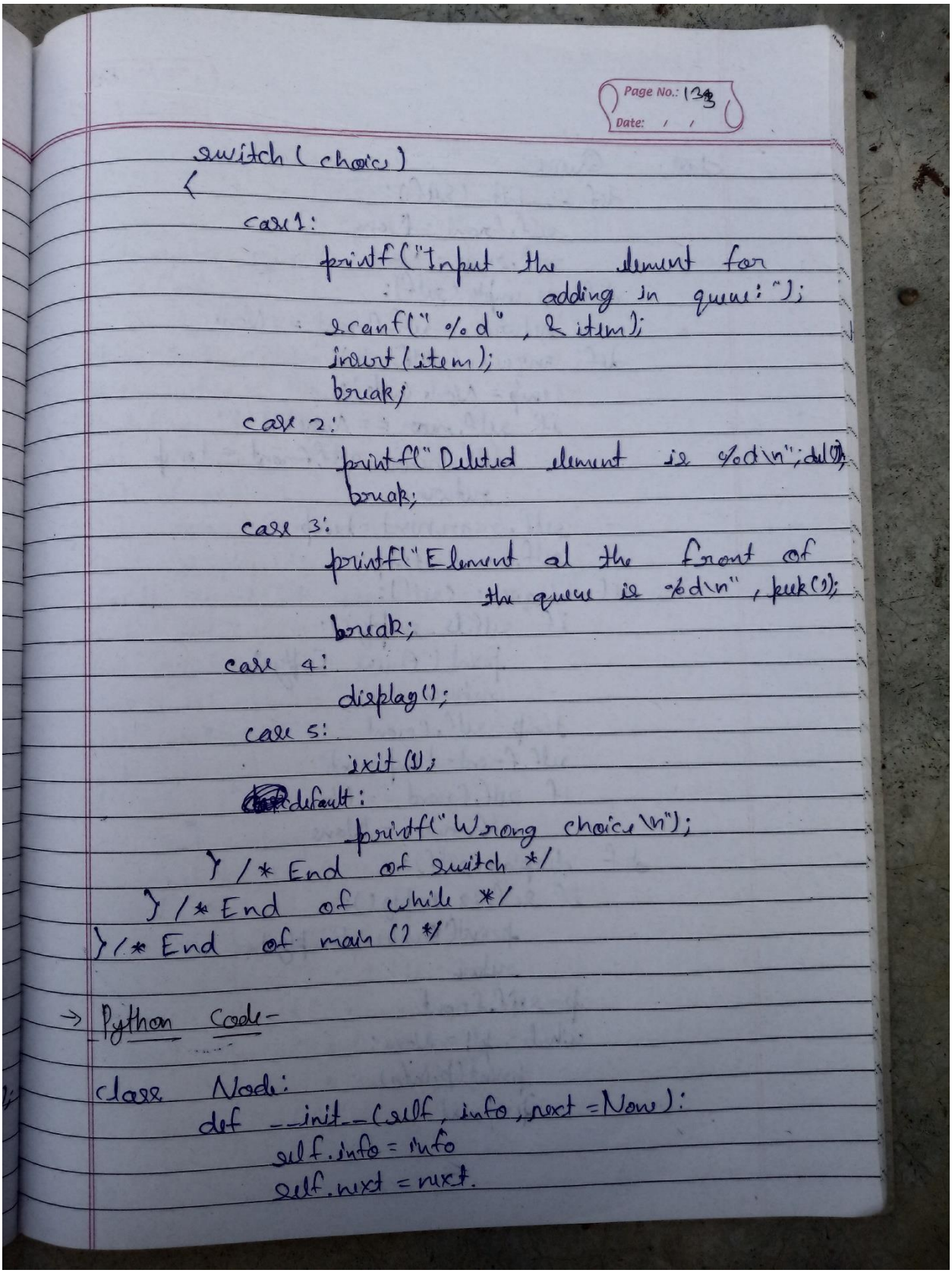
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int info;
    struct node *link;
} *front=NULL, *rear=NULL;
void insert(int item)
{
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node
    if(temp==NULL)
    {
        printf("Memory not available\n");
        return;
    }
    temp->info=item;
    temp->link=NULL;
    if(front==NULL) /* If Queue is
```



```
                front = temp;
            else
                rear -> link = temp;
            rear = temp;
} /* End of insert() */
int del()
{
    struct node *temp;
    int item;
    if (isEmpty())
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    temp = front;
    item = temp -> info;
    front = front -> link;
    free(temp);
    return item;
} /* End of del() */
int peek()
{
    if (isEmpty())
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    return front -> info;
} /* End of peek */
int isEmpty()
{
```



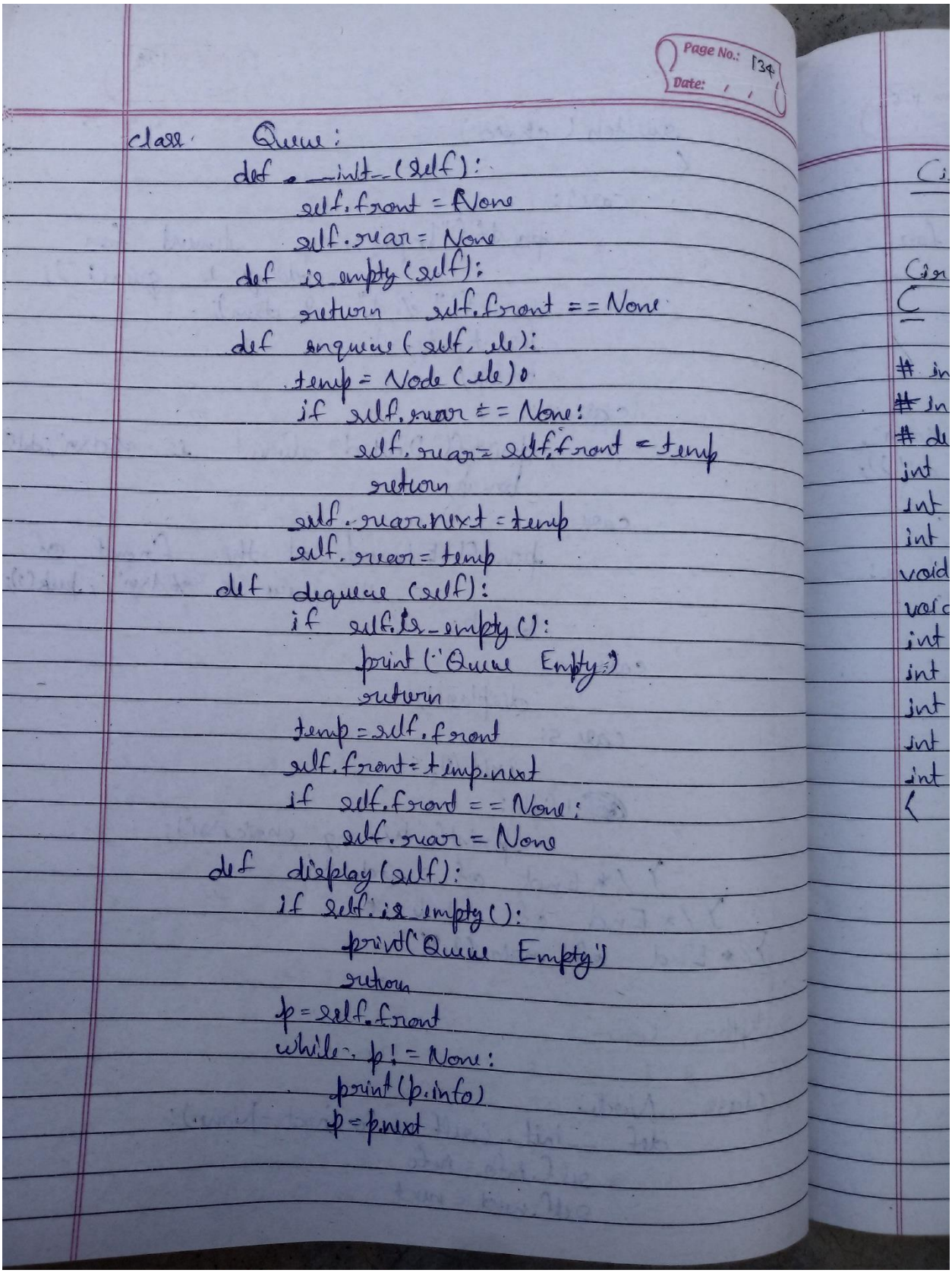
```
    return (!front)? 1:0;
} /* End of isEmpty() */
void display()
{
    struct node *ptr;
    ptr = front;
    if (isEmpty())
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements :\n\n");
    while (ptr != NULL)
    {
        printf("%d ", ptr -> info);
        ptr = ptr -> link;
    }
    printf("\n\n");
} /* End of display() */
int main()
{
    int choice, item;
    while(1)
    {
        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Display the element at the front\n");
        printf("4.Display all elements of the queue\n");
        printf("5.Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```



```
switch (choice)
{
    case 1:
        printf("Input the element for
                adding in queue: ");
        scanf("%d", &item);
        insert(item);
        break;
    case 2:
        printf("Deleted element is %d\n", del();
        break;
    case 3:
        printf("Element at the front of
                the queue is %d\n", peek());
        break;
    case 4:
        display();
    case 5:
        exit(1);
    default:
        printf("Wrong choice\n");
    } /* End of switch */
  } /* End of while */
} /* End of main() */
```

→ Python Code-

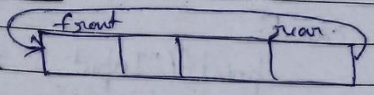
```
class Node:
    def __init__(self, info, next=None):
        self.info = info
        self.next = next
```



```
class Queue:
    def __init__(self):
        self.front = None
        self.rear = None
    def is_empty(self):
        return self.front == None
    def enqueue(self, ele):
        temp = Node(ele)
        if self.rear == None:
            self.rear = self.front = temp
            return
        self.rear.next = temp
        self.rear = temp
    def dequeue(self):
        if self.is_empty():
            print("Queue Empty")
            return
        temp = self.front
        self.front = temp.next
        if self.front == None:
            self.rear = None
    def display(self):
        if self.is_empty():
            print("Queue Empty")
            return
        p = self.front
        while p != None:
            print(p.info)
            p = p.next
```

Page No.: 135
Date: / /

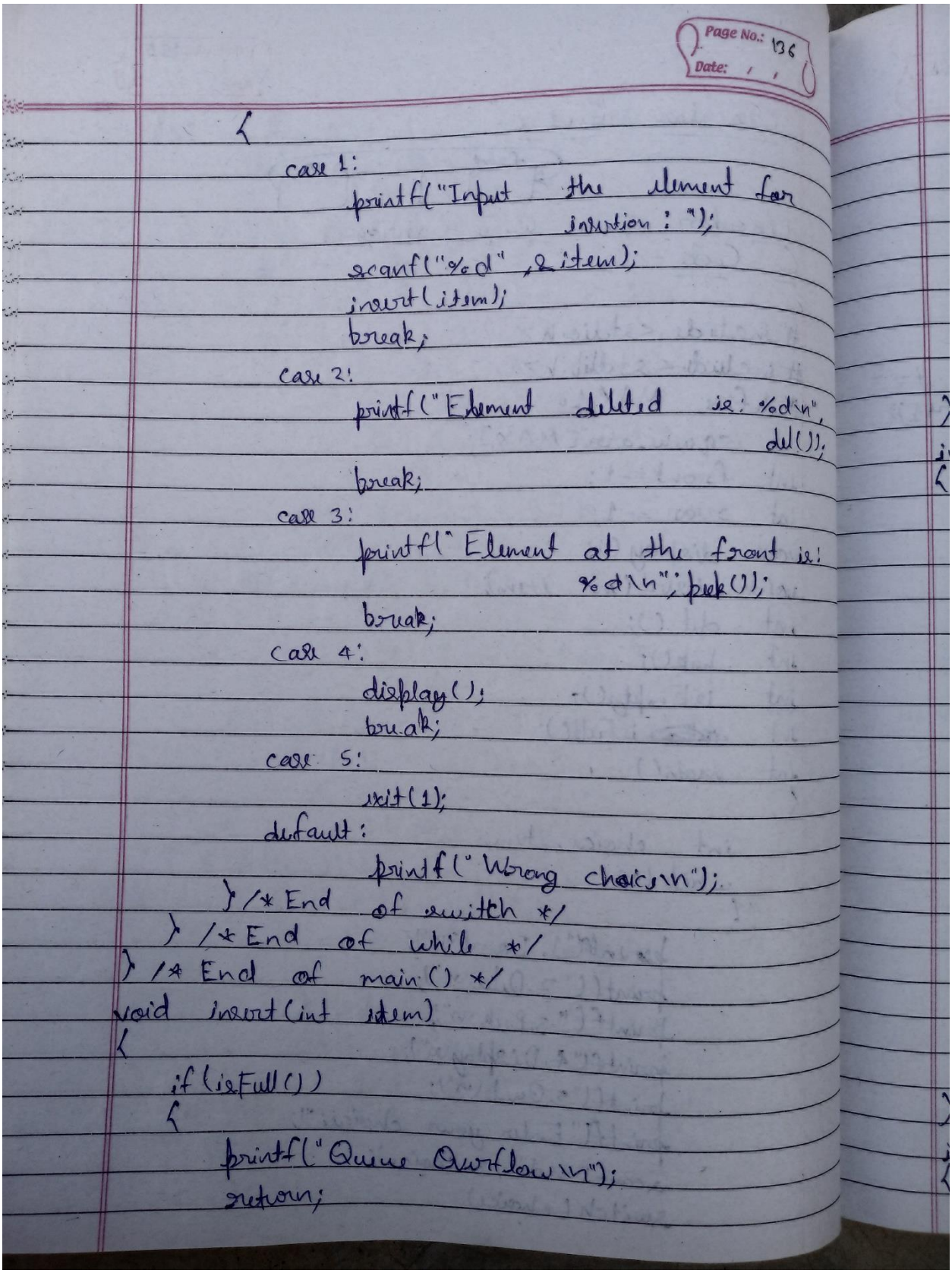
Circular Queue -



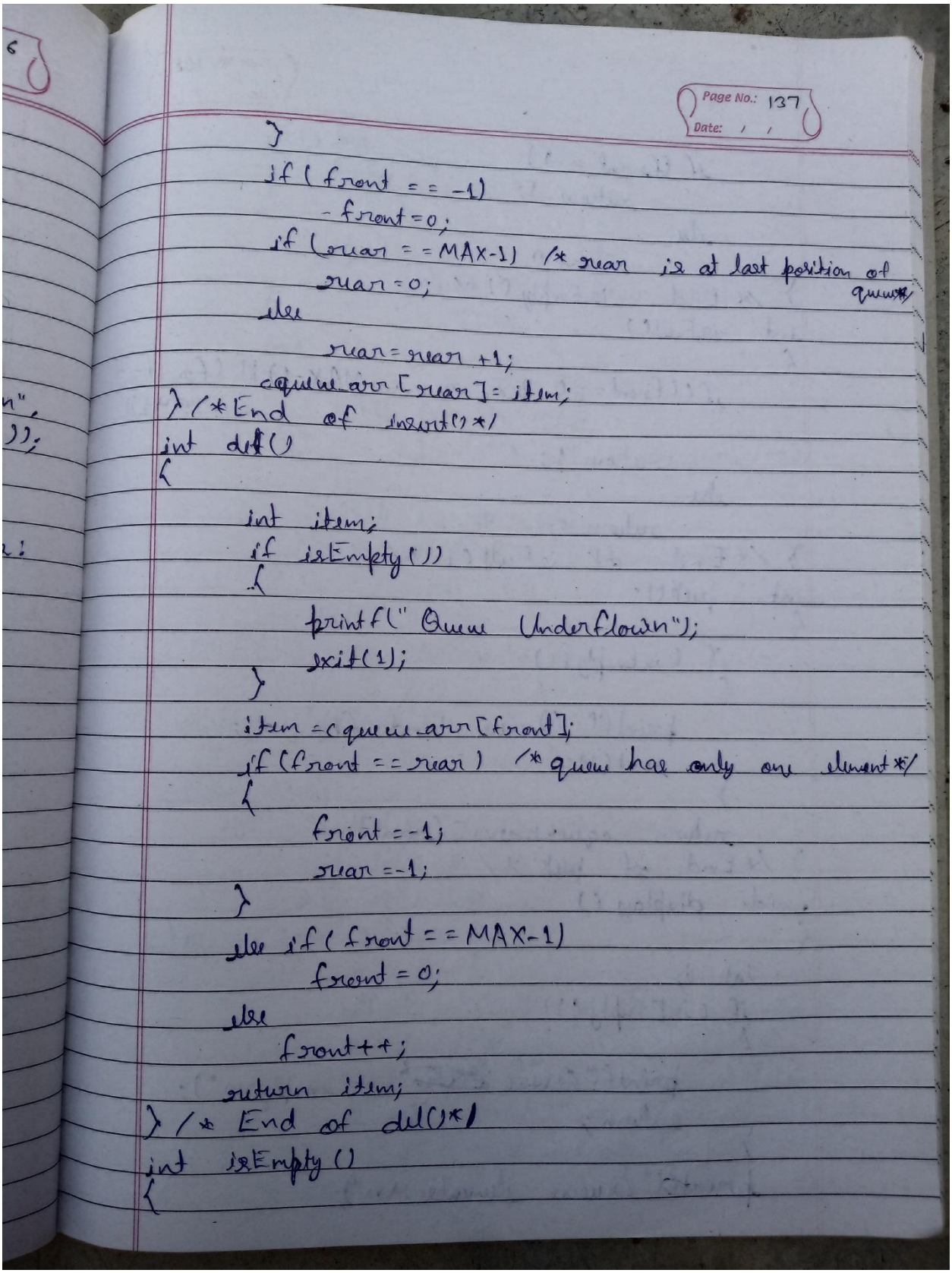
Circular queue using array-

C Code -

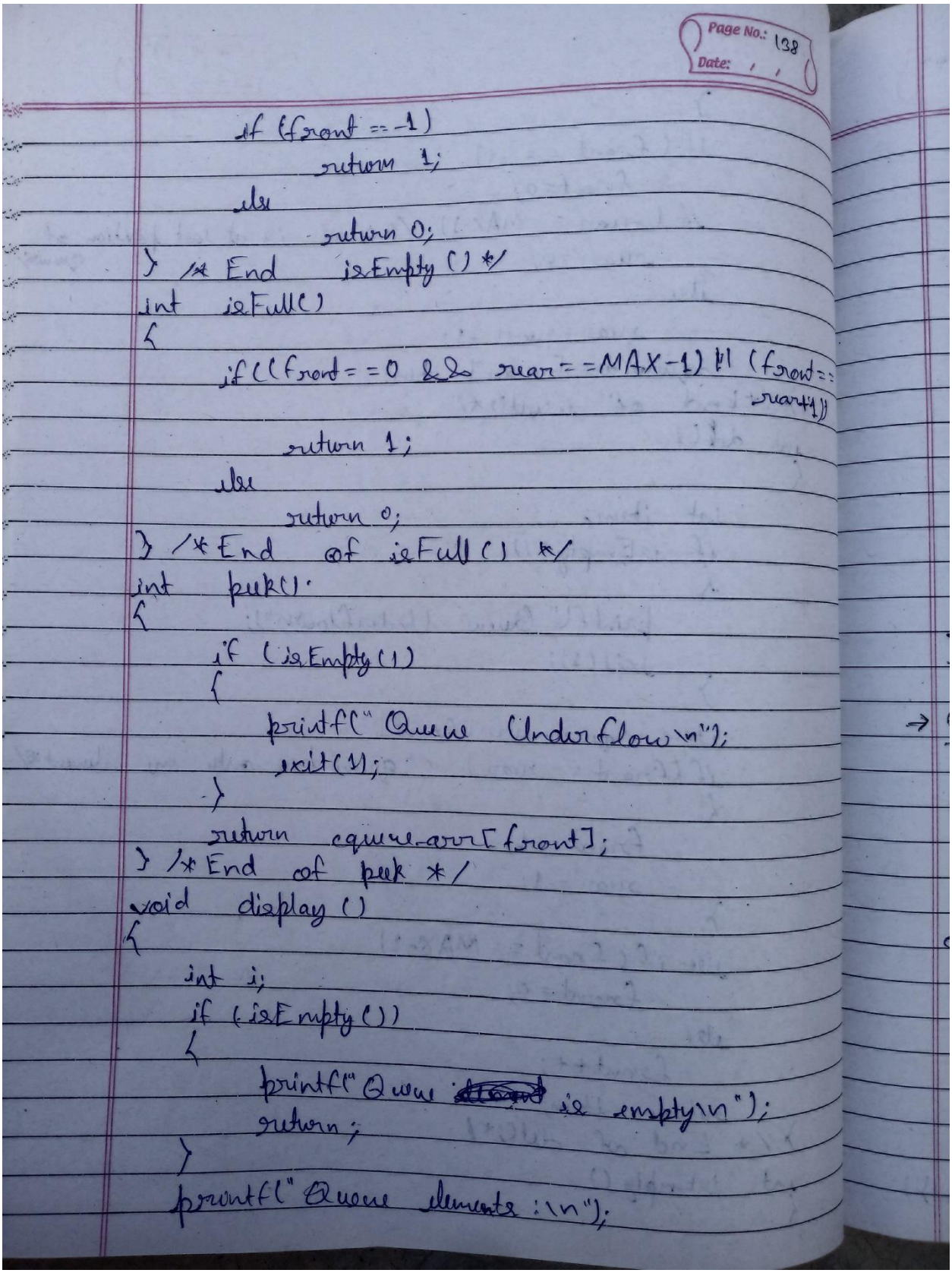
```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
int cqueue_arr[MAX];
int front=-1;
int rear=-1;
void display();
void insert(int item);
int del();
int peek();
int isEmpty();
int isFull();
int main()
{
    int choice, item;
    while(1)
    {
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Peek\n");
        printf("4. Display\n");
        printf("5. Quit\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch(choice)
```



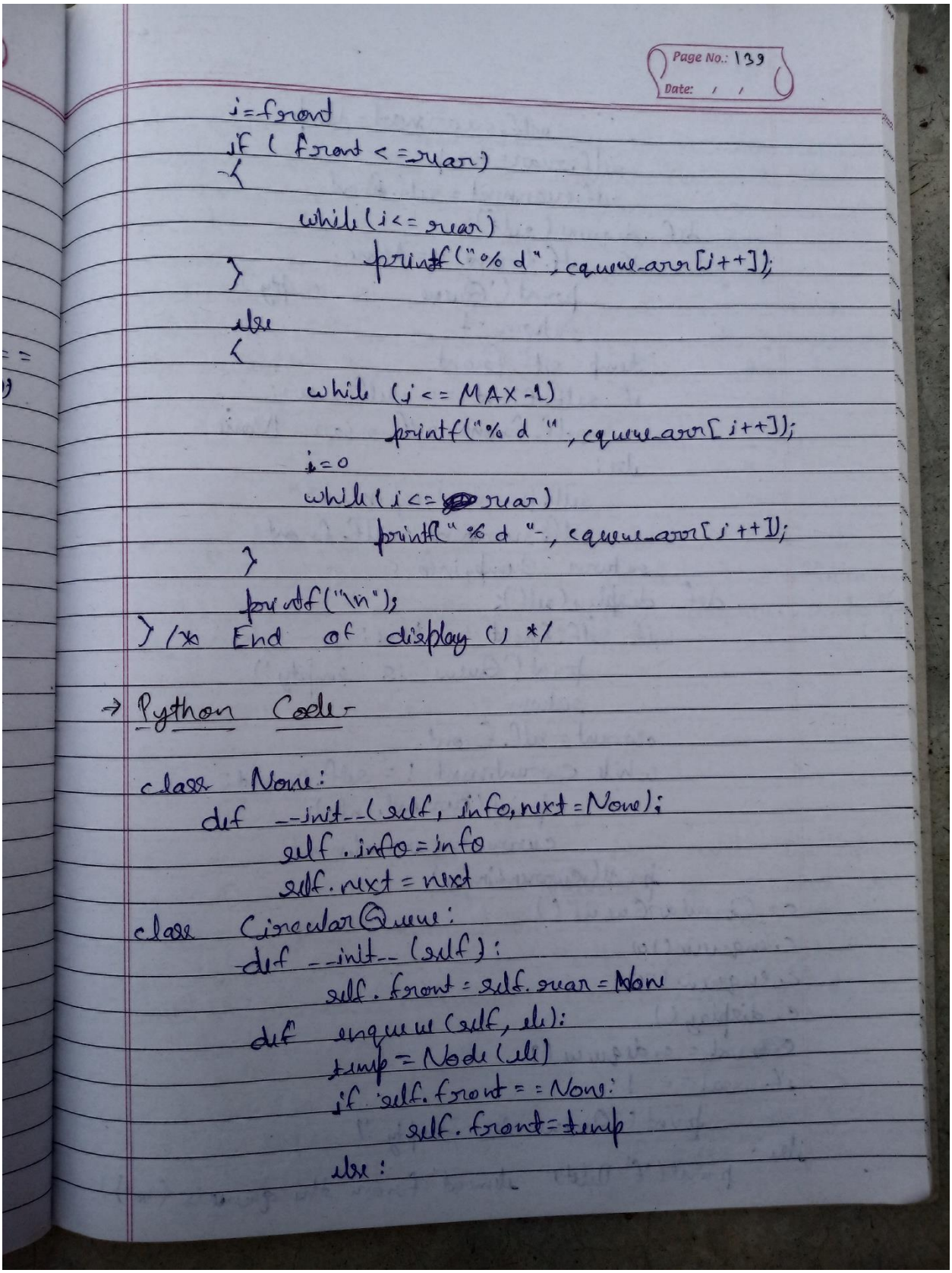
```
{
    case 1:
        printf("Input the element for
                insertion : ");
        scanf("%d", &item);
        insert(item);
        break;
    case 2:
        printf("Element deleted is: %d\n",
                del());
        break;
    case 3:
        printf("Element at the front is:
                %d\n", peek());
        break;
    case 4:
        display();
        break;
    case 5:
        exit(1);
    default:
        printf("Wrong choice\n");
    } /* End of switch */
} /* End of while */
} /* End of main() */
void insert(int item)
{
    if(isFull())
    {
        printf("Queue Overflow\n");
        return;
```



```
        }
        if (front == -1)
            front = 0;
        if (rear == MAX-1)  /* rear is at last position of queue */
            rear = 0;
        else
            rear = rear + 1;
        queue_arr[rear] = item;
} /* End of insert() */
int del()
{
    int item;
    if isEmpty()
    {
        printf("Queue Underflown");
        exit(1);
    }
    item = queue_arr[front];
    if (front == rear)  /* queue has only one element */
    {
        front = -1;
        rear = -1;
    }
    else if (front == MAX-1)
        front = 0;
    else
        front++;
    return item;
} /* End of del() */
int isEmpty()
{
```



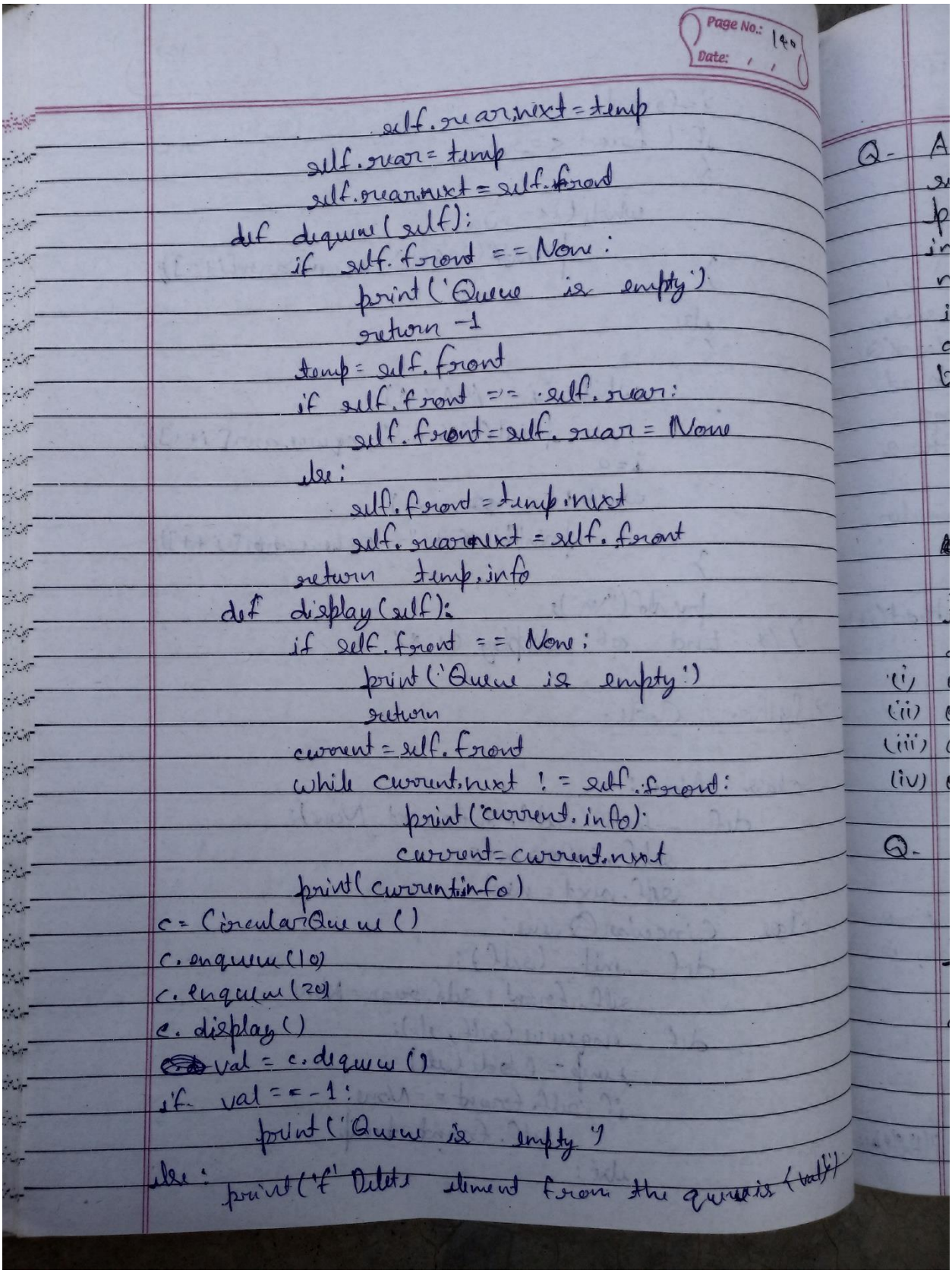
```
        if (front == -1)
            return 1;
        else
            return 0;
} /* End isEmpty() */
int isFull()
{
    if((front==0 && rear==MAX-1) || (front==
                                    rear+1))
        return 1;
    else
        return 0;
} /* End of isFull() */
int peek()
{
    if (isEmpty())
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    return cqueue_arr[front];
} /* End of peek */
void display()
{
    int i;
    if (isEmpty())
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements :\n");
```



```
Page No.: 139
Date: / /

i=front
if ( front <= rear)
{
    while(i<=rear)
        printf("%d", cqueue_arr[i++]);
}
else
{
    while (i<=MAX-1)
        printf("%d ", cqueue_arr[i++]);
    i=0
    while(i<=rear)
        printf("%d ", cqueue_arr[i++]);
}
printf("\n");
} /* End of display() */
```

→ Python Code-

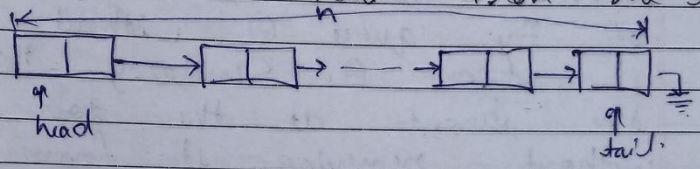
```
class Node:
    def __init__(self, info, next=None):
        self.info=info
        self.next=next
class CircularQueue:
    def __init__(self):
        self.front = self.rear = None
    def enqueue(self, ele):
        temp = Node(ele)
        if self.front == None:
            self.front=temp
        else:
```



```
            self.rear.next = temp
        self.rear = temp
        self.rear.next = self.front
    def dequeue(self):
        if self.front == None:
            print('Queue is empty')
            return -1
        temp = self.front
        if self.front == self.rear:
            self.front = self.rear = None
        else:
            self.front = temp.next
            self.rear.next = self.front
        return temp.info
    def display(self):
        if self.front == None:
            print('Queue is empty')
            return
        current = self.front
        while current.next != self.front:
            print(current.info)
            current = current.next
        print(current.info)
c = CircularQueue()
c.enqueue(10)
c.enqueue(20)
c.display()
val = c.dequeue()
if val == -1:
    print('Queue is empty')
else:
    print('Deleted element from the queue is', val)
```

Page No.: 141
Date: / /

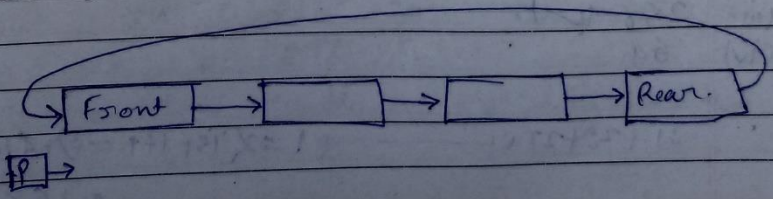
Q- A queue is implemented use a non-circular singly linked list. The queue has a head pointer and a tail pointer, as shown in the figure. Let n denote the number of nodes in the queue. Let enqueue be implemented by inserting a new node at the head, and dequeue be implemented by deletion of a node from the tail.

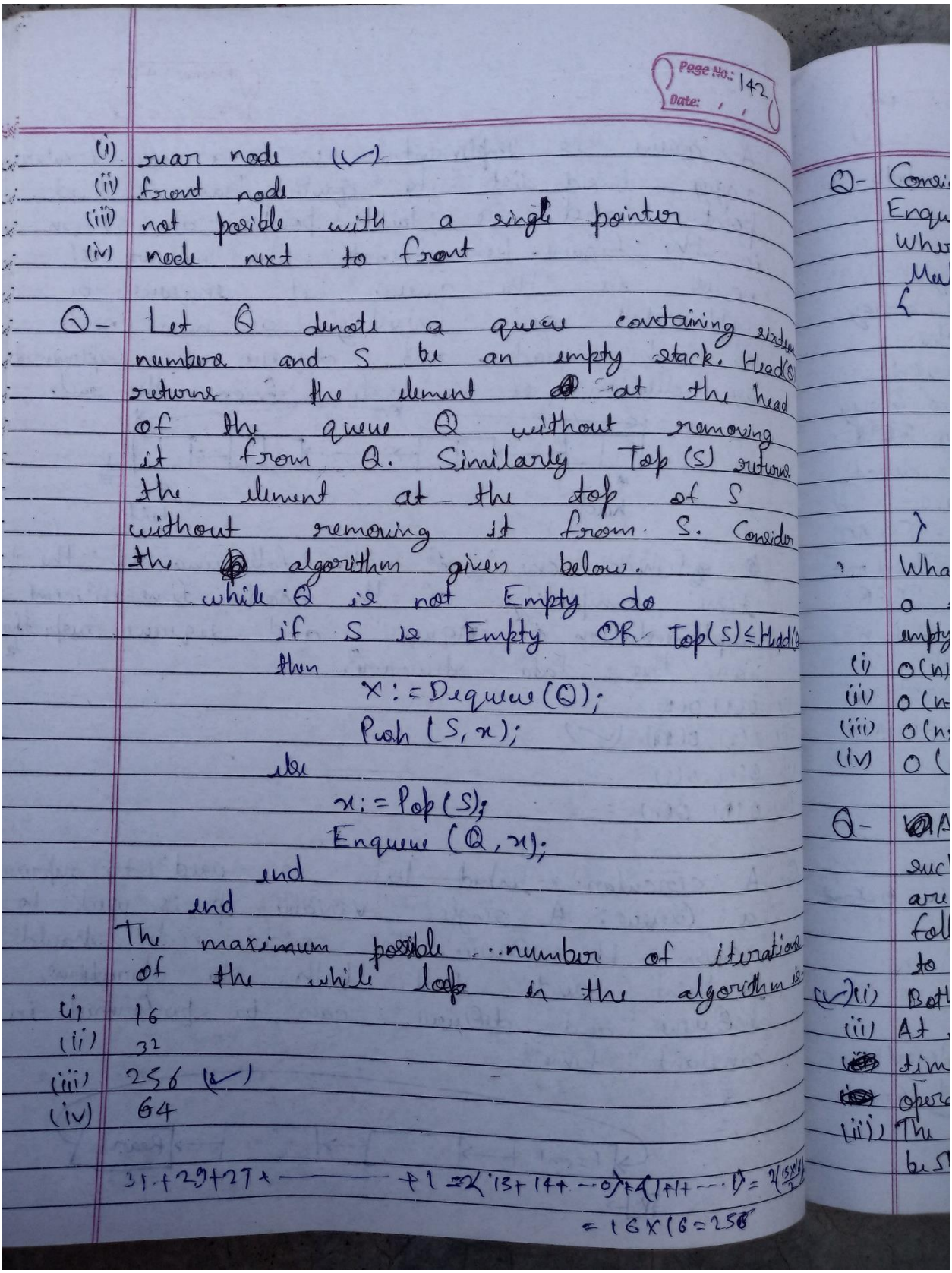


Which one of the following is the time complexity of the most time-efficient implementation of enqueue and dequeue, respectively for this data structure?

(i) $\Theta(1), \Theta(1)$
(ii) $\Theta(1), \Theta(n)$ (✓)
(iii) $\Theta(n), \Theta(1)$
(iv) $\Theta(n), \Theta(n)$

Q- A circular linked list is used to represent a Queue. A single variable p is used to access the Queue. To which node should p point such that both the operations enQueue and deQueue can be performed in constant time?





(i) rear node (✓)
(ii) front node
(iii) not posible with a single pointer
(iv) node next to front

Q- Let Q denote a queue containing sixteen numbers and S be an empty stack. Head(Q) returns the element at the head of the queue Q without removing it from Q. Similarly Top(S) returns the element at the top of S without removing it from S. Consider the algorithm given below.

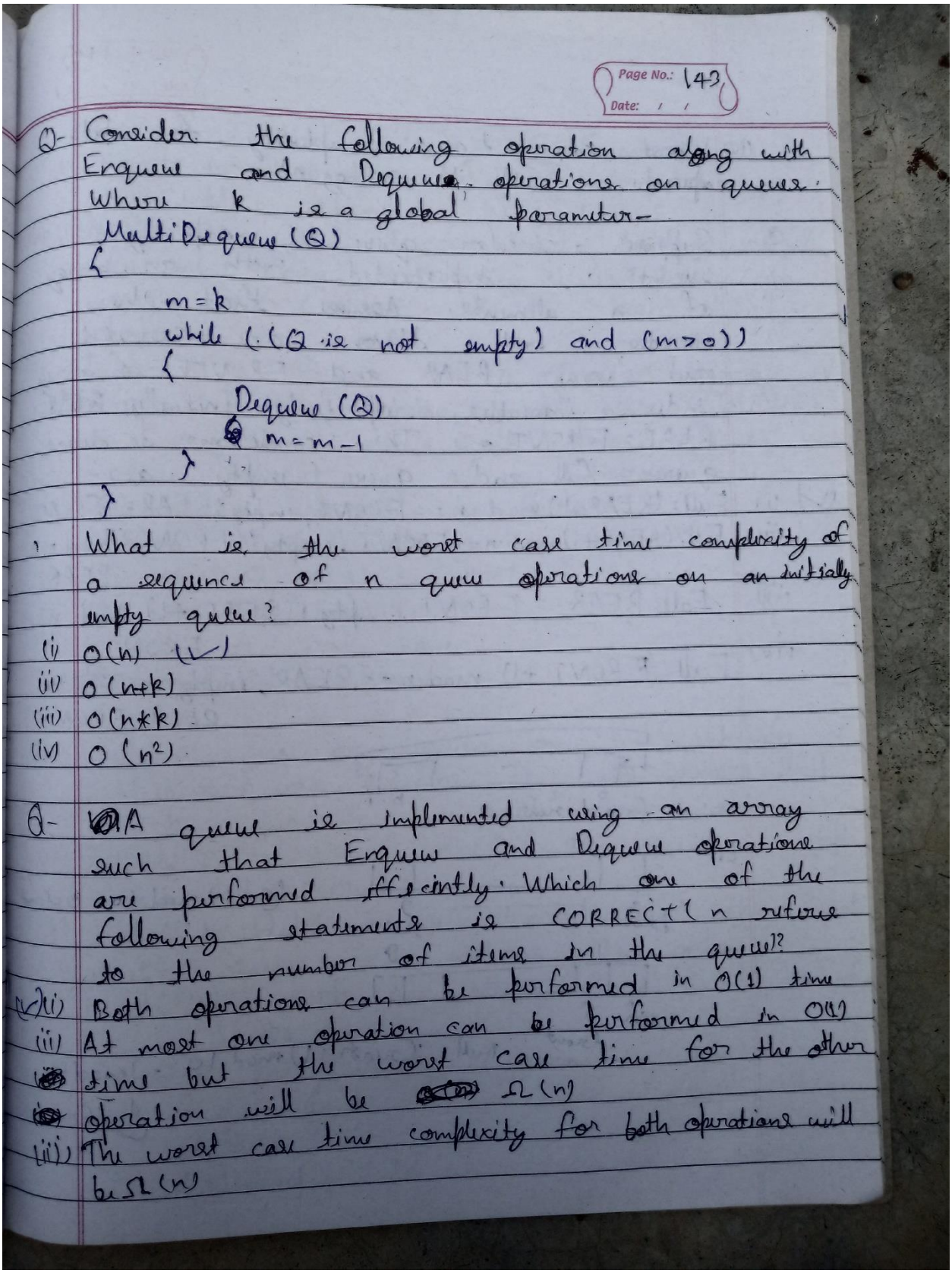
```
while Q is not Empty do
    if S is Empty OR Top(S) ≤ Head(Q)
    then
        x := Dequeue(Q);
        Push(S, x);
    else
        x := Pop(S);
        Enqueue(Q, x);
    end
end
```

The maximum possible number of iterations of the while loop in the algorithm is:

(i) 16
(ii) 32
(iii) 256 (✓)
(iv) 64

$31+29+27+\cdots+1 = 2(1+3+5+\cdots+15) - 0 + 4(1+1+\cdots+1) = 2\left(\frac{15\times16}{2}\right)$

$= 16\times16 = 256$



Page No.: 143
Date: / /

Q- Consider the following operation along with Enqueue and Dequeue operations on queues. Where k is a global parameter-

```
MultiDequeue (Q)
{
    m = k
    while ((Q is not empty) and (m > 0))
    {
        Dequeue (Q)
        m = m - 1
    }
}
```

What is the worst case time complexity of a sequence of n queue operations on an initially empty queue?

(i) O(n)
(ii) O(n+k)
(iii) O(n*k)
(iv) O(n²)

Q- A queue is implemented using an array such that Enqueue and Dequeue operations are performed efficiently. Which one of the following statements is CORRECT (n refers to the number of items in the queue)?

(i) Both operations can be performed in O(1) time
(ii) At most one operation can be performed in O(1) time but the worst case time for the other operation will be Ω(n)
(iii) The worst case time complexity for both operations will be Ω(n)

(iv) Worst case time complexity for both operations will be $\Omega(\log n)$.

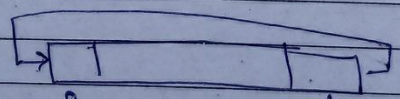
Q- Suppose a circular queue of capacity (n+1) elements is implemented with an array of n elements. Assume that the insertion and deletion are carried out using REAR and FRONT as array index variables, respectively. Initially, REAR = FRONT = 0. The conditions to detect queue full and queue empty are-

(A) (i) Full: (REAR+1) mod n == FRONT, empty: REAR == FRONT

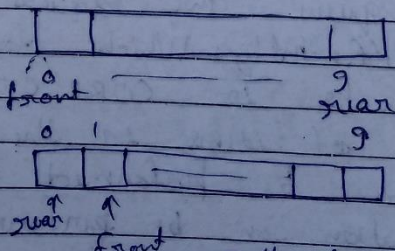
(ii) Full: (REAR+1) mod n == FRONT, empty: (FRONT+1) mod n == REAR

(iii) Full: REAR == FRONT, empty: (REAR+1) mod n == FRONT

(iv) Full: (FRONT+1) mod n == REAR, empty: REAR = FRONT

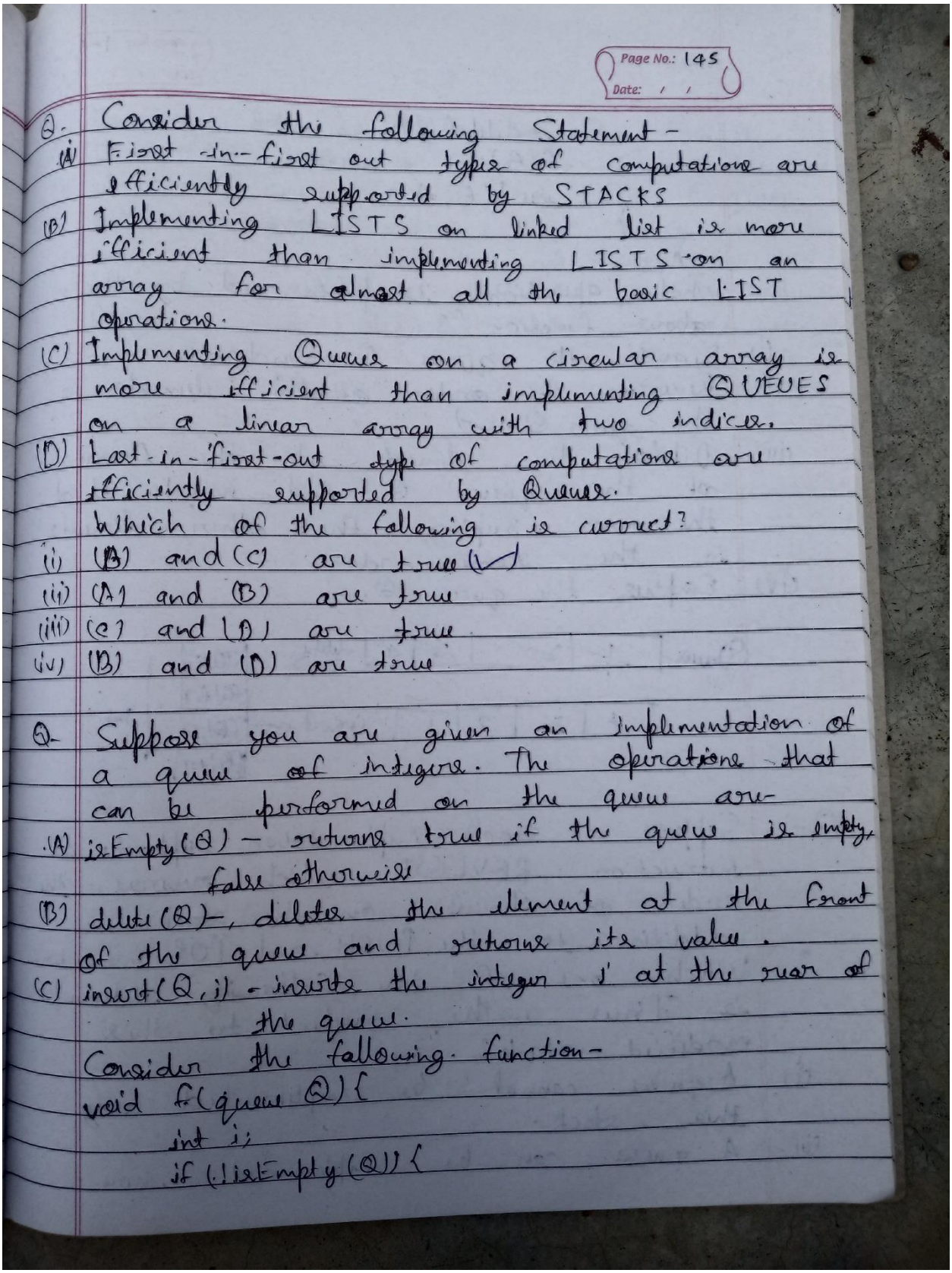


Let



(9+1) mod 10 = 0 = front

Full = (rear+1) mod 10 = 1 mod 10 = 1 = front



Page No.: 145
Date: / /

Q. Consider the following Statement -
(A) First-in-first out types of computations are efficiently supported by STACKS
(B) Implementing LISTS on linked list is more efficient than implementing LISTS on an array for almost all the basic LIST operations.
(C) Implementing Queues on a circular array is more efficient than implementing QUEUES on a linear array with two indices.
(D) Last-in-first-out type of computations are efficiently supported by Queues.
Which of the following is correct?
(i) (B) and (C) are true (✓)
(ii) (A) and (B) are true
(iii) (C) and (D) are true
(iv) (B) and (D) are true

Q. Suppose you are given an implementation of a queue of integers. The operations that can be performed on the queue are-
(A) isEmpty(Q) - returns true if the queue is empty, false otherwise
(B) delete(Q) - deletes the element at the front of the queue and returns its value.
(C) insert(Q, i) - inserts the integer 'i' at the rear of the queue.
Consider the following function-

```
void f(queue Q){
    int i;
    if (!isEmpty(Q)){
```

```
        i = delete(Q);
        f(Q);
        insert(Q, i);
    }
}
```

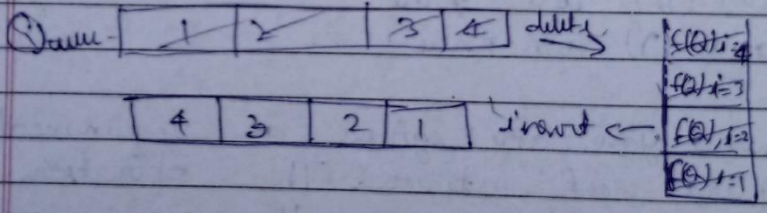
What operation is performed by the above function f?

(i) Leaves the queue Q unchanged.

(ii) Reverses the order of the elements in the queue Q. (✓)

(iii) Deletes the element at the front of the queue Q and inserts it at the rear keeping the other elements in the same order

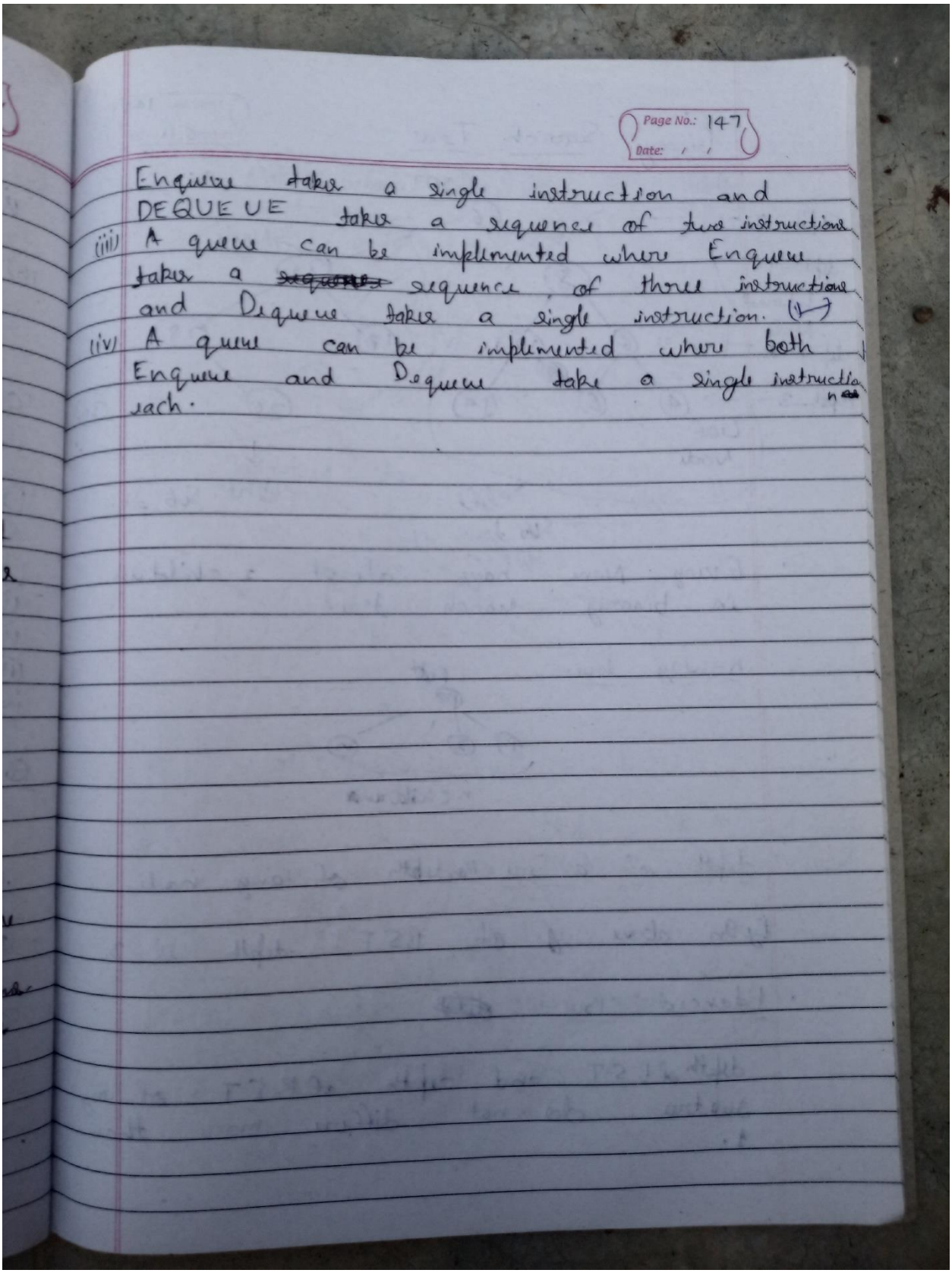
(iv) Empties the queue Q.



Q- Suppose a stack implementation supports an instruction REVERSE which reverses the order of elements on the stack, in addition to the PUSH and POP instructions. Which one of the following statements is True with respect to this modified stack?

(i) A queue cannot be implemented using this stack.

(ii) A queue can be implemented where



Page No.: 147
Date: / /

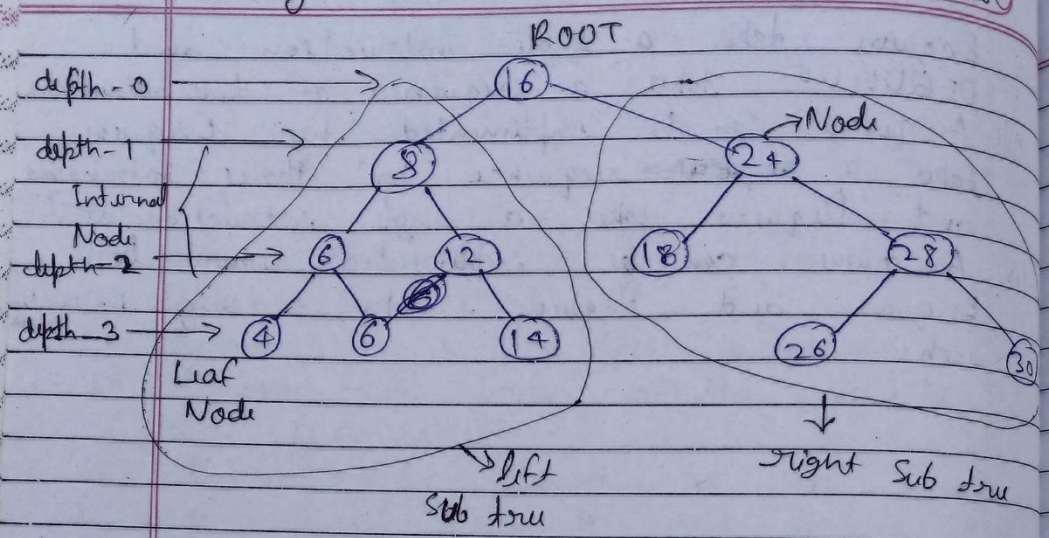
Enqueue take a single instruction and DEQUEUE take a sequence of two instructions.

(iii) A queue can be implemented where Enqueue take a ~~sequence~~ sequence of three instructions and Dequeue take a single instruction. (✓)

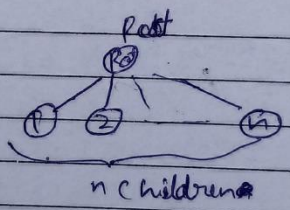
(iv) A queue can be implemented where both Enqueue and Dequeue take a single instruction each.

Binary Search Tree

Page No.: 148
Date: 6/11/20



- Every Node have atmost 2 children in binary search tree.
- n-way tree-



- depth of a Tree = Max depth of any node.

Eg. In above eg. of BST depth is 3.

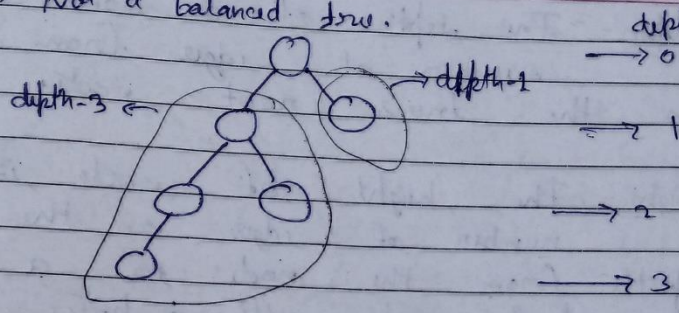
- Balanced Tree-

depth of LST and depth of RST of any subtree do not differ more than 1.

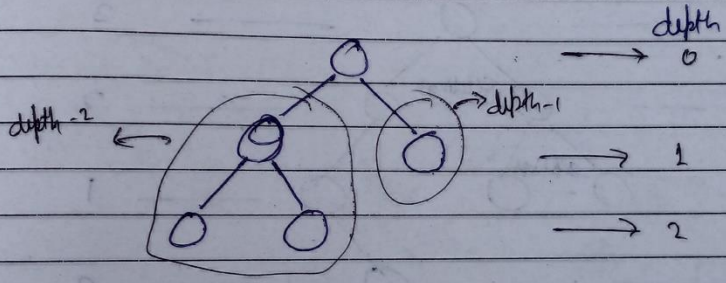
Note-
(i)
(ii)

Page No.: 149
Date: / /

Eg- (i) Not a balanced tree.

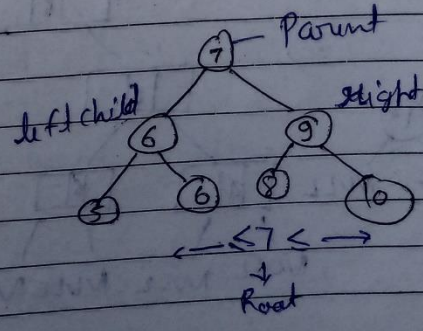


(ii) Balanced tree



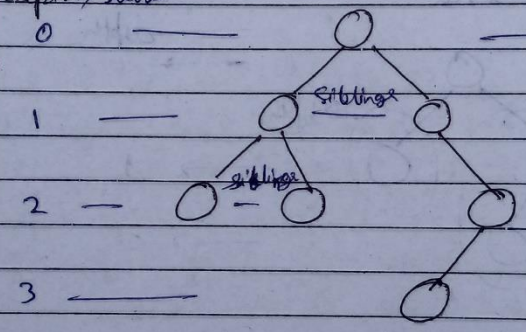
Note- For all Nodes in a BST-

(i) all nodes in left subtree ≤ parent-node
(ii) all nodes in right subtree ≥ parent-node



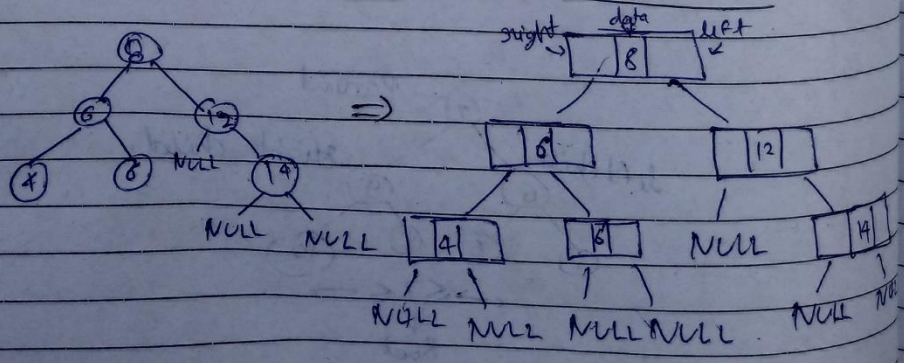
Page No.: 150
Date: / /

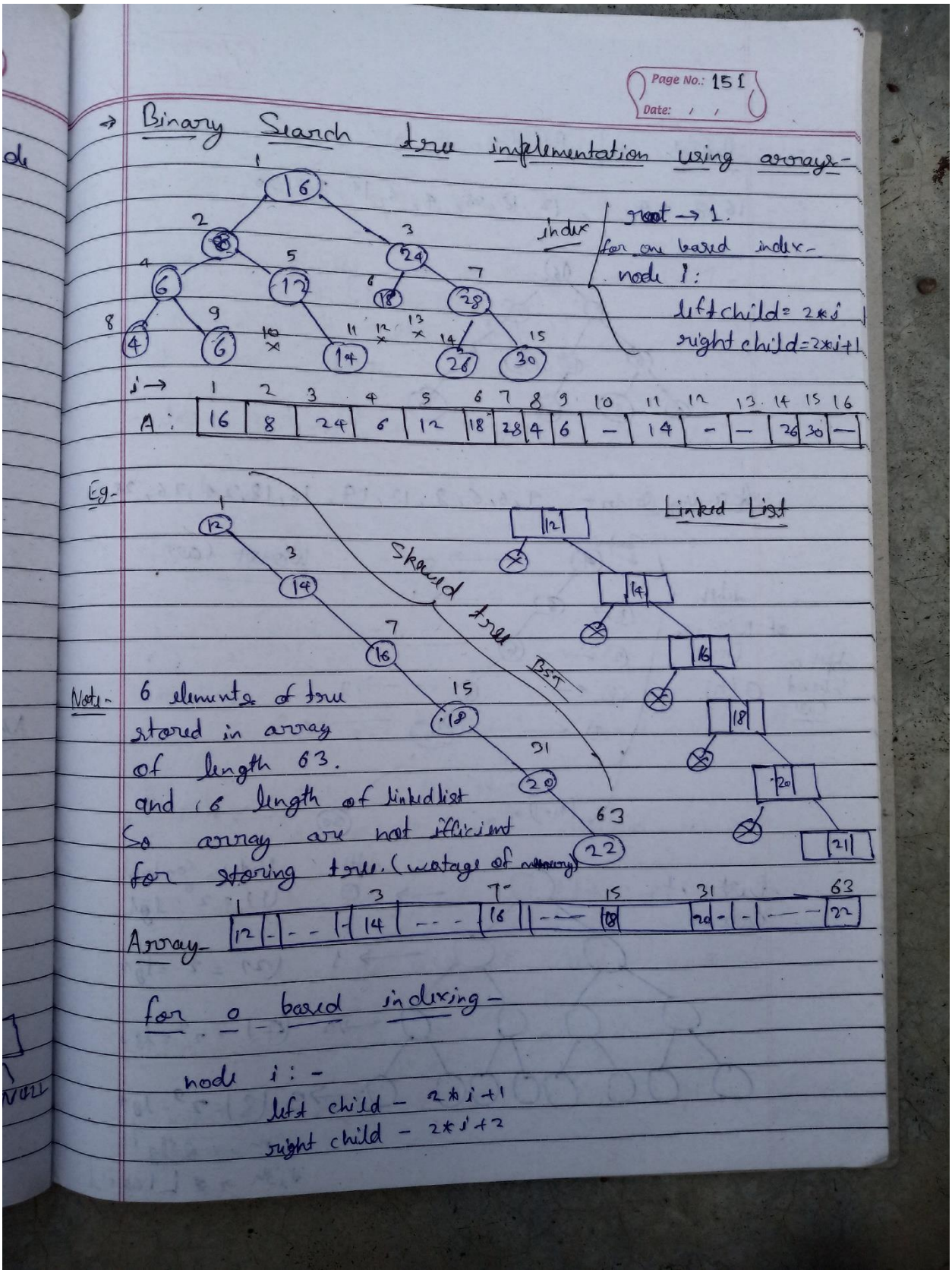
- depth - The depth of a node is the number of edges from the node to the tree's root node.
- hight - The height of node is the number of edges on the longest path from the node to a leaf. A leaf node will have a height 0.
- depth/level Height



degree - The degree of a node is the number of children of the node.

→ Binary Search Tree Implimentation-

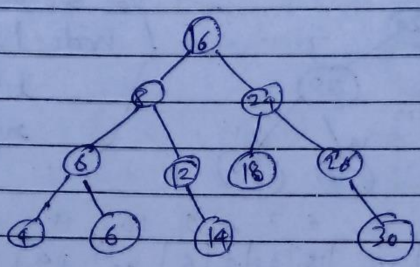



Page No.: 151
Date: / /
→ Binary Search tree implementation using arrays-
root → 1.
for one based index-
node i:
left child = 2*i
right child = 2*i+1
i → 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
A: 16 8 24 6 12 18 28 4 6 - 14 - - 26 30 -
Eg-
Linked List
Skewed tree
BST
Note- 6 elements of tree stored in array of length 63.
and 6 length of linkedlist
So array are not efficient for storing tree. (wastage of memory)
Array-
for 0 based indexing-
node i:-
left child - 2*i+1
right child - 2*i+2

Page No.: 152
Date: / /

Build a BST-

16, 8, 24, 6, 12, 18, 28, 4, 6, 14, 26, 30

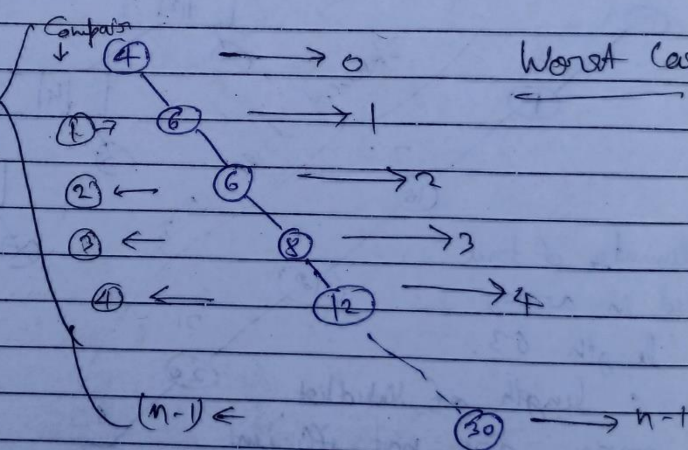


Ascending Order - 4, 6, 6, 8, 12, 14, 16, 18, 24, 26, 28, 30

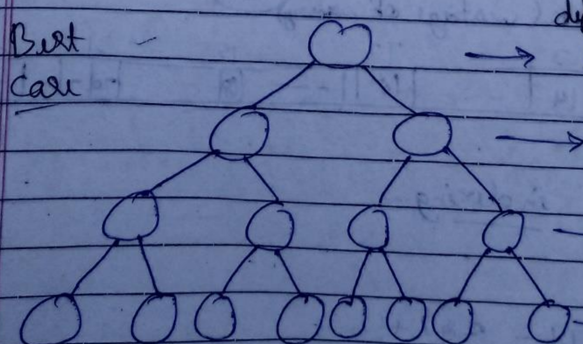
Worst Case

depth of BST is (n-1)

depth in Worst Case - $\Theta(n)$



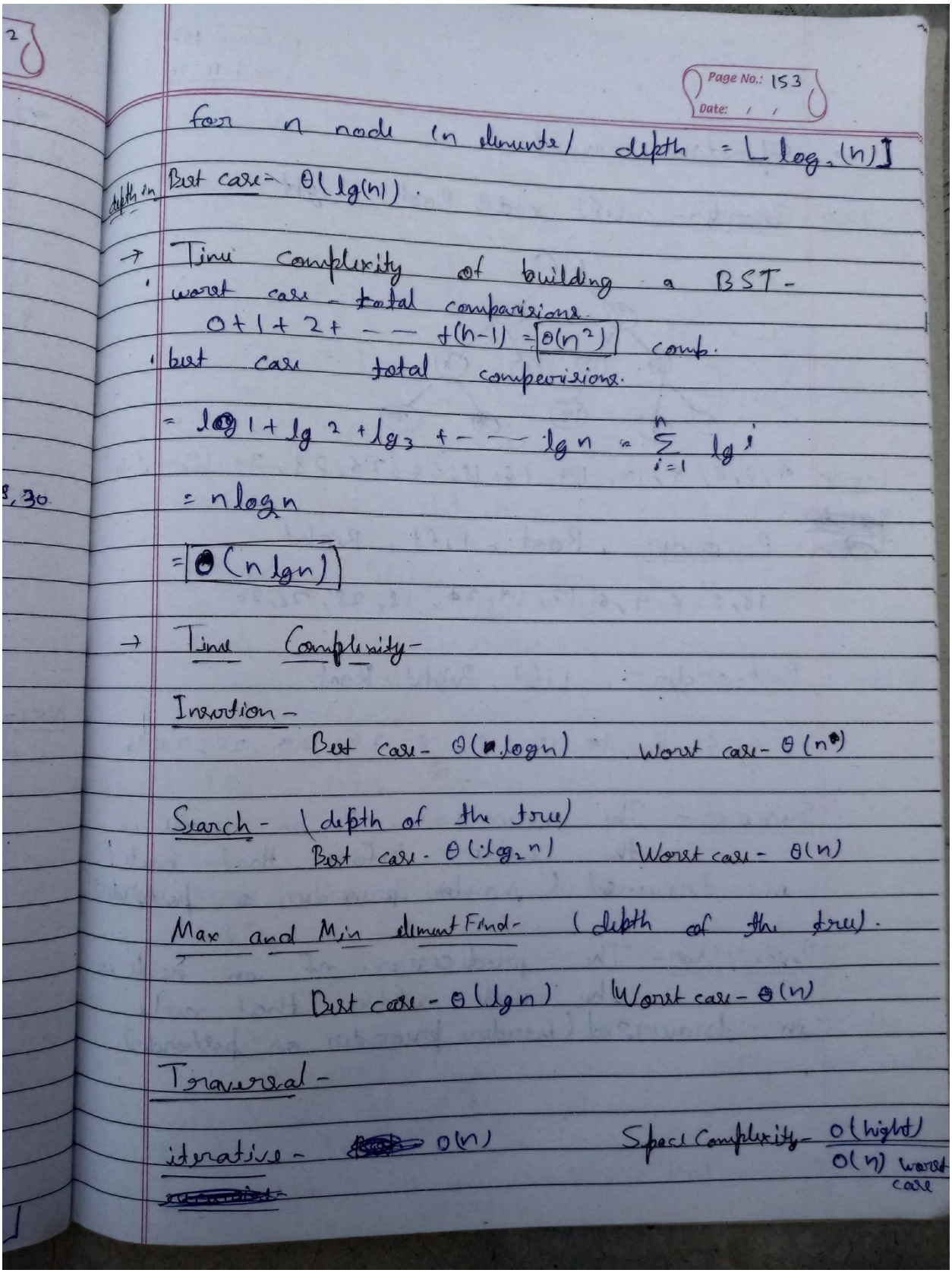
Best Case



depth	Nodes	Comparison
→ 0	(1) = 2^0	= lg 1
→ 1	(2) = 2^1	= lg 2
→ 2	(4) = 2^2	= lg 3
→ 3	(8) = 2^3	= lg 4

ith → Θ lg i

ith → $\lfloor \log_2 i \rfloor$



Page No.: 153
Date: / /

for n node (n elements) / depth = $\lfloor \log_2(n) \rfloor$

depth in Best case - $\Theta(\lg(n))$.

→ Time complexity of building a BST -
worst case - total comparisions.
$0+1+2+\cdots+(n-1) = \boxed{\Theta(n^2)}$ comp.
best case total comperisions.

$= \lg 1 + \lg 2 + \lg 3 + \cdots \lg n = \sum_{i=1}^{n} \lg i$

$= n \log n$

$= \boxed{\Theta(n \lg n)}$

→ Time Complexity -

Insertion -
Best case - $\Theta(\log n)$ Worst case - $\Theta(n)$

Search - (depth of the tree)
Best case - $\Theta(\log_2 n)$ Worst case - $\Theta(n)$

Max and Min element Find - (depth of the tree).
Best case - $\Theta(\lg n)$ Worst case - $\Theta(n)$

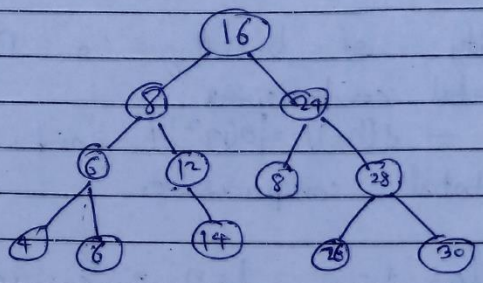
Traversal -

iterative - $O(n)$ Space Complexity - $O(\text{height})$ / $O(n)$ worst case

Page No.: 154
Date: 7/11/20

→ Operations - Traversal -

- Inorder - Left node, Root, Right



Inorder - 4, 6, 6, 8, 12, 14, 16, 18, 24, 26, 28, 30 (Sorted list)

- Preorder - Root, Left, Right

16, 8, 6, 4, 6, 12, 14, 24, 18, 28, 26, 30

- Post-order - Left, Right, Root

4, 6, 6, 14, 12, 8, 8, 26, 30, 28, 24, 16

- Successor - The successor of an node is the node before that node in traversal (inorder, preorder, or postorder)

- Predecessor - The predecessor of an node is the node after that node in traversal (inorder, preorder or postorder)

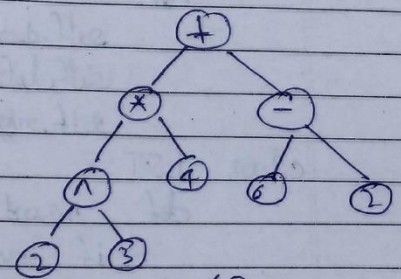
Page No.: 155
Date: / /

Application-

Expression Tree - Binary tree but Not BST.

leaf nodes - Operands
non-leaf nodes - Operators

Eg- ((2^3)*4)+(6-2))



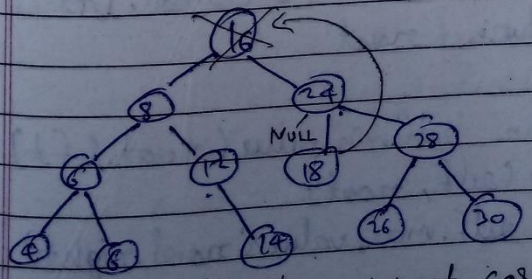
1- Inorder - infix expression
2- Preorder - prefix expression → + * ^ 2 3 4 - 6 2
3- Postorder - postfix expression → 2 3 ^ 4 * 6 2 - +

→ Operation - Delete - a pointer to the node that I want to delete.

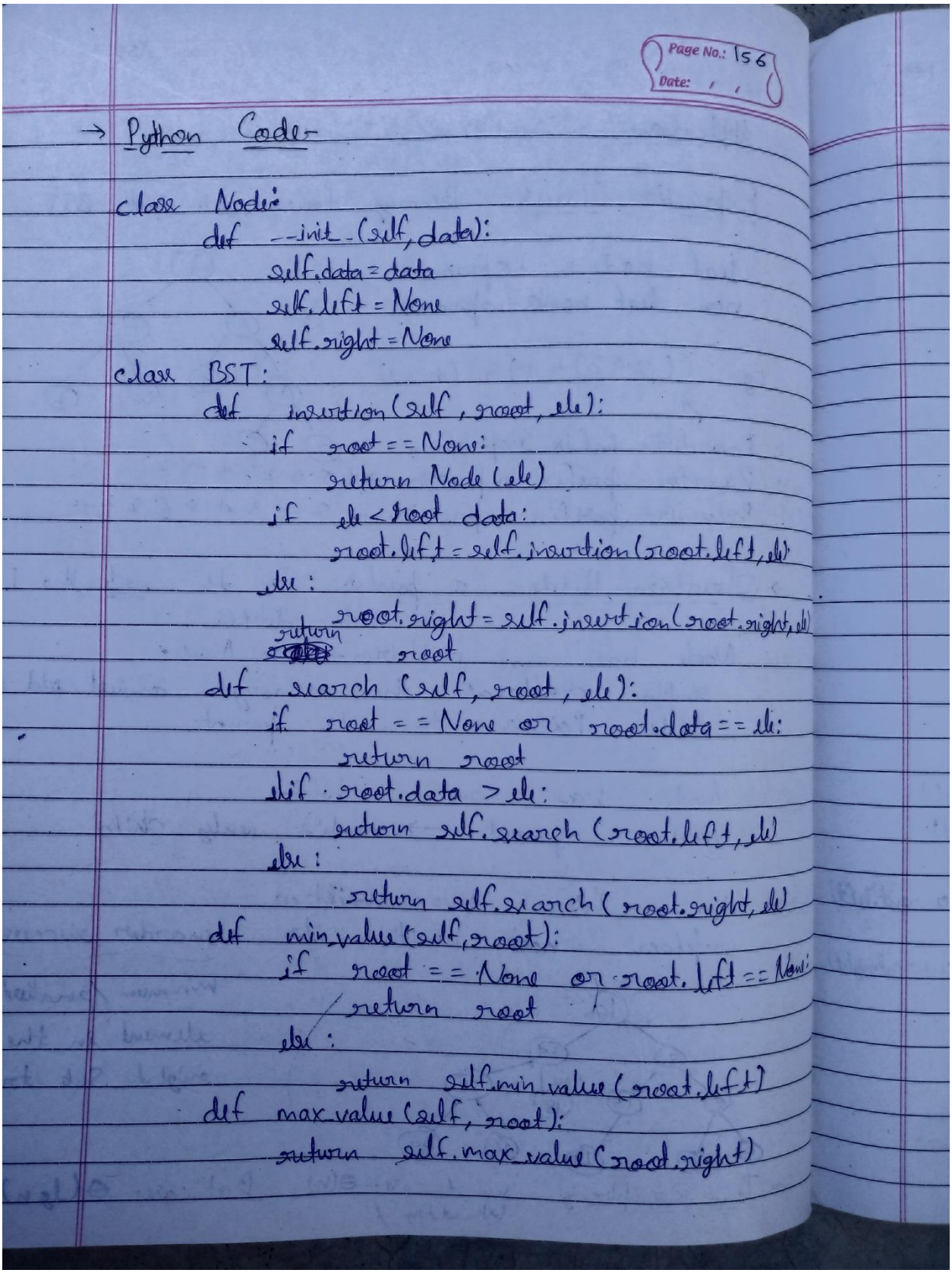
Case1- Node has no children → Leaf Node -
simply deallocate the memory & adjust/add the Null Pointer to its parent.

Case2- Node has one child -
Node's parent → node's only child

Case3- Node has two children -
replace node with its inorder successor
minimum/smallest element in the right Sub tree

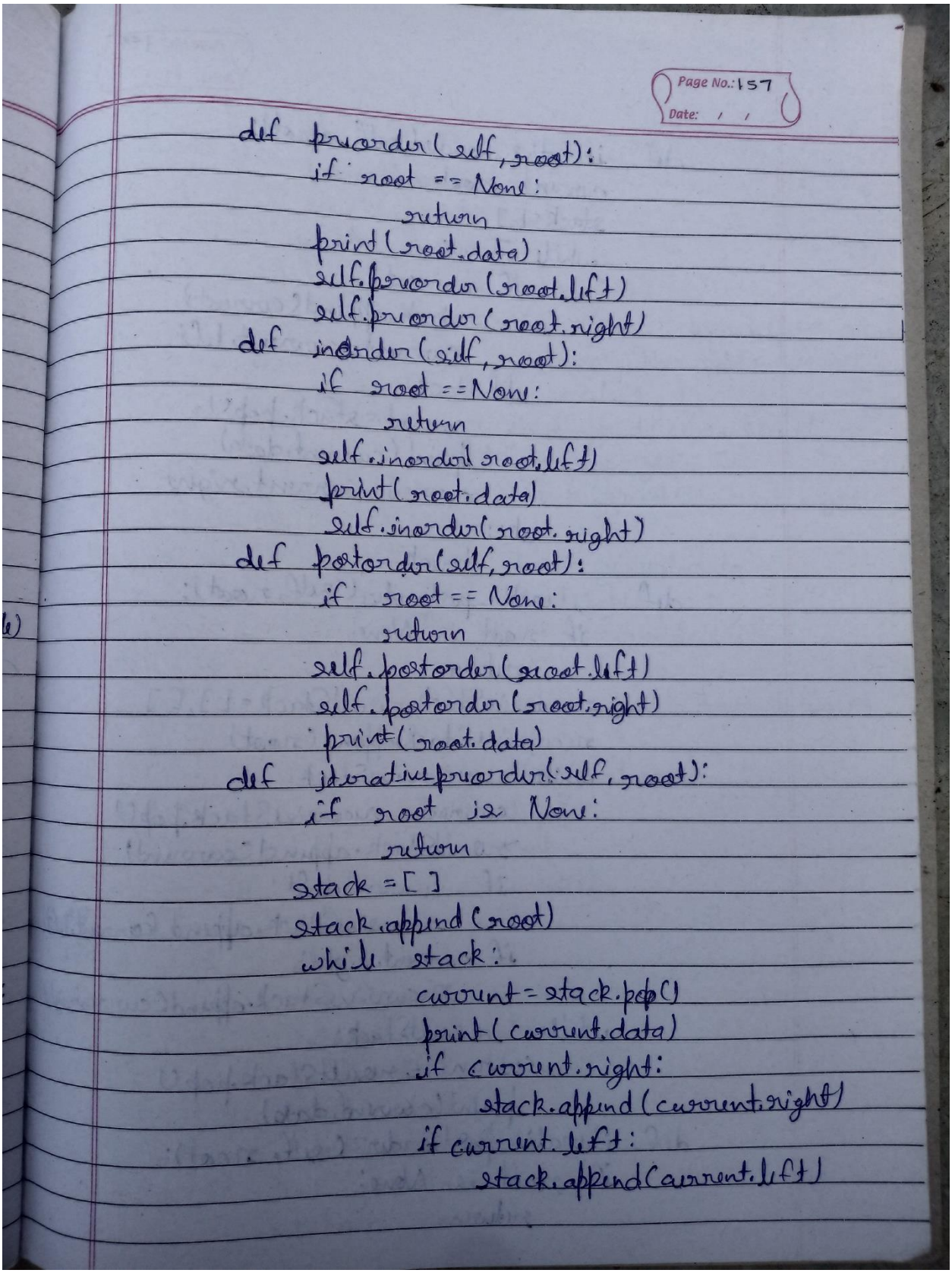


→ Time complexity - Worst case - $\Theta(n)$ Best case - $\Theta(\lg n)$
(skewed tree)

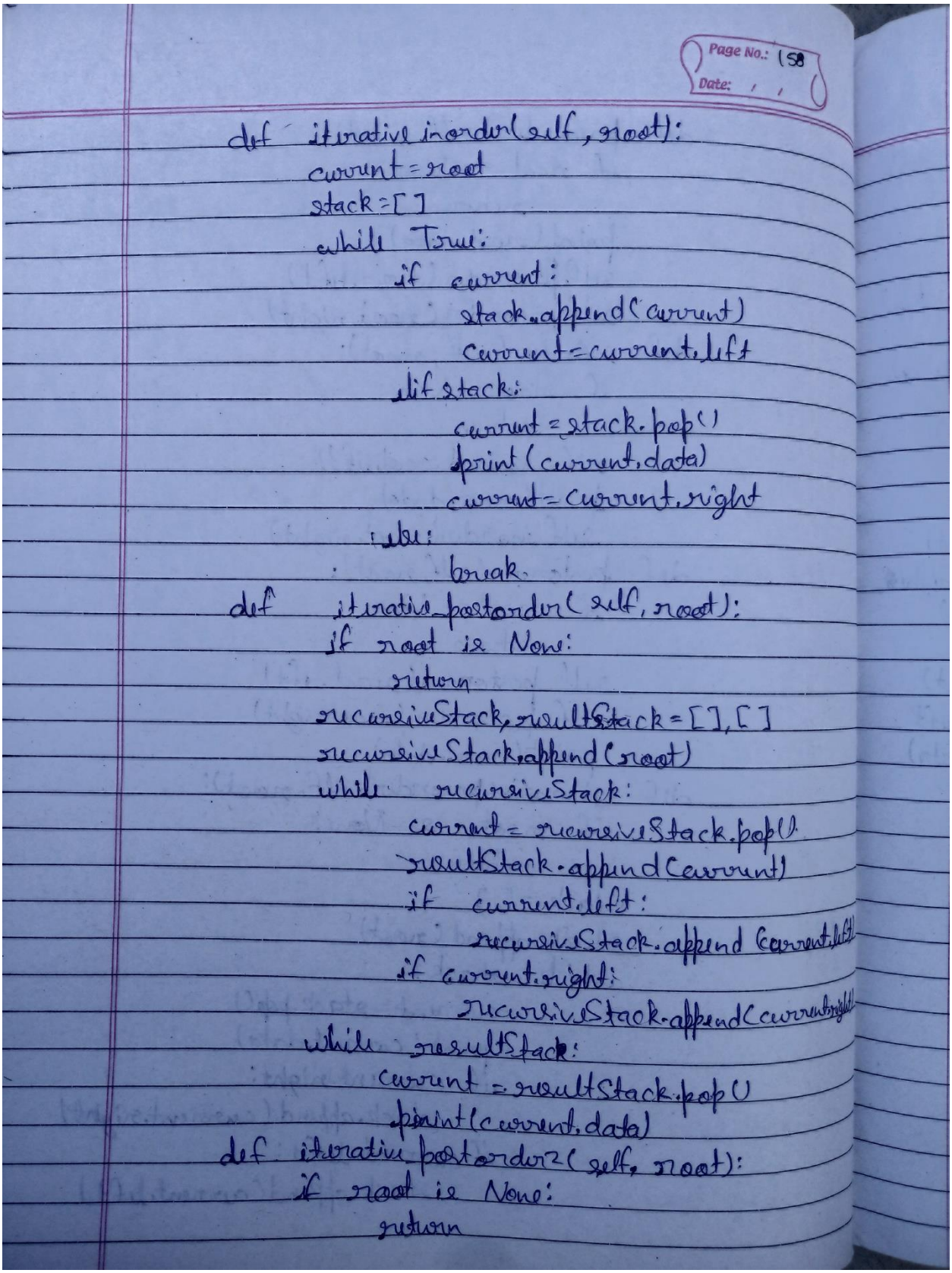


→ Python Code-

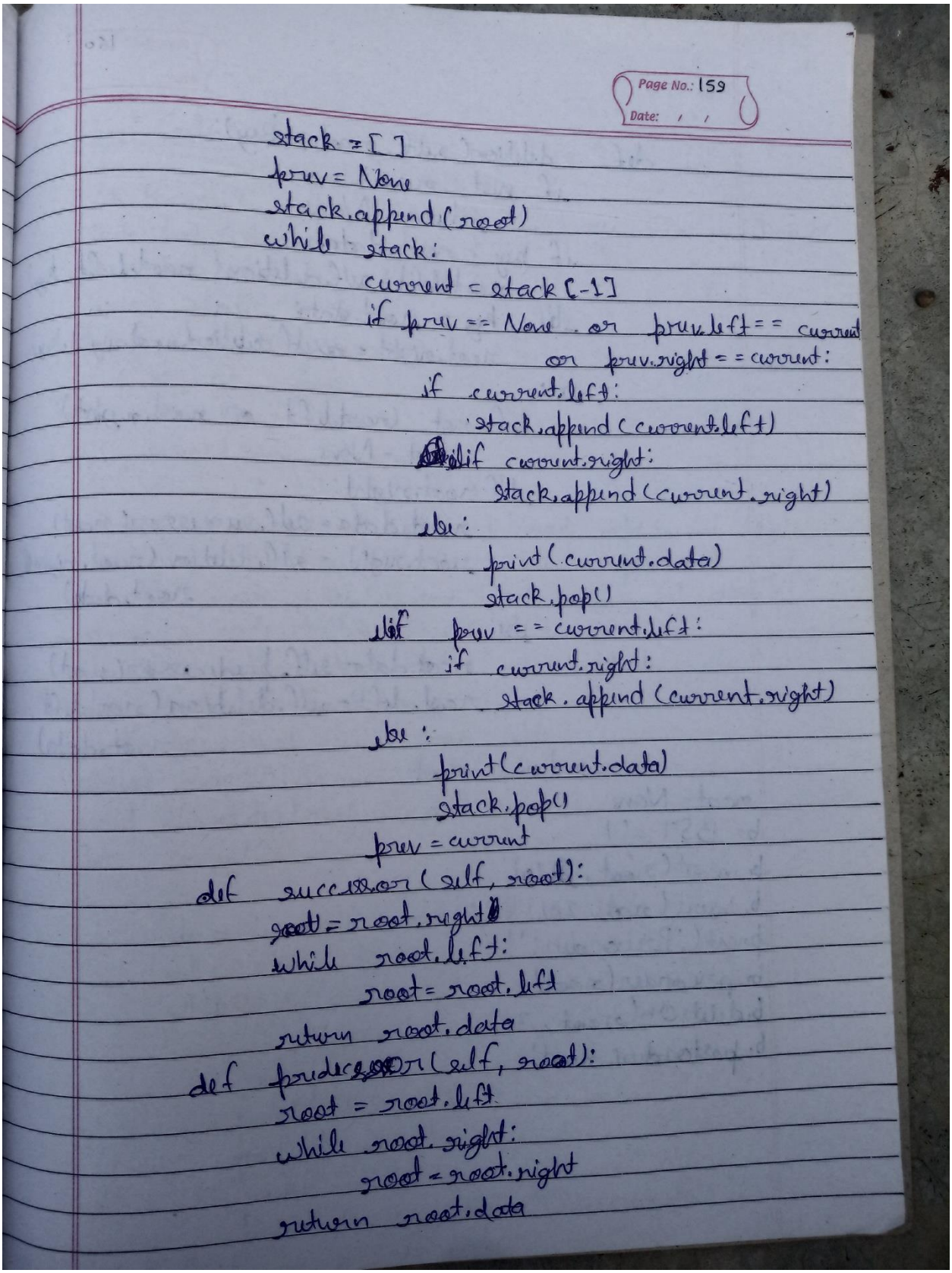
```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
class BST:
    def insertion(self, root, ele):
        if root == None:
            return Node(ele)
        if ele < root.data:
            root.left = self.insertion(root.left, ele)
        else:
            root.right = self.insertion(root.right, ele)
        return root
    def search(self, root, ele):
        if root == None or root.data == ele:
            return root
        elif root.data > ele:
            return self.search(root.left, ele)
        else:
            return self.search(root.right, ele)
    def min_value(self, root):
        if root == None or root.left == None:
            return root
        else:
            return self.min_value(root.left)
    def max_value(self, root):
        return self.max_value(root.right)
```



```
Page No.: 157
Date: / /

def preorder(self, root):
    if root == None:
        return
    print(root.data)
    self.preorder(root.left)
    self.preorder(root.right)
def inorder(self, root):
    if root == None:
        return
    self.inorder(root.left)
    print(root.data)
    self.inorder(root.right)
def postorder(self, root):
    if root == None:
        return
    self.postorder(root.left)
    self.postorder(root.right)
    print(root.data)
def iterativepreorder(self, root):
    if root is None:
        return
    stack = []
    stack.append(root)
    while stack:
        current = stack.pop()
        print(current.data)
        if current.right:
            stack.append(current.right)
        if current.left:
            stack.append(current.left)
```

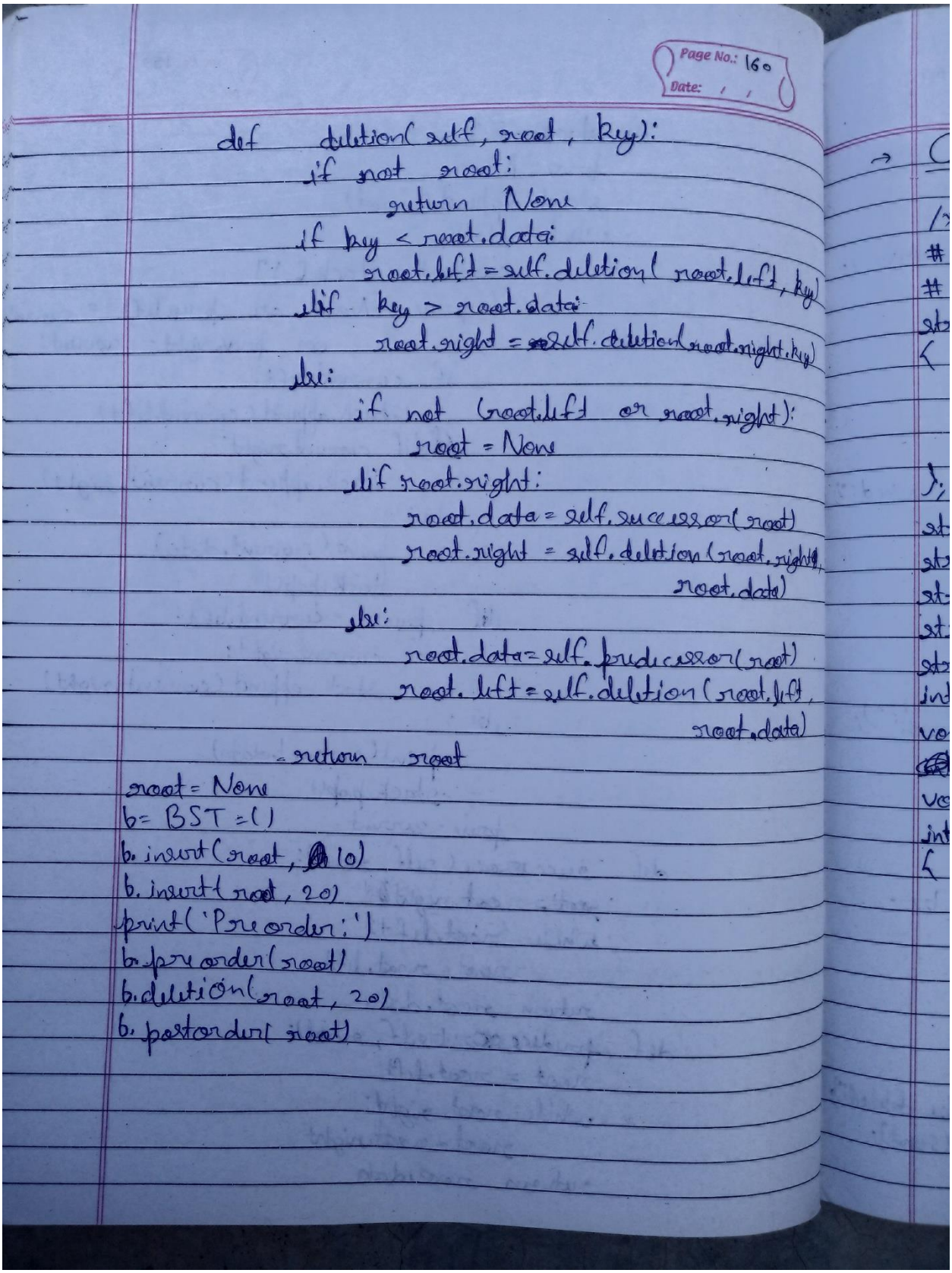


```
Page No.: 58
Date:

def iterative inorder(self, root):
    current = root
    stack = []
    while True:
        if current:
            stack.append(current)
            current = current.left
        elif stack:
            current = stack.pop()
            print(current.data)
            current = current.right
        else:
            break
def iterative_postorder(self, root):
    if root is None:
        return
    recursiveStack, resultStack = [], []
    recursiveStack.append(root)
    while recursiveStack:
        current = recursiveStack.pop()
        resultStack.append(current)
        if current.left:
            recursiveStack.append(current.left)
        if current.right:
            recursiveStack.append(current.right)
    while resultStack:
        current = resultStack.pop()
        print(current.data)
def iterativePostorder2(self, root):
    if root is None:
        return
```

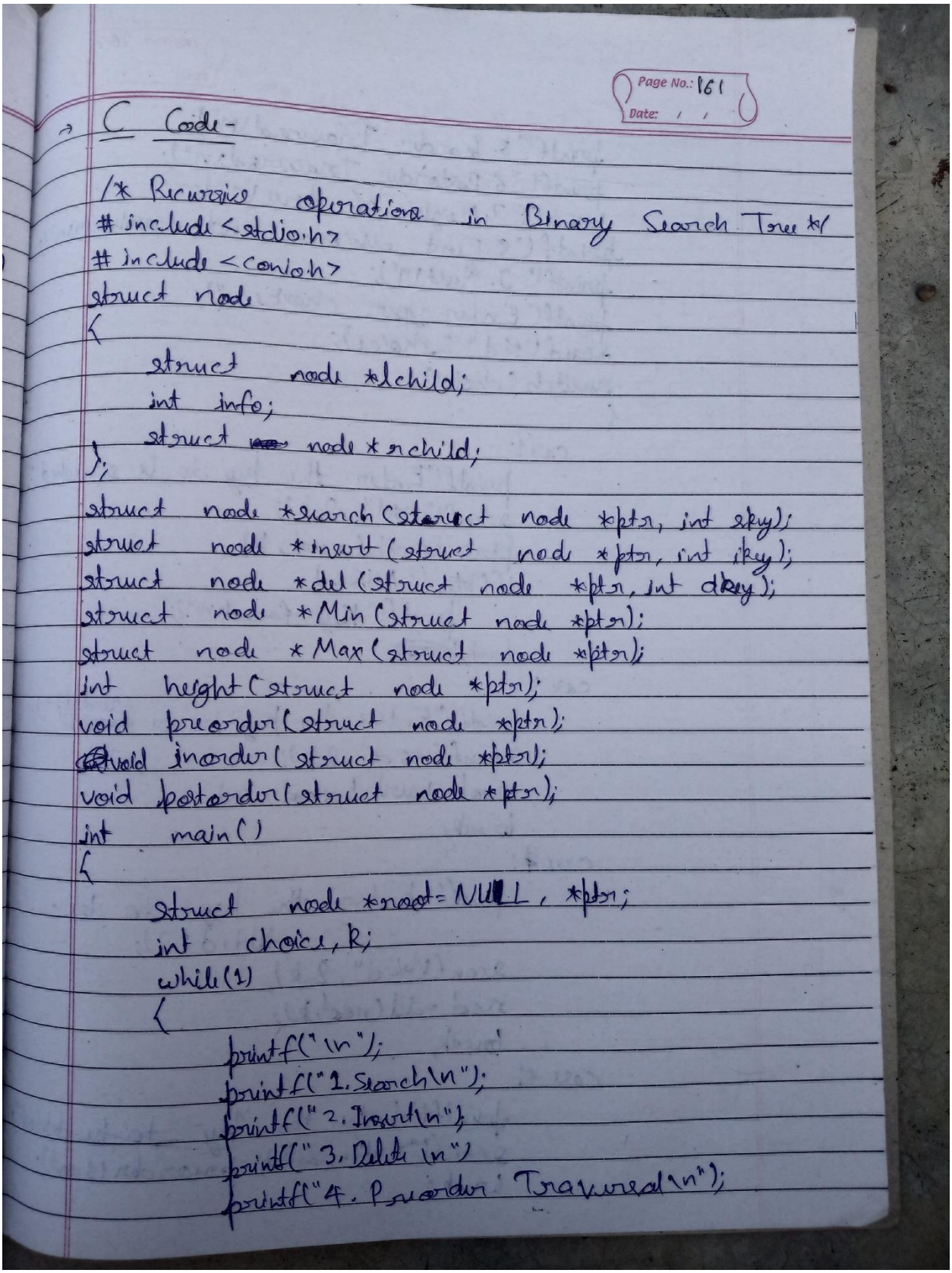


Page No.: 159
Date: / /

```
stack = [ ]
prev = None
stack.append(root)
while stack:
    current = stack[-1]
    if prev == None or prev.left == current
                    or prev.right == current:
        if current.left:
            stack.append(current.left)
        elif current.right:
            stack.append(current.right)
        else:
            print(current.data)
            stack.pop()
    elif prev == current.left:
        if current.right:
            stack.append(current.right)
    else:
        print(current.data)
        stack.pop()
    prev = current
def successor(self, root):
    root = root.right
    while root.left:
        root = root.left
    return root.data
def predecessor(self, root):
    root = root.left
    while root.right:
        root = root.right
    return root.data
```



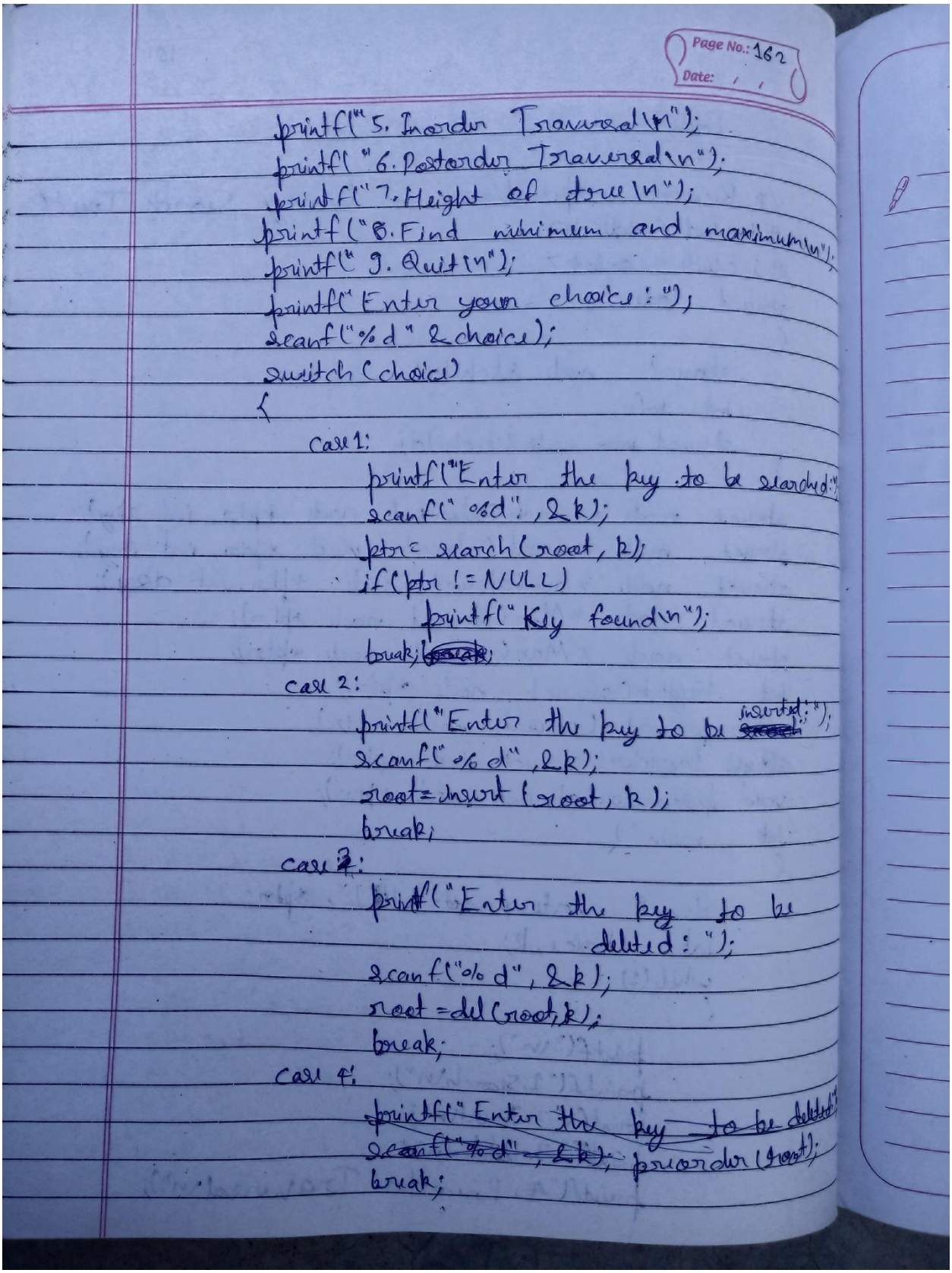
```
def deletion(self, root, key):
    if not root:
        return None
    if key < root.data:
        root.left = self.deletion(root.left, key)
    elif key > root.data:
        root.right = self.deletion(root.right, key)
    else:
        if not (root.left or root.right):
            root = None
        elif root.right:
            root.data = self.successor(root)
            root.right = self.deletion(root.right,
                                       root.data)
        else:
            root.data = self.predecessor(root)
            root.left = self.deletion(root.left,
                                      root.data)
    return root
root = None
b = BST()
b.insert(root, 10)
b.insert(root, 20)
print('Preorder:')
b.preorder(root)
b.deletion(root, 20)
b.postorder(root)
```



Page No.: 161
Date: / /

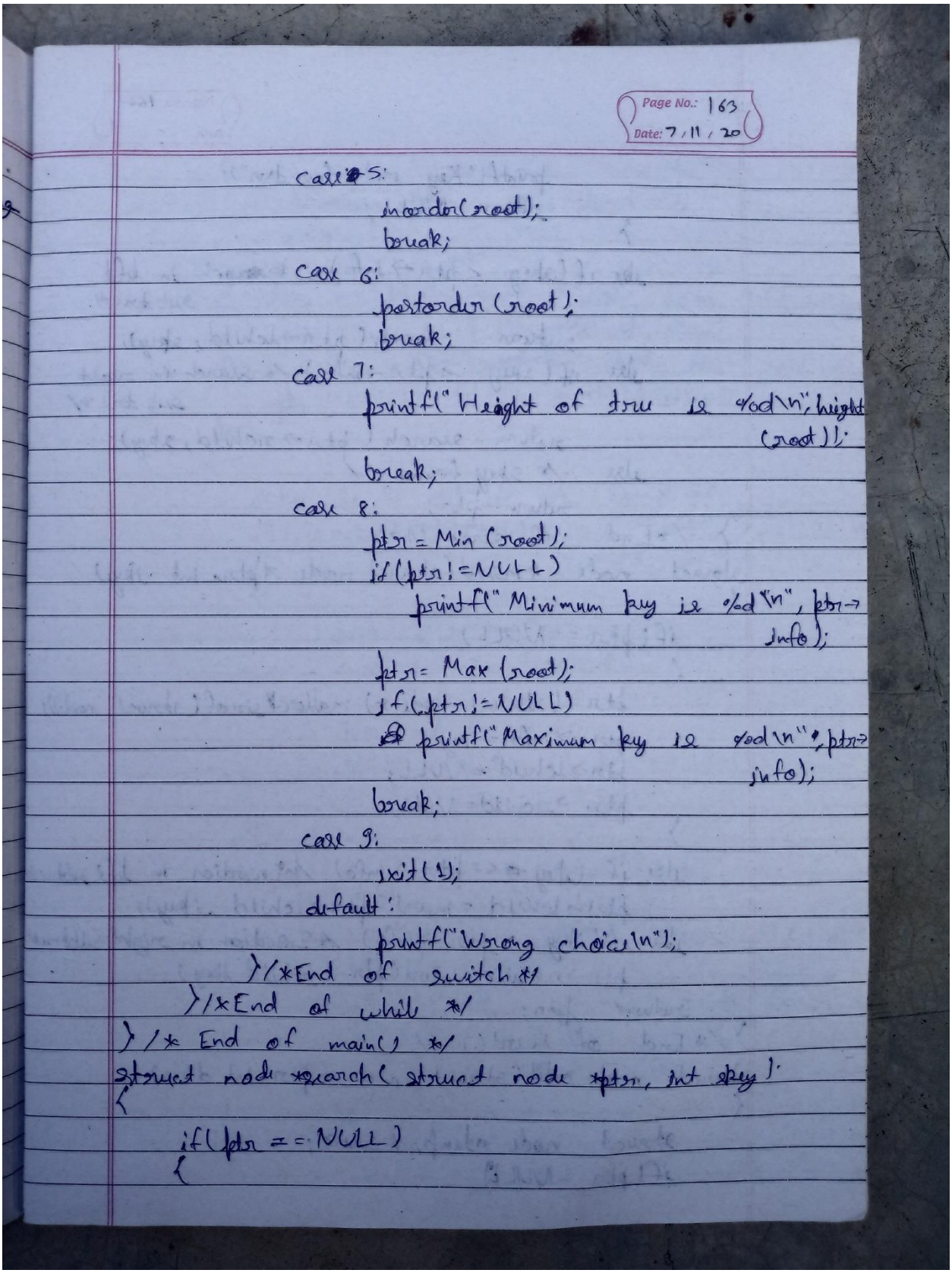
→ C Code -

```
/* Recursive operations in Binary Search Tree */
#include <stdio.h>
#include <conio.h>
struct node
{
    struct node *lchild;
    int info;
    struct node *rchild;
};
struct node *search(struct node *ptr, int skey);
struct node *insert(struct node *ptr, int ikey);
struct node *del(struct node *ptr, int dkey);
struct node *Min(struct node *ptr);
struct node *Max(struct node *ptr);
int height(struct node *ptr);
void preorder(struct node *ptr);
void inorder(struct node *ptr);
void postorder(struct node *ptr);
int main()
{
    struct node *root=NULL, *ptr;
    int choice, k;
    while(1)
    {
        printf("\n");
        printf("1. Search\n");
        printf("2. Insert\n");
        printf("3. Delete\n")
        printf("4. Preorder Traversal\n");
```

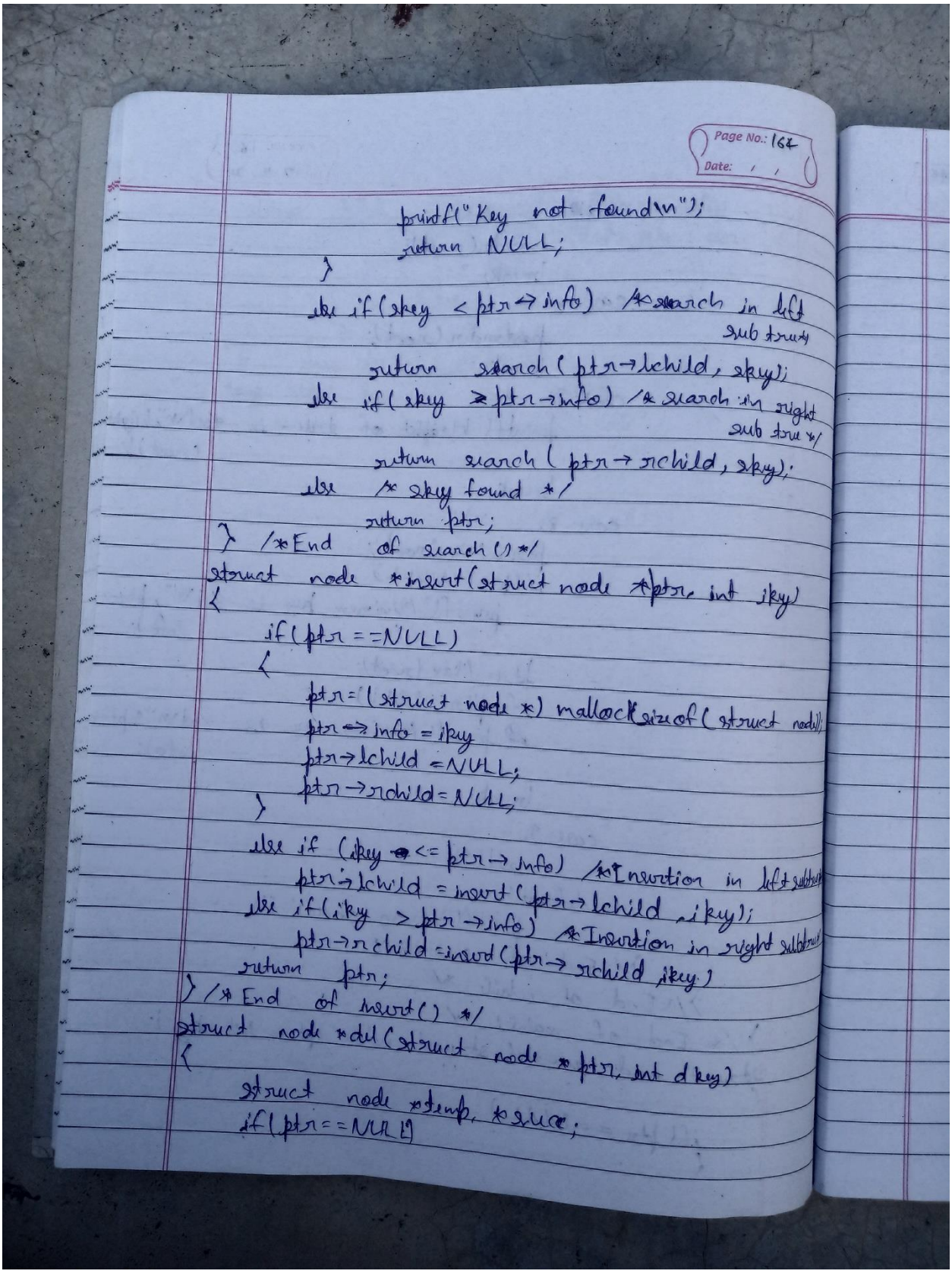


Page No.: 162
Date: / /

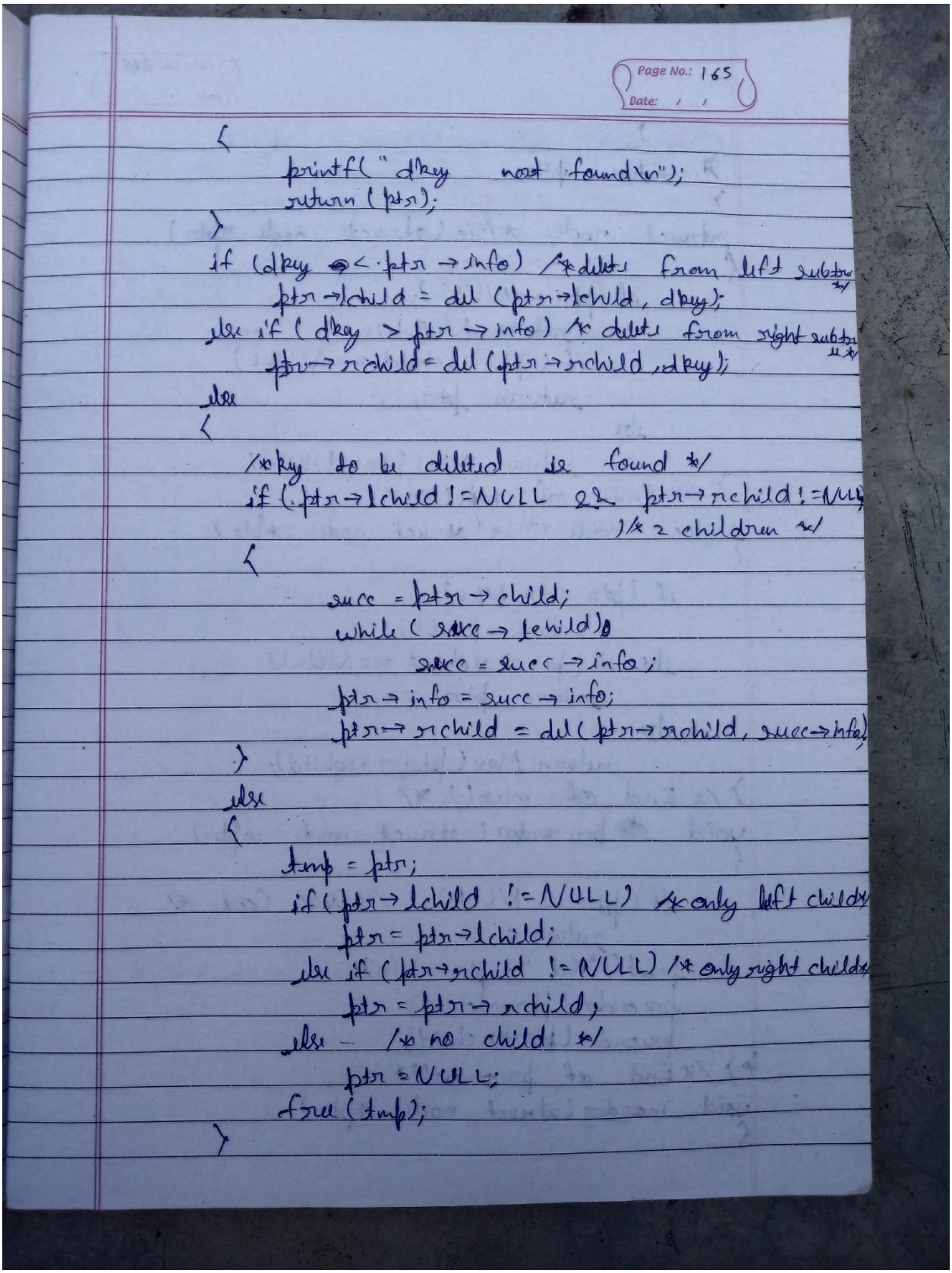
```
printf("5. Inorder Traversal\n");
printf("6. Postorder Traversal\n");
printf("7. Height of tree\n");
printf("8. Find minimum and maximum\n");
printf("9. Quit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch(choice)
{
    case 1:
        printf("Enter the key to be searched: ");
        scanf("%d", &k);
        ptr = search(root, k);
        if(ptr != NULL)
            printf("Key found\n");
        break;
    case 2:
        printf("Enter the key to be inserted: ");
        scanf("%d", &k);
        root = insert(root, k);
        break;
    case 3:
        printf("Enter the key to be deleted: ");
        scanf("%d", &k);
        root = del(root, k);
        break;
    case 4:
        printf("Enter the key to be deleted: ");
        scanf("%d", &k); preorder(root);
        break;
```



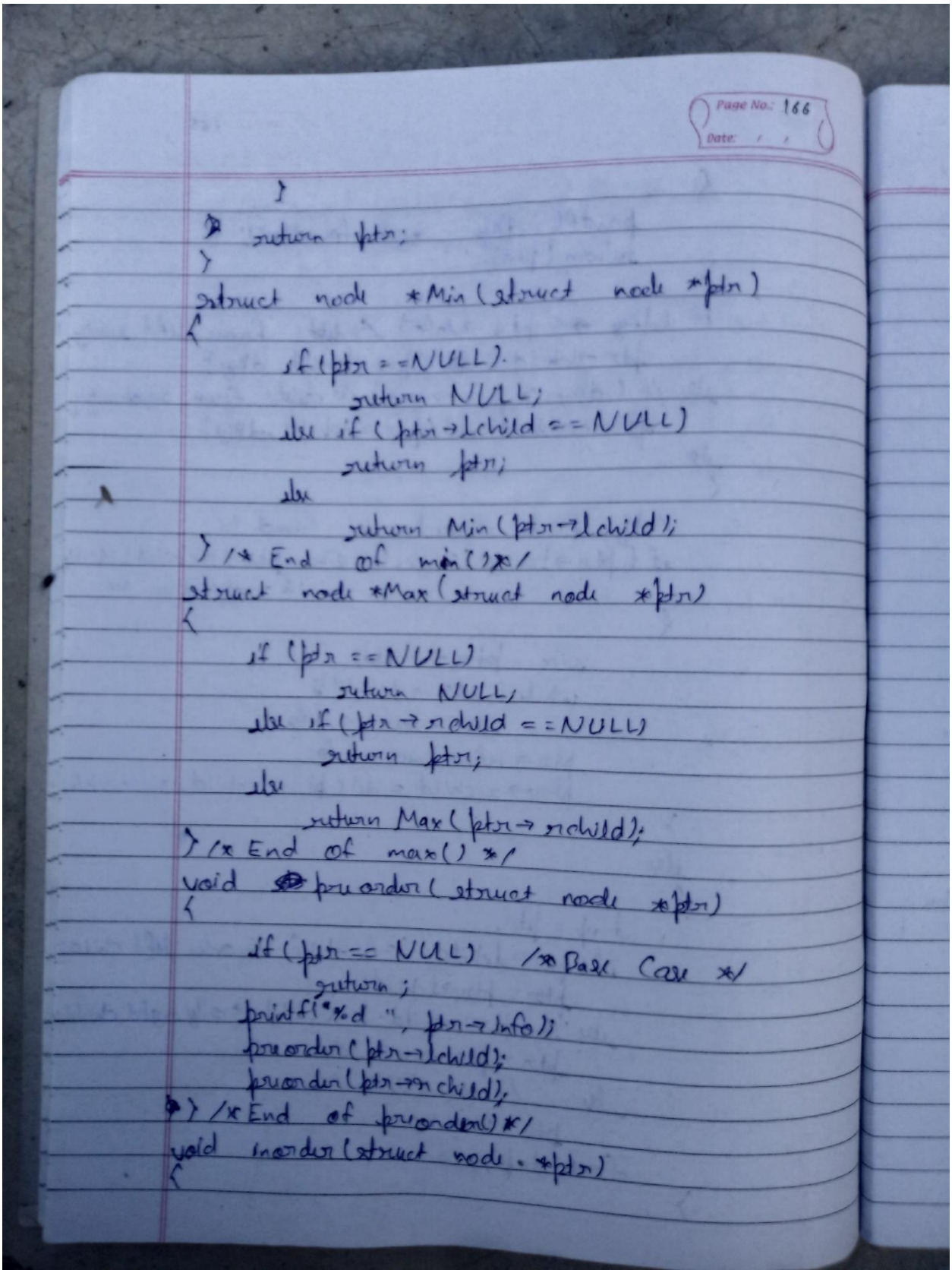
```
Page No.: 163
Date: 7/11/20

        case 5:
                inorder(root);
                break;
        case 6:
                postorder(root);
                break;
        case 7:
                printf("Height of tree is %d\n", height
                                                (root));
                break;
        case 8:
                ptr = Min(root);
                if(ptr != NULL)
                    printf("Minimum key is %d\n", ptr->
                                                info);
                ptr = Max(root);
                if(ptr != NULL)
                    printf("Maximum key is %d\n", ptr->
                                                info);
                break;
        case 9:
                exit(1);
        default:
                printf("Wrong choice\n");
            }/*End of switch*/
        }/*End of while*/
}/* End of main() */
struct node *search(struct node *ptr, int key)
{
    if(ptr == NULL)
    {
```



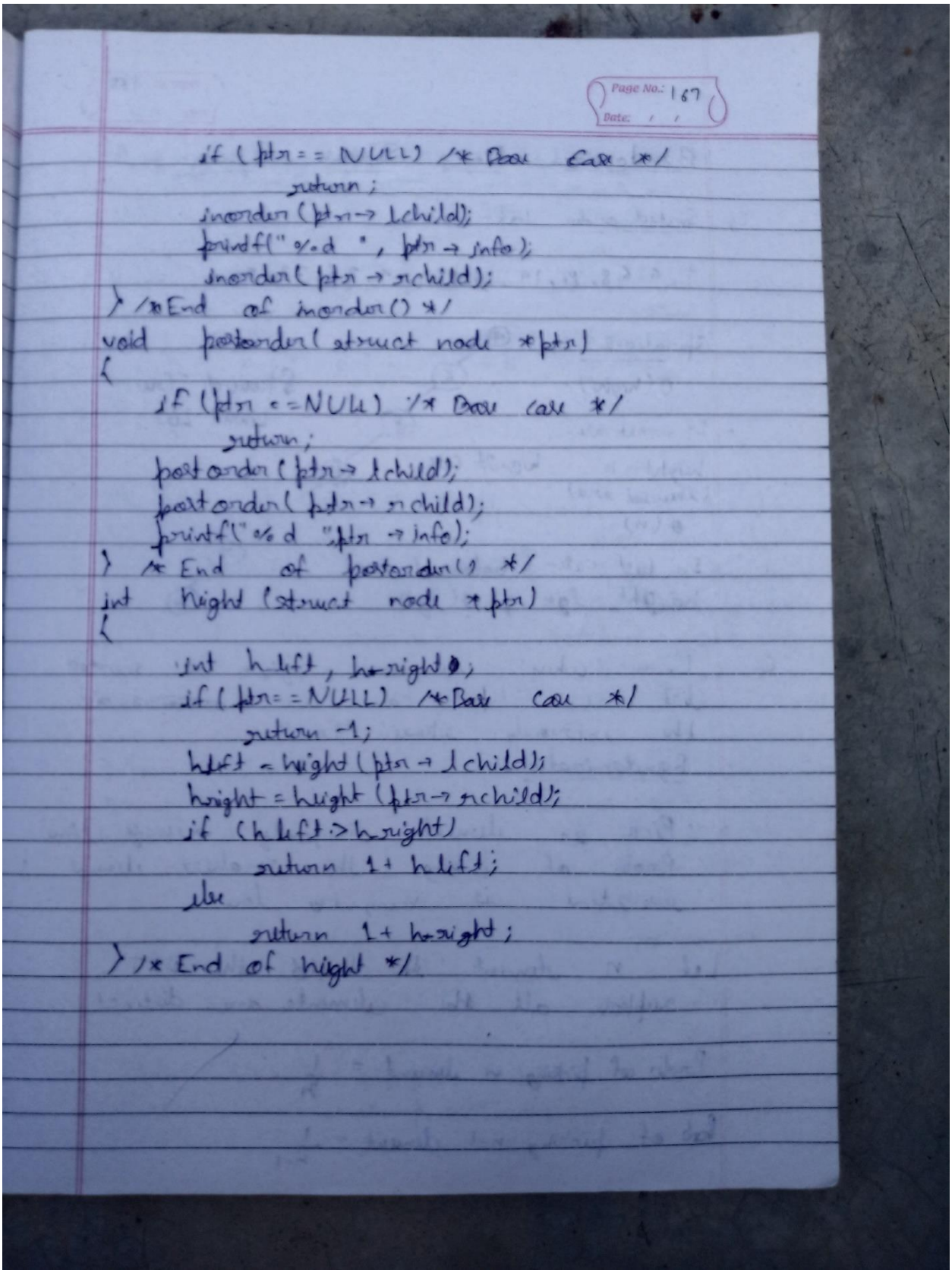
```
Page No.: 164
Date: / /

        printf("Key not found\n");
        return NULL;
    }
    else if(skey < ptr->info) /*search in left
                                sub tree*/
        return search(ptr->lchild, skey);
    else if(skey > ptr->info) /* search in right
                                sub tree */
        return search(ptr->rchild, skey);
    else /* skey found */
        return ptr;
} /*End of search()*/
struct node *insert(struct node *ptr, int ikey)
{
    if(ptr==NULL)
    {
        ptr=(struct node *) malloc(sizeof(struct node));
        ptr->info = ikey;
        ptr->lchild = NULL;
        ptr->rchild = NULL;
    }
    else if (ikey <= ptr->info) /*Insertion in left subtree*/
        ptr->lchild = insert(ptr->lchild, ikey);
    else if(ikey > ptr->info) /*Insertion in right subtree*/
        ptr->rchild = insert(ptr->rchild, ikey);
    return ptr;
} /* End of insert() */
struct node *del(struct node *ptr, int dkey)
{
    struct node *temp, *succ;
    if(ptr==NULL)
```



```
{
    printf("dkey not found\n");
    return(ptr);
}
if (dkey < ptr->info) /* delete from left subtree */
    ptr->lchild = del(ptr->lchild, dkey);
else if (dkey > ptr->info) /* delete from right subtree */
    ptr->rchild = del(ptr->rchild, dkey);
else
{
    /* key to be deleted is found */
    if (ptr->lchild != NULL && ptr->rchild != NULL)
                                /* 2 children */
    {
        succ = ptr->rchild;
        while (succ->lchild)
            succ = succ->info;
        ptr->info = succ->info;
        ptr->rchild = del(ptr->rchild, succ->info);
    }
    else
    {
        tmp = ptr;
        if (ptr->lchild != NULL) /* only left child */
            ptr = ptr->lchild;
        else if (ptr->rchild != NULL) /* only right child */
            ptr = ptr->rchild;
        else /* no child */
            ptr = NULL;
        free(tmp);
    }
```



```
        }
    return ptr;
}
struct node *Min(struct node *ptr)
{
    if(ptr==NULL)
        return NULL;
    else if(ptr->lchild==NULL)
        return ptr;
    else
        return Min(ptr->lchild);
} /* End of min() */
struct node *Max(struct node *ptr)
{
    if(ptr==NULL)
        return NULL;
    else if(ptr->rchild==NULL)
        return ptr;
    else
        return Max(ptr->rchild);
} /* End of max() */
void preorder(struct node *ptr)
{
    if(ptr==NULL)  /* Base Case */
        return;
    printf("%d ", ptr->info);
    preorder(ptr->lchild);
    preorder(ptr->rchild);
} /* End of preorder() */
void inorder(struct node *ptr)
{
```



```
Page No.: 167
Date:

    if (ptr == NULL) /* Base Case */
        return;
    inorder(ptr->lchild);
    printf("%d ", ptr->info);
    inorder(ptr->rchild);
} /* End of inorder() */
void postorder(struct node *ptr)
{
    if (ptr == NULL) /* Base case */
        return;
    postorder(ptr->lchild);
    postorder(ptr->rchild);
    printf("%d ", ptr->info);
} /* End of postorder() */
int height(struct node *ptr)
{
    int hleft, hright;
    if (ptr == NULL) /* Base case */
        return -1;
    hleft = height(ptr->lchild);
    hright = height(ptr->rchild);
    if (hleft > hright)
        return 1 + hleft;
    else
        return 1 + hright;
} /* End of height */
```

Page No.: 168
Date: / /

Randomized Binary Search Trees-

Eg Sorted order list-

4, 6, 6, 8, 12, 14, 16, 18, 24, 26, 28, 30

Operations -

Θ(height).

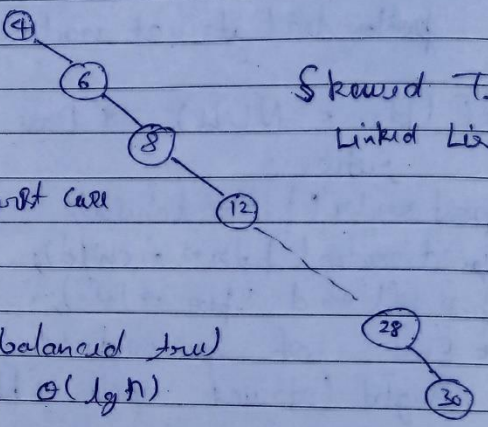
- In worst case -

height = n (skewed tree)

$\Theta(n)$

- In best case - (balanced tree)

height = lgn, $\Theta(\lg n)$



Skewed Tree
Linked List
Worst Case

Q- Even when we are given a sorted list as input, can we avoid the extremely skewed Tree?

Randomization -

- Pick an element randomly every time.
- Prob. of picking the smallest element every time is very very low.

Let n element to build the BST suppose all the elements are distinct

Prob. of picking n element = $\frac{1}{n}$

Prob. of picking n-1 element = $\frac{1}{n-1}$

Q-

(i)
(ii)
(iii)
(iv)

Page No.: 169
Date: / /

Prob. of picking $n-2$ element $= \frac{1}{n-2}$

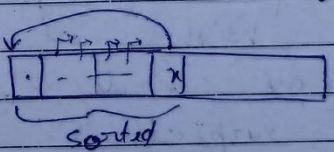
prob. of picking smallest element every time

$= \frac{1}{n} * \frac{1}{n-1} * \frac{1}{n-2} \cdots - 1$

$= \frac{1}{n!}$

Q- The unusual $\theta(n^2)$ implementation of Insertion Sort to sort an array use linear search to identify the position where an element is to be inserted into the already sorted part of the array. If ~~inserted~~ instead we use binary search to identify the position, the worst case running time will

(i) remain $\theta(n^2)$ (✓)
(ii) become $\theta(n(\log n)^2)$
(iii) become $\theta(n \log n)$
(iv) become $\theta(n)$



Linear search -
swap - $\theta(k)$
compa $\theta(k)$

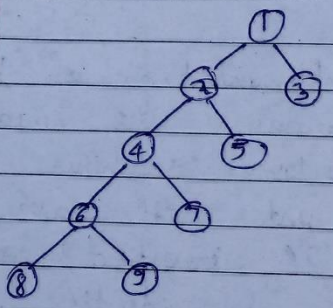
binary Search to find the position of x -
Linear search → $\theta(k)$
Binary Search → $\theta(\lg k)$ $k \leq (n-1)$

Total time complexity - $\theta(n^2)$
= Comp &, Swaps
= $\theta(\lg k)$, $\theta(k)$

Page No.: 170
Date: / /

Q- The postorder traversal of a binary tree is 8, 9, 6, 7, 4, 5, 2, 3, 1. The inorder traversal of the same tree is, 8, 6, 9, 4, 7, 2, 5, 1, 3. The height of a tree is the length of the longest path from root to any leaf. The height of the binary of the binary tree above is (4)

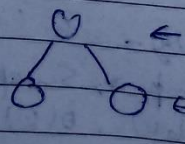
Post order- L, R, Rt, Inorder- L, Rt, R



height = 4.

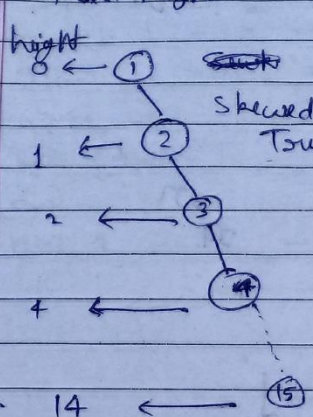
Q- Let T be a binary search tree with 15 nodes. The minimum and maximum poible heights of T are-
Note- The height of a tree with a single node is 0.

(i) 4 and 15 respectively
(ii) 3 and 14 respectively (✓)
(iii) 4 and 14 respectively
(iv) 3 and 15 respectively

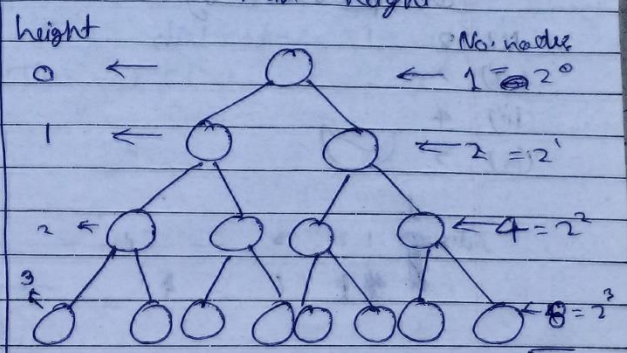


Page No.: 171
Date: / /

Max. height



Min height



Perfect

For Complete binary search tree-

depth No. nodes-

0 → 1 = 2^0 = $2^1 - 1$

1 → 1+2 = 3 = $2^2 - 1$

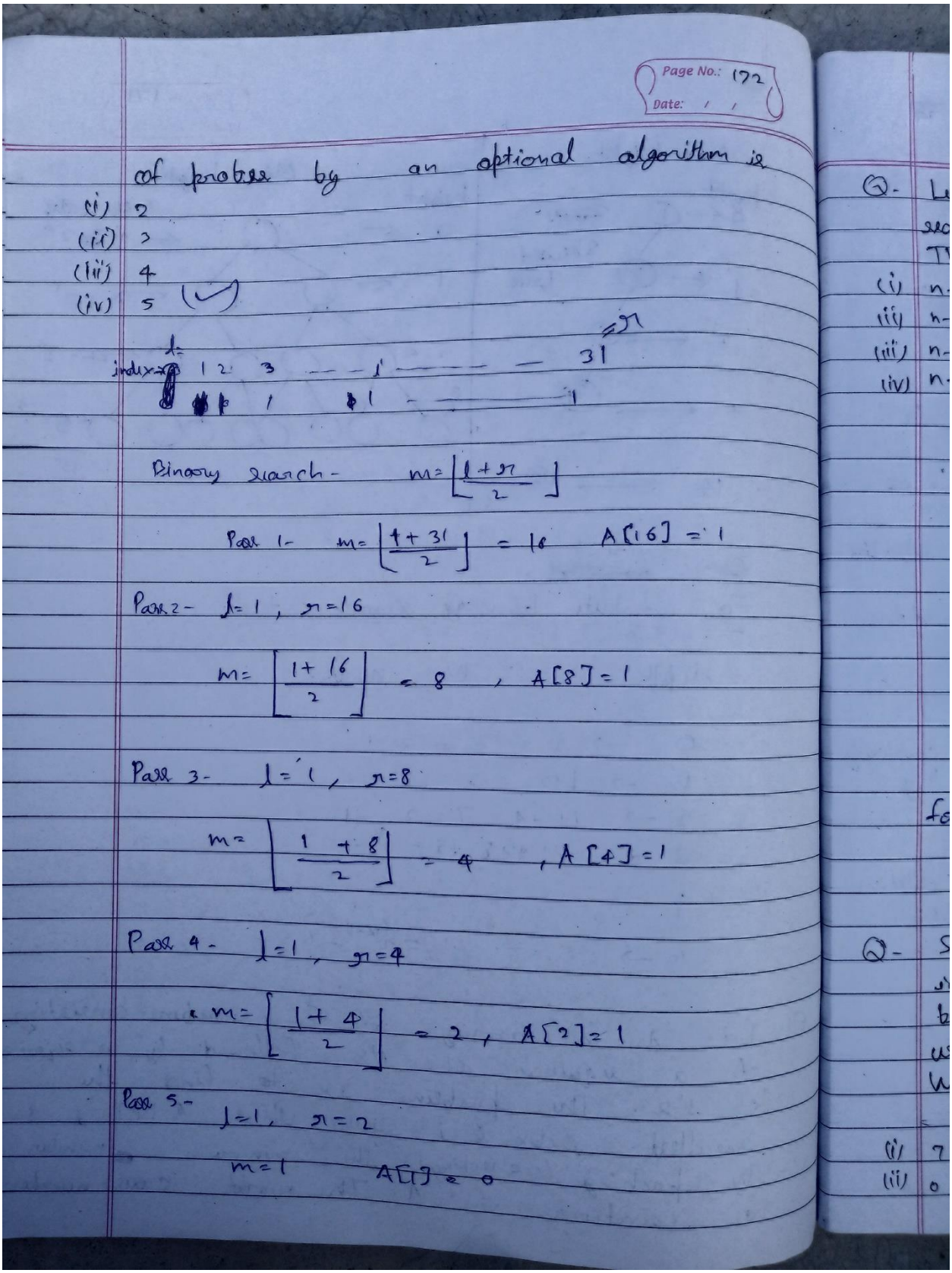
2 → 1+2+4 = 7 = $2^3 - 1$

3 → 1+2+4+8 = 15 = $2^4 - 1$

⋮

k → $= 2^{(k+1)} - 1$

Q- Let A be array of 31 numbers consisting of a sequence of 0's followed by a sequence of 1's. The problem is to find the smallest index i such that A[i] is 1 by probing (checking) the minimum number of locations in A. The worst case number



of probes by an optimal algorithm is

(i) 2
(ii) 3
(iii) 4
(iv) 5 (✓)

index → 1 2 3 ... i ... 31 = n

Binary search - $m = \left\lfloor \frac{l+r}{2} \right\rfloor$

Pass 1 - $m = \left\lfloor \frac{1+31}{2} \right\rfloor = 16$ A[16] = 1

Pass 2 - l = 1, r = 16

$m = \left\lfloor \frac{1+16}{2} \right\rfloor = 8$, A[8] = 1

Pass 3 - l = 1, r = 8

$m = \left\lfloor \frac{1+8}{2} \right\rfloor = 4$, A[4] = 1

Pass 4 - l = 1, r = 4

$m = \left\lfloor \frac{1+4}{2} \right\rfloor = 2$, A[2] = 1

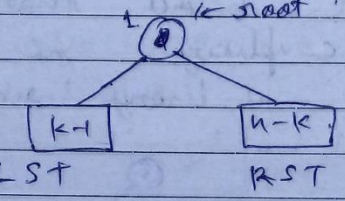
Pass 5 - l = 1, r = 2

m = 1 A[1] = 0

Page No.: 173
Date: / /

Q- Let $T(n)$ be the number of different binary search trees on n distinct elements.
Then $T(n) = \sum_{k=1}^{n} T(k-1)\,T(x)$, where x is -

(i) $n-k+1$
(ii) $n-k$ ✓
(iii) $n-k-1$
(iv) $n-k-2$

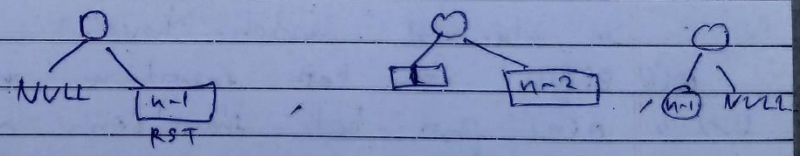


$$T(n) = \sum_{k=1}^{n} T(k-1)\,T(x)$$ → $n-k$

$n = 1 + (k-1) + n-k$

$k = 1, 2, 3, 4, \ldots n$

for $k=1$.



Q- Suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. The binary search tree uses the usual ordering on natural numbers. What is in-order traversal sequence of the resultant tree?

(i) 7 5 1 0 3 2 4 6 8 9
(ii) 0 2 4 3 1 6 5 9 8 7

(iii) 0 1 2 3 4 5 6 7 8 9 (✓)

(iv) 9 8 6 4 2 3 0 1 5 7

inorder traversal of BST always a sorted array.

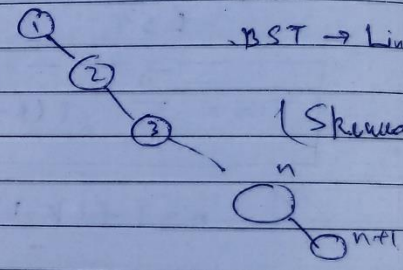
Q- Which one of the following is the tightest upper bound that represents the time complexity of inserting an object into a binary search tree of n nodes?

(i) O(1)

(ii) O(lgn)

(iii) O(n) (✓)

(iv) O(nlgn)



BST → Linked List (Skewed tree)

Q- What are the worst-case time complexity of insertion and deletion of a key in binary search tree?

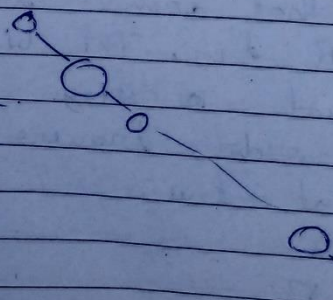
(i) Θ(lgn) for both insertion and deletion

(✓) (ii) O(n) for both insertion and and deletion

(iii) Θ(n) for insertion and Θ(lgn) for deletion

(iv) Θ(lgn) for insertion and Θ(n) for deletion.

n → BST = LL



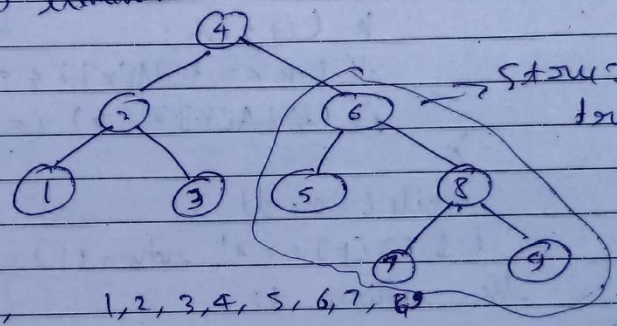
Page No.: 175
Date: / /

Q- We are given a set of n distinct elements and an unlabeled binary tree with n nodes. In how many ways can we populate the tree with the given set so that it become a binary search tree?

(i) 0
(ii) 1 (✓)
(iii) n!
(iv) $(1/(n+1)).{}^{2n}C_n$

for distinct elements.

only one way



→ Structure of tree fixed.

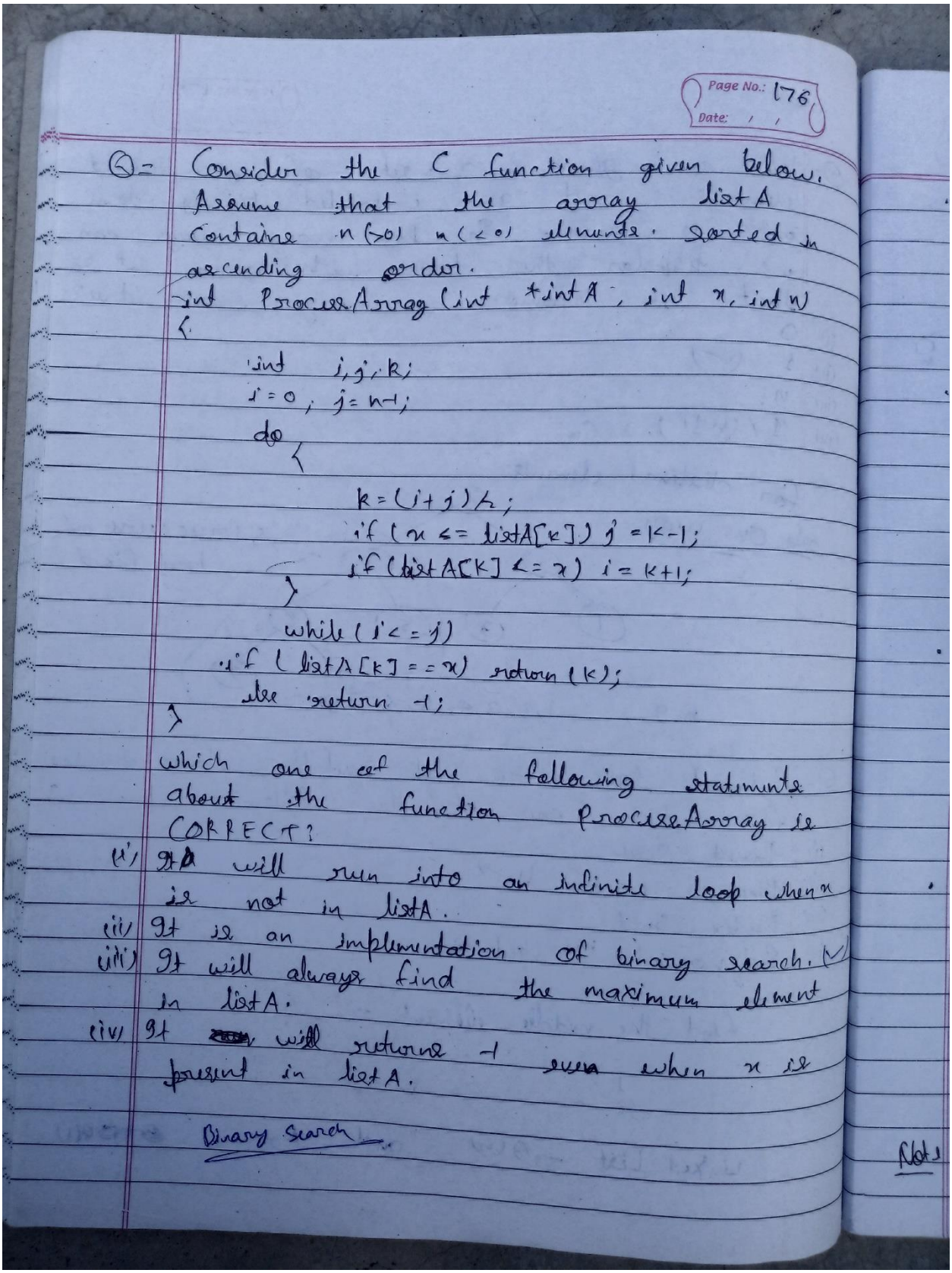
n=9, 1, 2, 3, 4, 5, 6, 7, 8, 9

Q- Linked lists are not suitable data structure for which one of the following problems?

(i) Insert Sort
(ii) Binary Search (✓)
(iii) Radix Sort
(iv) Polynomial manipulation.

find the middle element -

Linked List - $\theta(n)$ and Array O(1)



Page No.: 176
Date: / /

Q= Consider the C function given below. Assume that the array listA contains n (>0) n (<0) elements. sorted in ascending order.

```
int ProcessArray (int *intA, int x, int n)
{
    int i, j, k;
    i = 0; j = n-1;
    do
    {
        k = (i+j)/2;
        if (x <= listA[k]) j = k-1;
        if (listA[k] <= x) i = k+1;
    }
    while (i <= j)
    if (listA[k] == x) return (k);
    else return -1;
}
```

which one of the following statements about the function ProcessArray is CORRECT?

(i) It will run into an infinite loop when x is not in listA.
(ii) It is an implementation of binary search. ✓
(iii) It will always find the maximum element in listA.
(iv) It will returns -1 even when x is present in listA.

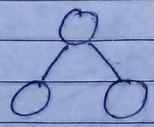
Binary Search

Note:

Trees -

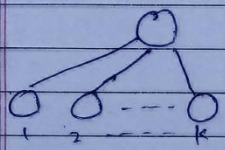
Page No.: 179
Date: / /

- **Binary tree** - 2 ary tree



Every node in binary tree has a max. of 2 children

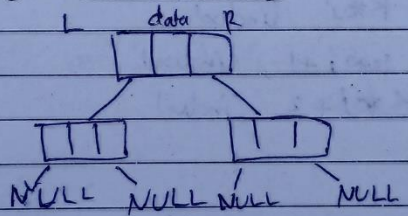
- **K-ary tree** - k way tree (A simple generalization of Binary tree)



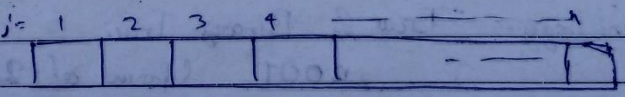
Every node in a K-ary tree has a max. of K-children.

Binary tree implementation -

- using Pointer/reference -



- using Array -



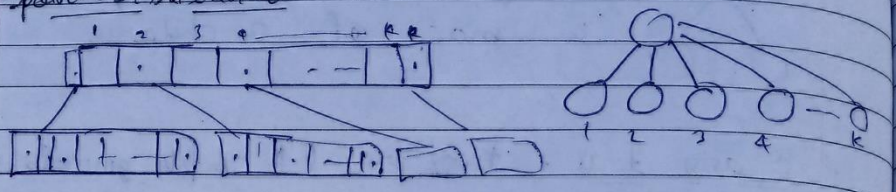
at i index
- left child = 2*i
- right child = 2*i + 1

Note: BST is a kind of Binary Tree with special properties.

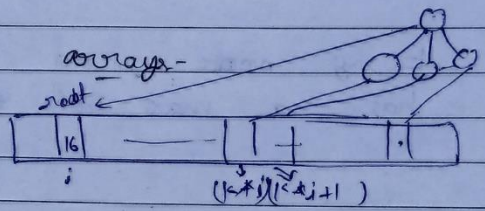
Page No.: 178
Date: / /

K-ary tree implementation-

using - pointers reference -



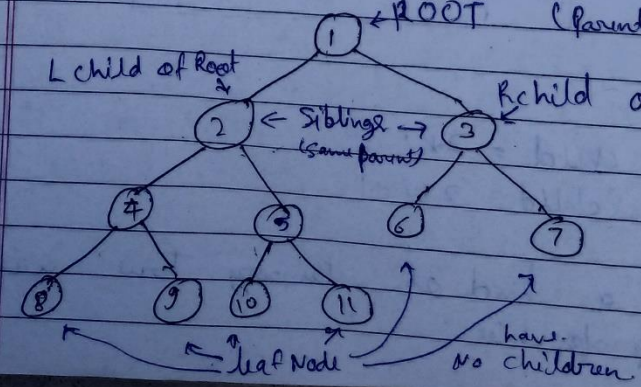
using arrays -



for i^{th} index node -

1st child ← k*i (index).
2nd child ← k*i+1 (index)
3rd child ← k*i+2 (index)
|
k^{th} child ← (k*i) + (k-1) (index)

Terminology - Tree & Binary Tree.



Page No.: 179
Date: / /

- **Descendent** - all the nodes which are exist in the left subtree of a parent node are called the descendent of that parent node.

  2, 4, 8, 9, 5, 10, 11 are descendent of 1

- **Ancestor** - opposit of descendent

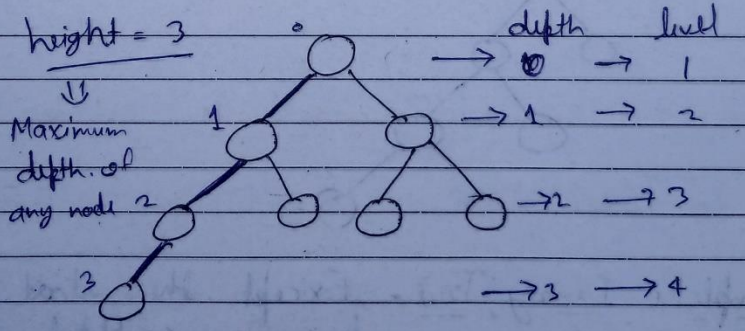
  4, 2, 1 are the ancestor of 8

- **Leaf (External) Node** - have no child.

- **Internal Node** - No-leaf, have a child

- **Degree of a Node** - No. of children.

- **Depth of a Node** - No. of edges / Links between node & Root.



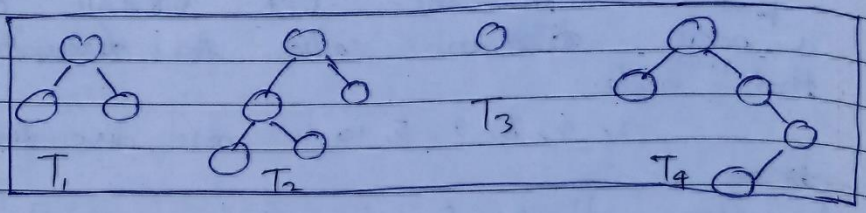
- **Level of a node** - 1+ depth of the node.

  = 1+ no. of edges between node & root

- **Height of a Tree** - No. of edge on the longest path from any node to the root node.

Page No.: 180
Date: / /

- Forest - Set of disjoint Trees.



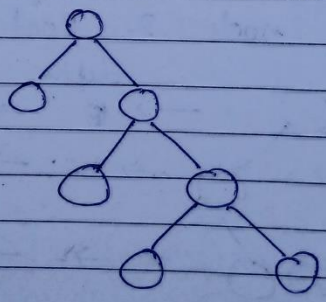
→ Traversal -

- Inorder - L, Root, R
- Preorder - Root, L, R
- Postorder - L, R, Root

Types of Binary Tree -

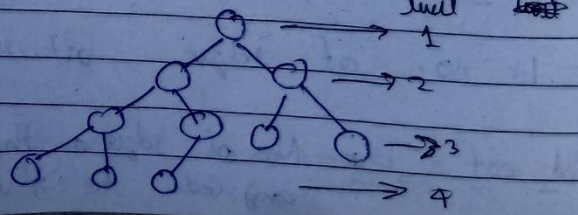
1- Full Binary Tree - Every node has 0 or 2 children.

Eg -



2- Complete Binary Tree - Except the last level tree is completely filled and last level leafs are present left to right.

Eg -

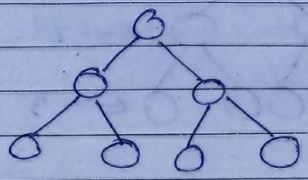


Page No.: 181
Date: / /

3- Perfect Binary Tree-

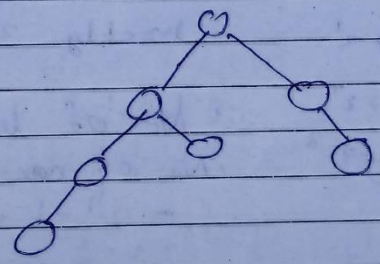
- All interior nodes have 2 children and all the leaves are at same level.

Eg-



4- Balanced Binary Tree- At any node, height of LST and RST do not differ by more than 1.

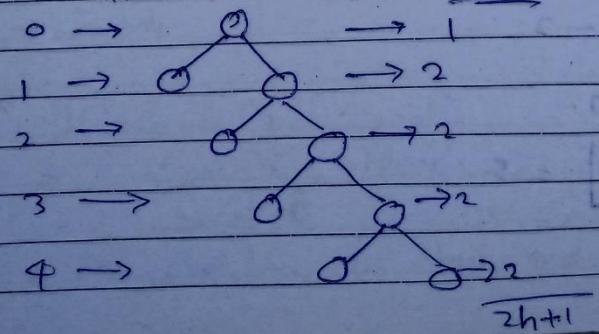
Eg-



→ Properties of Binary Trees-

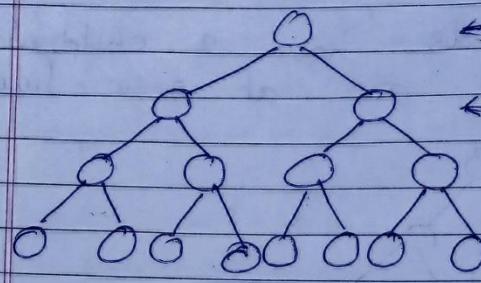
1- Full Binary Tree- Let height of tree = h.

- Minimum no. of Nodes = $2h+1$



Page No.: 182
Date: / /

1- Maximum no. of nodes-



← 0 ⇒ 1 = 2^0

← 1 ⇒ 2 = 2^1

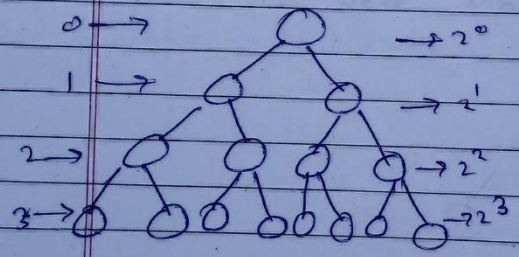
← 2 ⇒ 4 = 2^2

← 3 ⇒ 8 = 2^3

15 = $2^4 - 1$

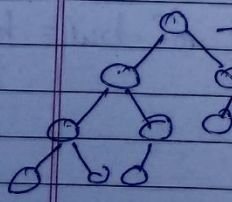
$2^0 + 2^1 + 2^2 + \cdots + 2^h = 2^{h+1} - 1$.

2- Perfect Binary Tree- Let height of tree = h.



- It has to contain exactly $2^{h+1} - 1$ nodes.
- No. of leaf nodes = $2^h = 1$
- No. of non-leaf nodes (Internal) = $2^h - 1 = 1 - 1$

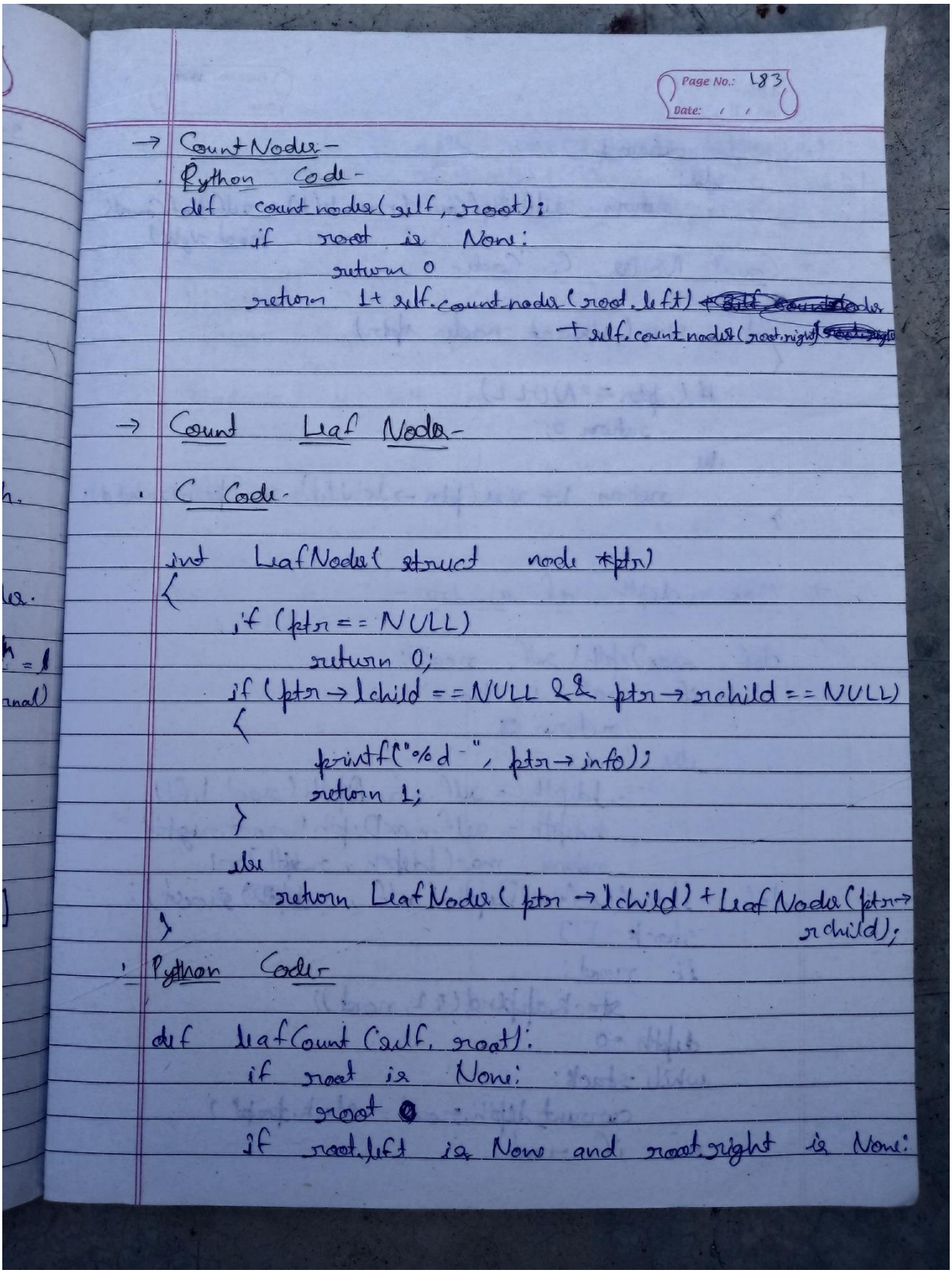
3- Complete Binary Tree-



if n = no. of nodes
No. of internal nodes = $\lfloor n/2 \rfloor$

n = 10

$\left\lfloor \frac{10}{2} \right\rfloor = 5$.



Page No.: 183
Date: / /

→ Count Node -

Python Code -

```
def count_node(self, root):
    if root is None:
        return 0
    return 1 + self.count_node(root.left) + self.count_node(root.right)
```

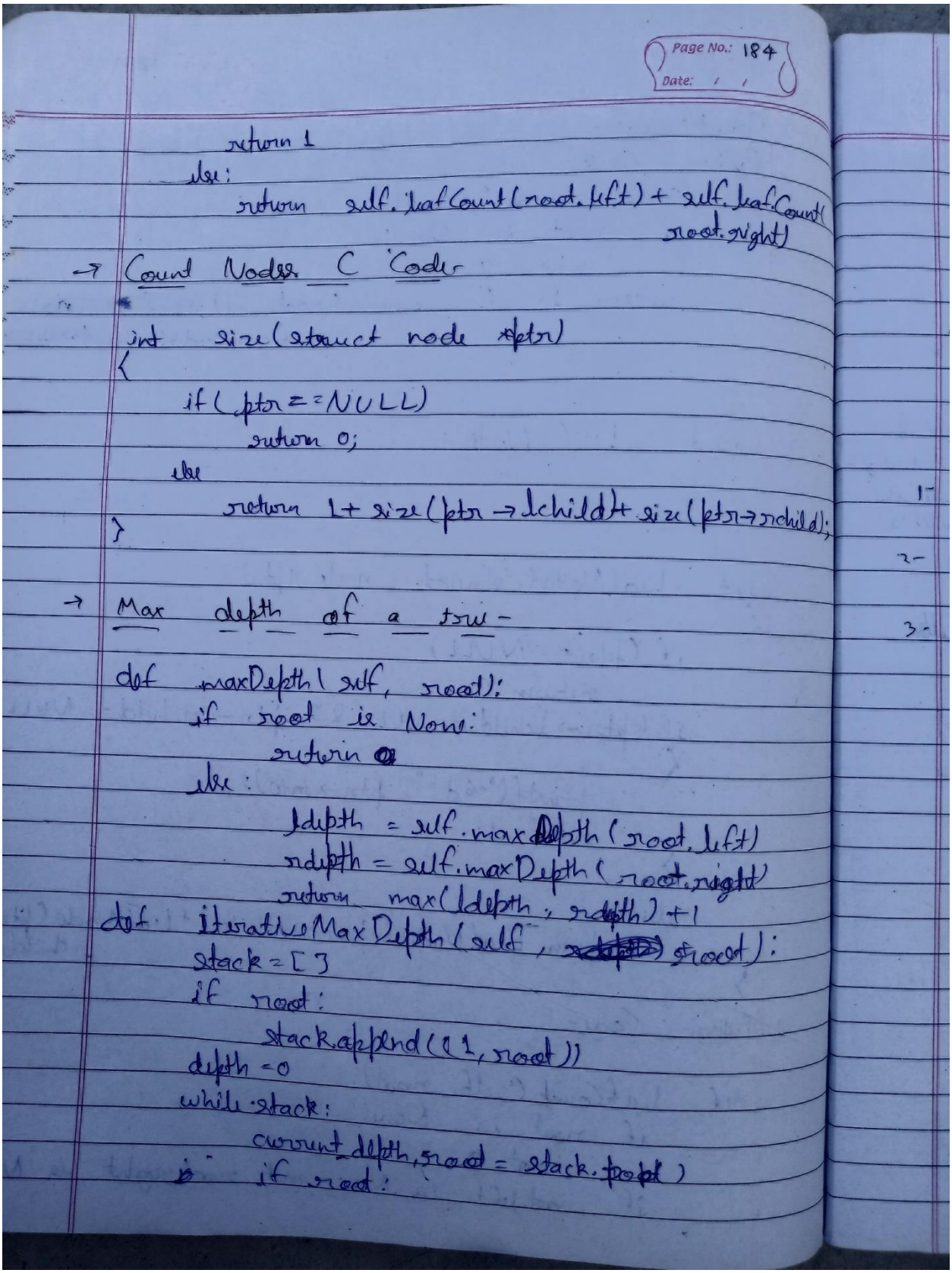
→ Count Leaf Node -

C Code -

```
int LeafNode(struct node *ptr)
{
    if (ptr == NULL)
        return 0;
    if (ptr->lchild == NULL && ptr->rchild == NULL)
    {
        printf("%d ", ptr->info);
        return 1;
    }
    else
        return LeafNode(ptr->lchild) + LeafNode(ptr->rchild);
}
```

Python Code -

```
def leafCount(self, root):
    if root is None:
        root
    if root.left is None and root.right is None:
```



Page No.: 184
Date: / /

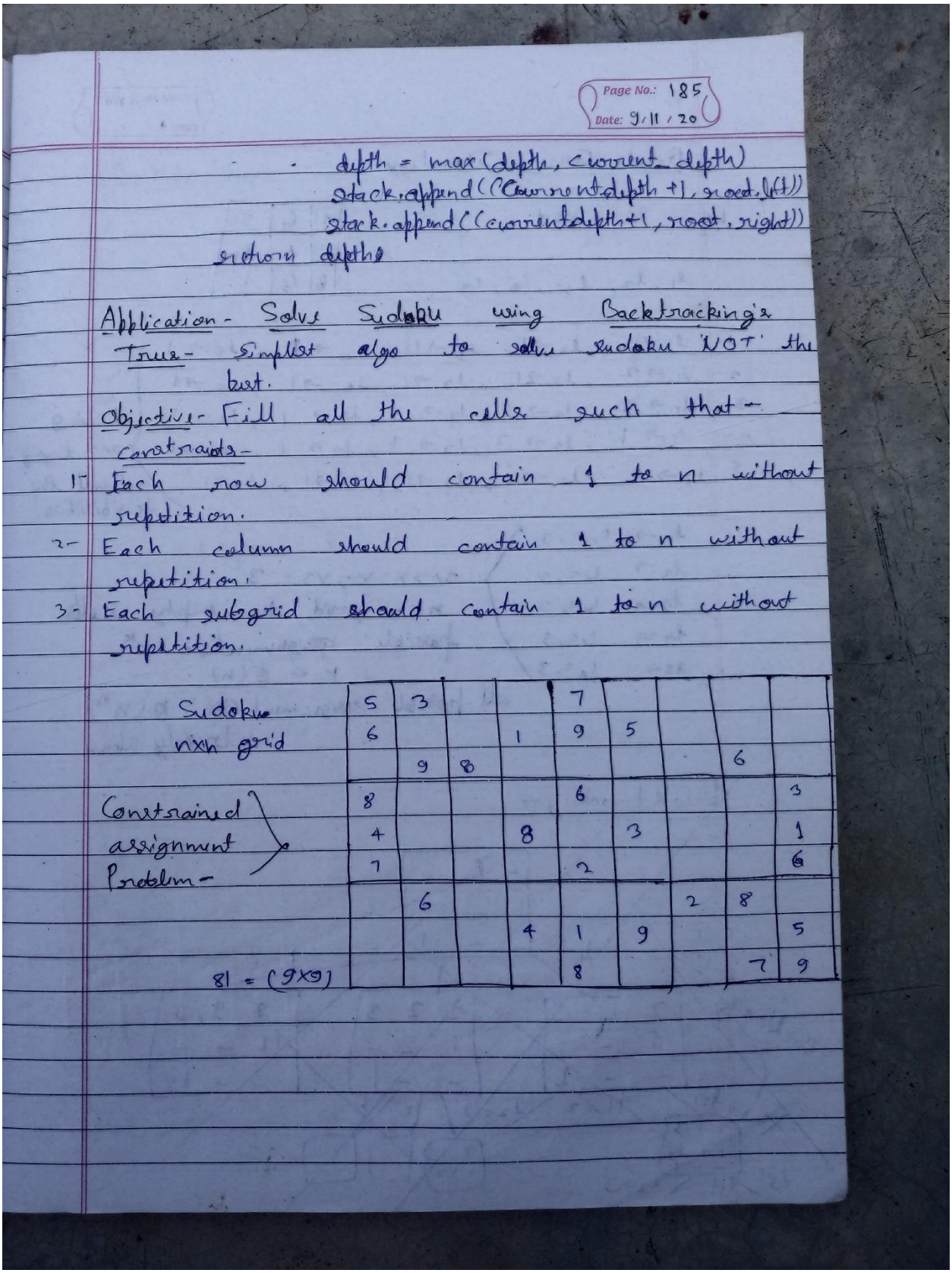
```
        return 1
    else:
        return self.leafCount(root.left) + self.leafCount(
                                            root.right)
```

→ Count Nodes C Code-

```
int size(struct node *ptr)
{
    if(ptr==NULL)
        return 0;
    else
        return 1+size(ptr->lchild)+size(ptr->rchild);
}
```

→ Max depth of a tree -

```
def maxDepth(self, root):
    if root is None:
        return 0
    else:
        ldepth = self.maxDepth(root.left)
        rdepth = self.maxDepth(root.right)
        return max(ldepth, rdepth)+1
def iterativeMaxDepth(self, root):
    stack = []
    if root:
        stack.append((1, root))
    depth = 0
    while stack:
        current_depth, root = stack.pop()
        if root:
```



Page No.: 185
Date: 9/11/20

depth = max(depth, current_depth)
stack.append((current_depth+1, root.left))
stack.append((current_depth+1, root.right))
return depth

Application - Solve Sudoku using Backtracking's
True - Simplest algo to solve sudoku NOT the best.

Objective - Fill all the cells such that -

Constraints -

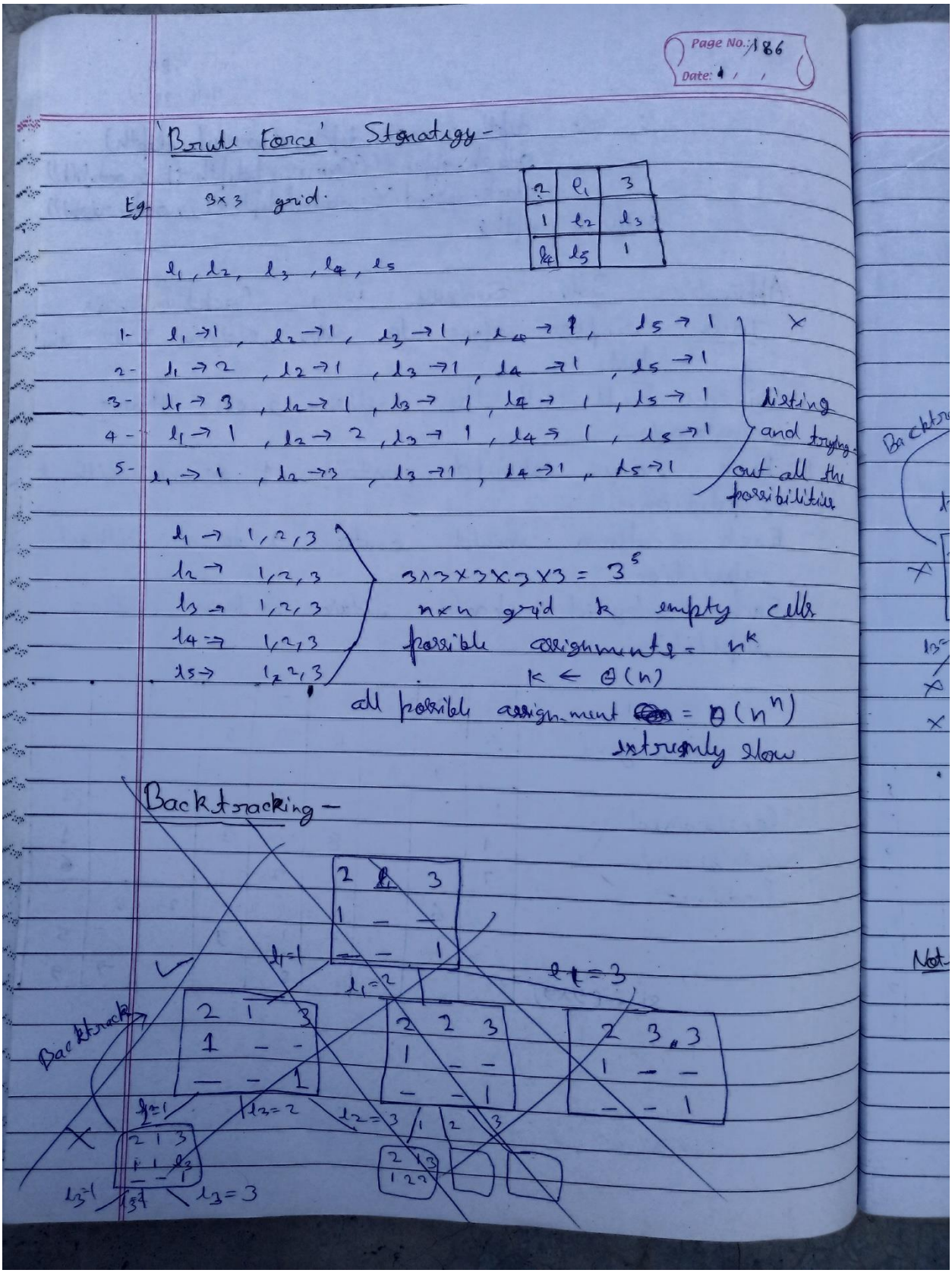
1- Each row should contain 1 to n without repetition.
2- Each column should contain 1 to n without repetition.
3- Each subgrid should contain 1 to n without repetition.

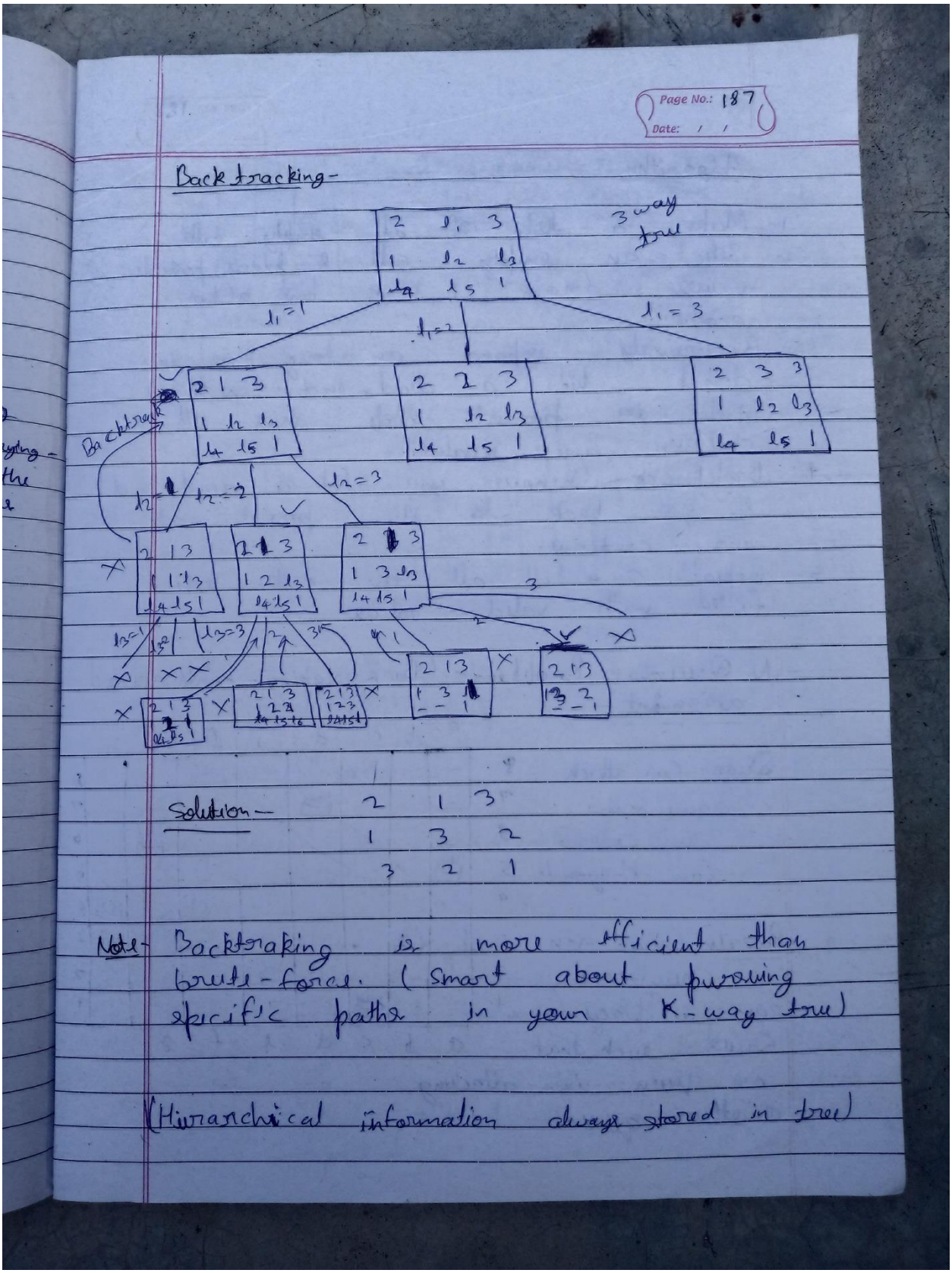
Sudoku nxn grid

Constrained assignment Problem -

81 = (9x9)

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9


Page No. 186
Date:
'Brute Force' Strategy -
Eg. 3×3 grid.
2 l1 3
1 l2 l3
l4 l5 1
l1, l2, l3, l4, l5
1- l1→1, l2→1, l3→1, l4→1, l5→1
2- l1→2, l2→1, l3→1, l4→1, l5→1
3- l1→3, l2→1, l3→1, l4→1, l5→1
4- l1→1, l2→2, l3→1, l4→1, l5→1
5- l1→1, l2→3, l3→1, l4→1, l5→1
listing and trying out all the possibilities
l1→ 1,2,3
l2→ 1,2,3
l3→ 1,2,3
l4→ 1,2,3
l5→ 1,2,3
3×3×3×3×3 = 3^5
n×n grid & empty cells
possible assignments = n^k
k ← Θ(n)
all possible assignment = Θ(n^n)
strongly slow
Backtracking -
l1=1
l1=2
l1=3
Backtrack
l2=1
l2=2
l2=3
l3=1
l3=2
l3=3


Page No.: 187
Date: / /
Back tracking-
3 way true
Backtrack
Solution-
2 1 3
1 3 2
3 2 1
Note- Backtracking is more efficient than brute-force. (Smart about pursuing specific paths in your K-way tree)
(Hierarchical information always stored in tree)

Page No.: 188
Date: / /

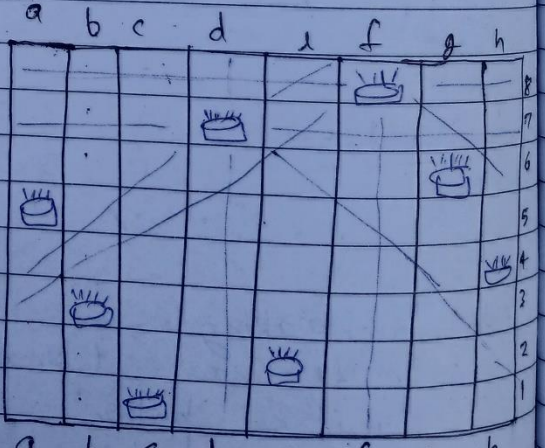
Algorithm -

1- Make a list of all empty cells.
2- Select an empty cell & place possible values from 1 to n one after another.
3- Recursively expand as long as you don't hit a dead-end. (node is invalid or parent node where all children are invalid).
4- Backtrack whenever you hit a dead-end & go back to the parent node and continue.
5- Repeat 2-4 till all the cells are filled with valid values.

N-Queens Problem - (Backtracking - constrained assignment)

Queen - Can attack
- same row
- same column
- same diagonals

Puzzle - Given n×n chessboard can we place n-Queens such that no queen is attacking another queen.



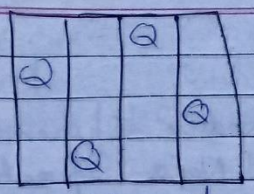
Eg -

1st Column

1-
2-
3-
4-

Page No.: 189
Date: / /

Eg-



4×4 grid
4 queens

Valid Assignment

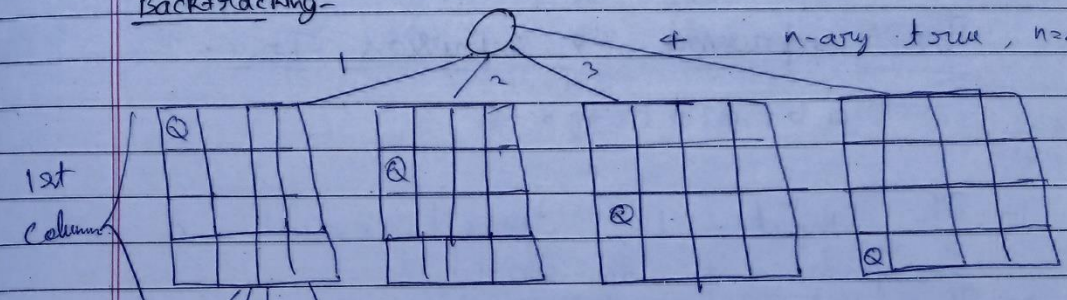
Brute force –

Total cells = n×n = 4×4 = 16 cells
queens = 4

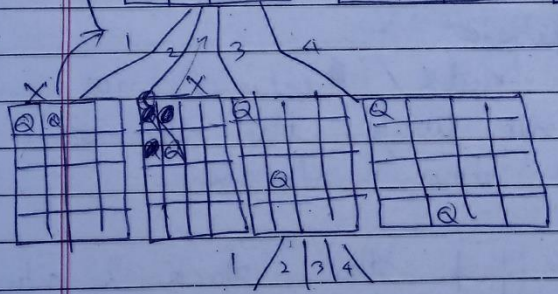
$^{16}C_4$ = No. ways you can pick 4 cells out of 16 cells.

$^{n\times n}C_n = {}^{n^2}C_n$ combinations / possible

Backtracking –



1st column

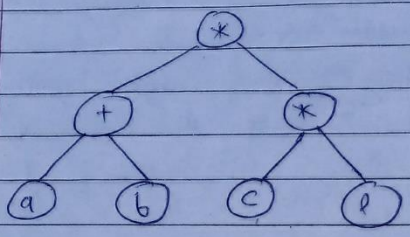


hit a dead-end
backtrack

1- Go column by column from left to right.
2- If n queens are placed (or) all columns return.
3- Try placing a queen in each row of the current column → recursively
→ If you hit a dad end → invalid node
→ Simply back track
→ Parent node will all invalid children
4- In case you are not able place n queens in an n×n grid after trying out all possibilities return "impossible".

Page No.: 190
Date: / /

Application - Expression Tree -



leaf Node - Operands
internal Node - operators.

(a+b)*(c*d)

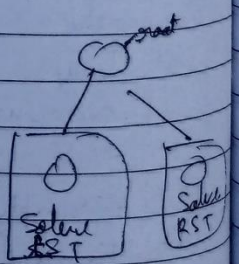
Postfix expression → expression Tree -

a b + c d * + *

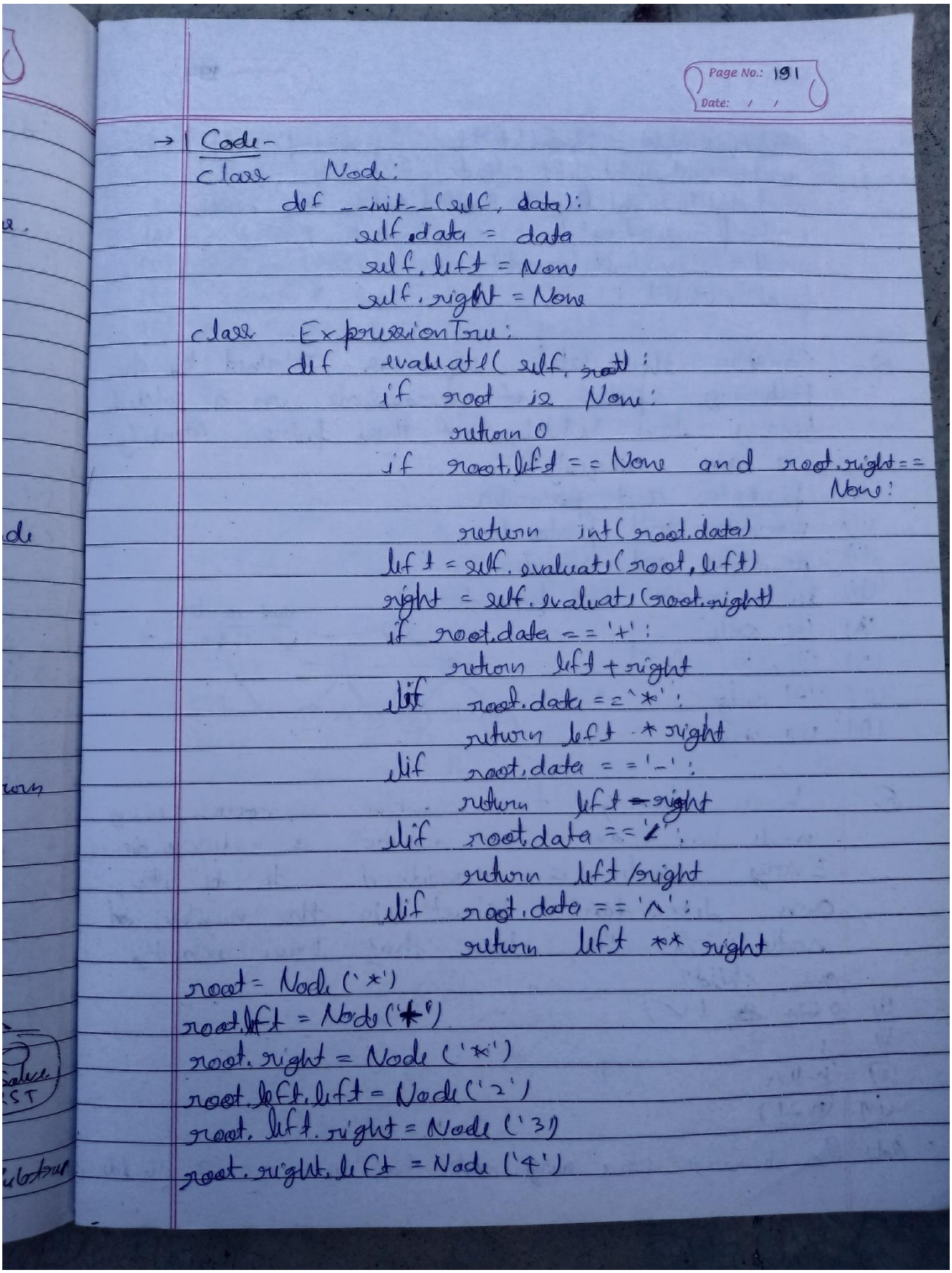
1- If input is operand create a node & push it to stack.
2- If input is operator,
a) pop the top 2 node / entries / tree.
b) Build a new tree using the operator
c) Push the new tree into stack.
d)
3- If end of input pop the stack & return the resultant tree.

→ Evaluating an expression Tree -

```
ival(node)
    if node is operand
        return operand
    else // node is an operator
        l = ival(node.left)
        r = ival(node.right)
        return (l.operator r)
```



Recursive Solve Subtrees



Page No.: 191
Date: / /

→ Code-

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
class ExpressionTree:
    def evaluate(self, root):
        if root is None:
            return 0
        if root.left == None and root.right == None:
            return int(root.data)
        left = self.evaluate(root.left)
        right = self.evaluate(root.right)
        if root.data == '+':
            return left + right
        elif root.data == '*':
            return left * right
        elif root.data == '-':
            return left - right
        elif root.data == '/':
            return left / right
        elif root.data == '^':
            return left ** right
root = Node('*')
root.left = Node('+')
root.right = Node('*')
root.left.left = Node('2')
root.left.right = Node('3')
root.right.left = Node('4')
```

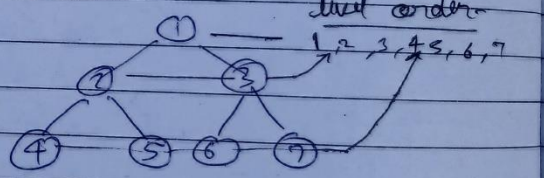
Page No.: 192
Date: / /

```
root.right.right = Node('+')
root.right.right.left = Node('5')
root.right.right.right = Node('6')
t = ExpressionTree()
result = t.Evaluate(root)
print(result)
```

Q- Consider the label sequence obtained by the following pairs of traversals on a labelled binary tree. Which of these pairs identify a tree uniquely?

(i) preorder and postorder
(ii) inorder and postorder
(iii) preorder and inorder
(iv) level order and postorder.

(A) (i) only
(B) (ii), (iii) ✓
(C) (iii) only
(D) (iv) only



Q- In a binary tree with n nodes, every node has an odd number of descendants. Every node is considered to be its own descendant. What is the number of nodes in the tree that have exactly one child?

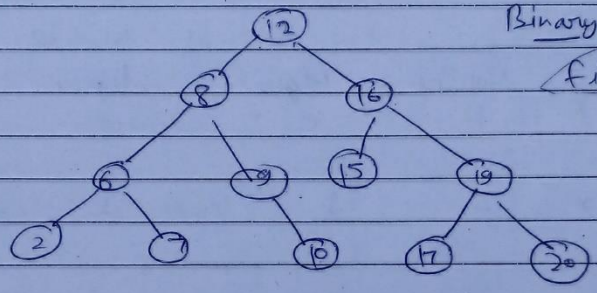
(i) 0 ✓
(ii) 1
(iii) (n-1)/2
(iv) (n-1)

Note- In binary tree every node has zero or 2 children

Page No.: 193
Date: / /

Q- The pre-order traversal of a binary search tree is given by 12, 8, 6, 2, 7, 9, 10, 16, 15, 19, 17, 20. Then the post-order traversal of this tree is-

(A) 2, 6, 7, 8, 9, 10, 12, 15, 16, 17, 19, 20
(B) 2, 7, 6, 10, 8, 15, 17, 20, 19, 16, 12 (✓)
(C) 7, 2, 6, 8, 9, 10, 20, 17, 19, 15, 16, 12
(D) 7, 6, 2, 10, 9, 8, 15, 16, 17, 20, 19, 12.

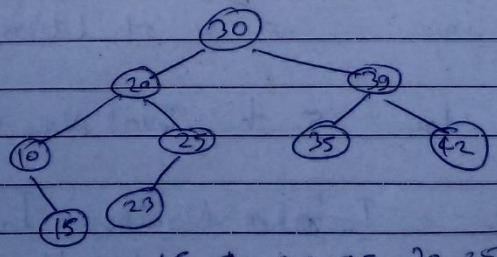


Binary and preorder first node is root like than 12 in left subtree and grater than 12 in right subtree

2, 7, 6, 10, 9, 8, 15, 17, 20, 19, 16, 12.

Q- The preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42 which one of the following is the postorder traversal sequence of the same tree?

(A) 10, 20, 15, 23, 25, 35, 42, 39, 30
(B) 15, 10, 25, 23, 20, 42, 35, 39, 30
(C) 15, 20, 10, 23, 25, 42, 35, 39, 30
(D) 15, 10, 23, 25, 20, 35, 42, 39, 30 (✓)

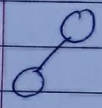
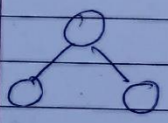


15, 10, 23, 25, 20, 35, 42, 39, 30

Page No.: 94
Date: / /

Q- In a binary tree the number of internal nodes of degree 1 is 5 and the number of internal nodes of degree 2 is 10. The number of leaf nodes in the binary tree is -

(A) 10
(B) 11 ✓
(C) 12
(D) 15

	Nodes	Edges	degree
○ →	1	0	0
 →	2	1	1
 →	3	2	2

for n nodes (n-1) edges

5 internal node → degree=1 } → edge = 5×1=5
10 internal nodes → degree=2 } → edge = 10×2=20
Total No. edges = 25

There is not any other internal node because in a binary tree any node can have 2 or 1 children only

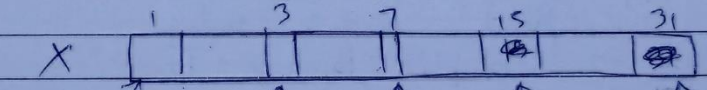
Total No. edges = 25 then Total No. nodes = 25+1 = 26

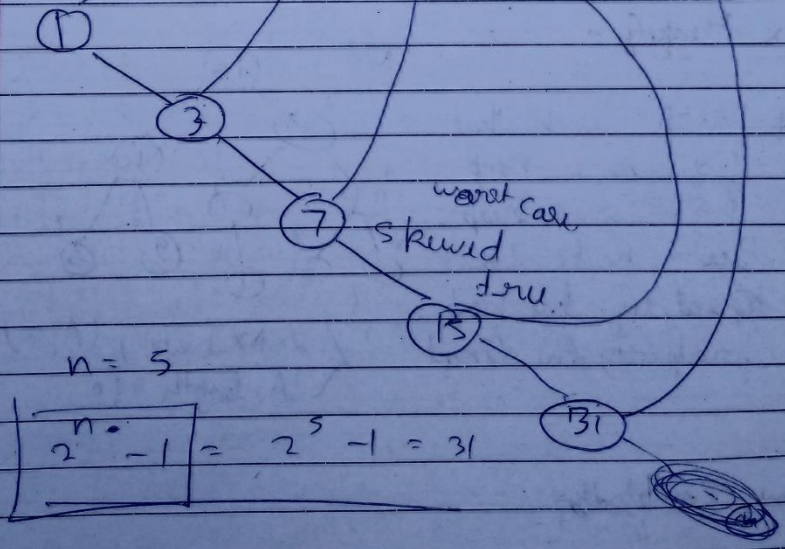
Leaf nodes = Total Nodes − Internal nodes
= 26 − (5+10) = 11

Page No.: 195
Date: / /

Q- A scheme for storing binary trees in an array X is as follows. Indexing of X starts at 1 instead of 0. The root is stored at X[1]. For a node stored at X[i], the left child, if any, is stored in X[2i] and the right child, if any, in X[2i+1]. To be able to store any binary tree on n vertices the minimum size of X should be.

(i) $\lg n$
(ii) n
(iii) $2n+1$
(iv) $2^n - 1$ (✓)





$n = 5$

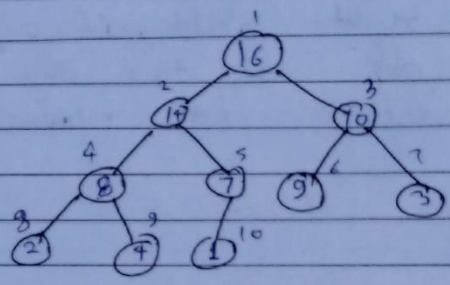
$\boxed{2^n - 1} = 2^5 - 1 = 31$

Heap

Page No.: 196
Date: 10/11/20

Heap ↔ Priority Queue } Min Heap & Max Heap

Max Heap – Complete Binary Tree every parent node's value ≥ the value in the child node



array implementation of a binary tree

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

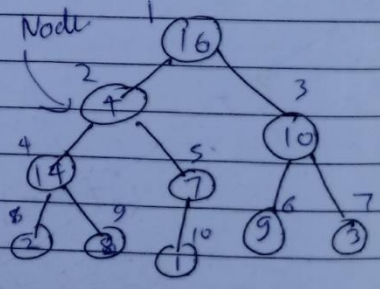
Min Heap – Every parent node's value ≤ child node's value.

Note– Heap is a Complete Binary Tree.

Heap Sort –

- **Max Heapify** –

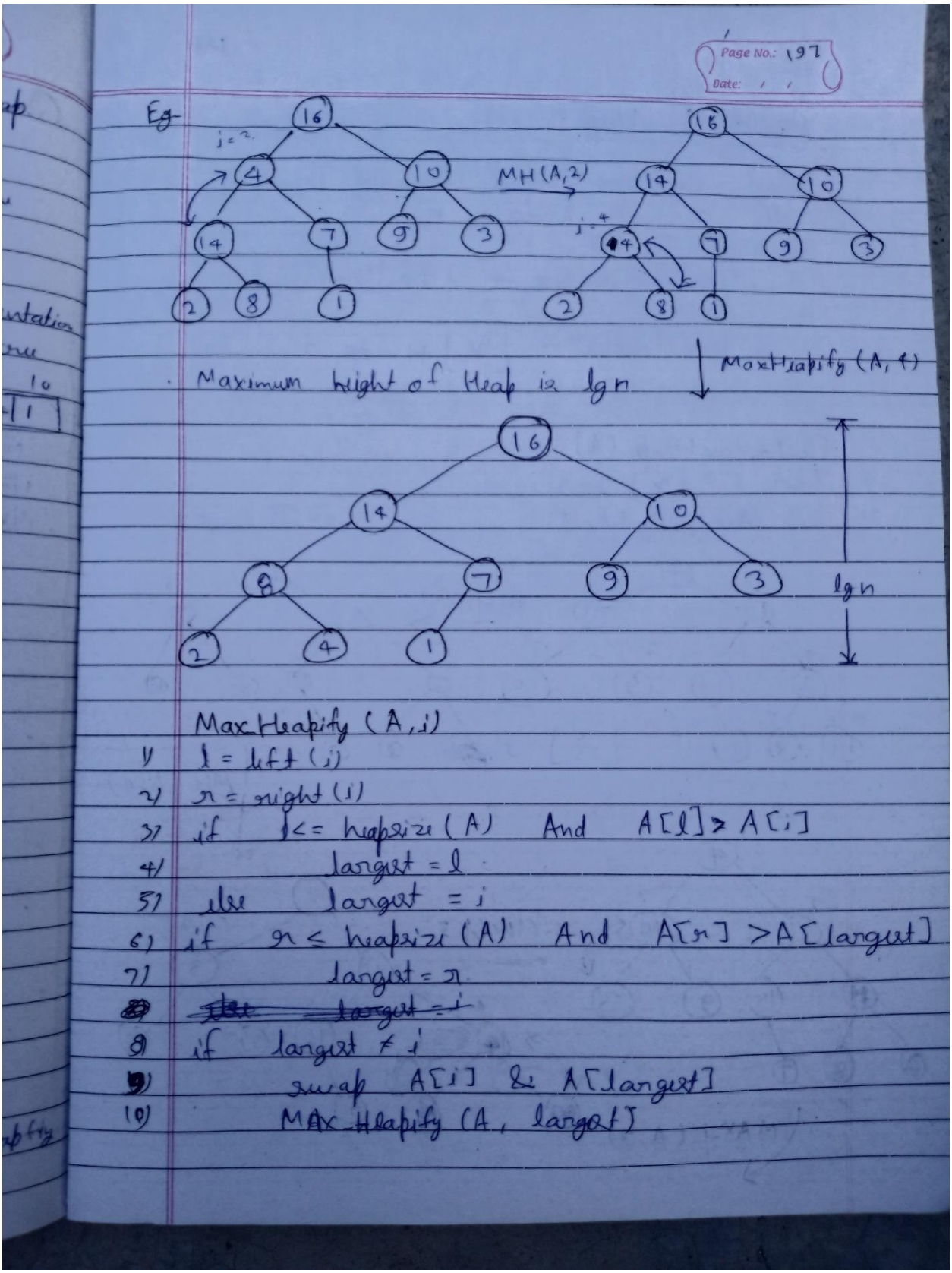
Task– Node such that LST is a heap RST is a heap But node doesn't satisfy the heap property the first




```
MAX_heapify (A, 2)
A.length = 10
```

Time Complexity

Max. no. of comparisons & swaps to max heapify = $h = \theta(\lg n)$. (for n nodes).

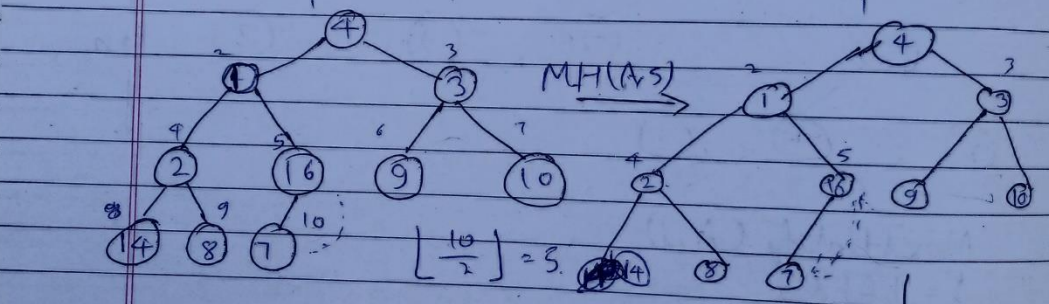

Page No.: 197
Date:
Eg
i=2
MH(A,2)
i=4
Maximum height of Heap is lgn
MaxHeapify (A, i)
lgn
Max_Heapify (A,i)
1) l = left (i)
2) r = right (i)
3) if l <= heapsize (A) And A[l] > A[i]
4) largest = l
5) else largest = i
6) if r <= heapsize (A) And A[r] > A[largest]
7) largest = r
8) ~~else largest = i~~
9) if largest ≠ i
9) swap A[i] & A[largest]
10) MAX_Heapify (A, largest)

Page No.: 198
Date: / /

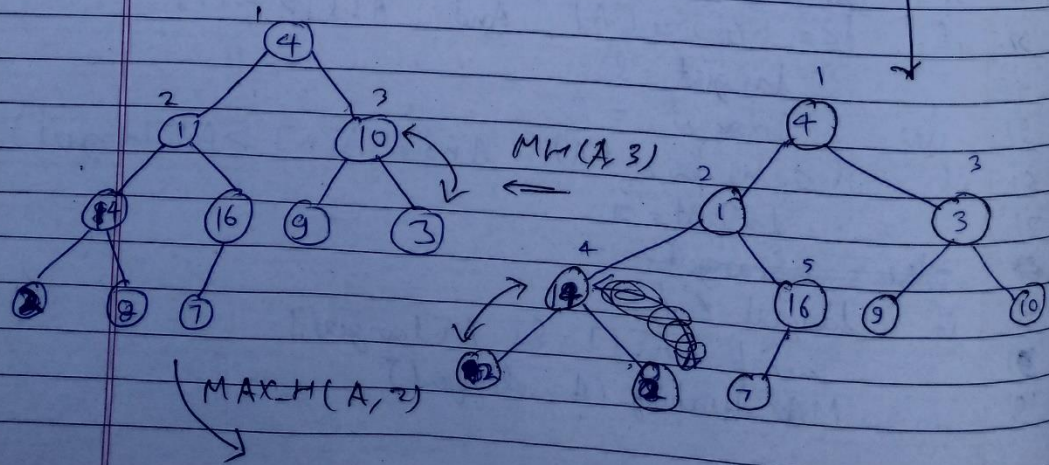
Build A Heap -

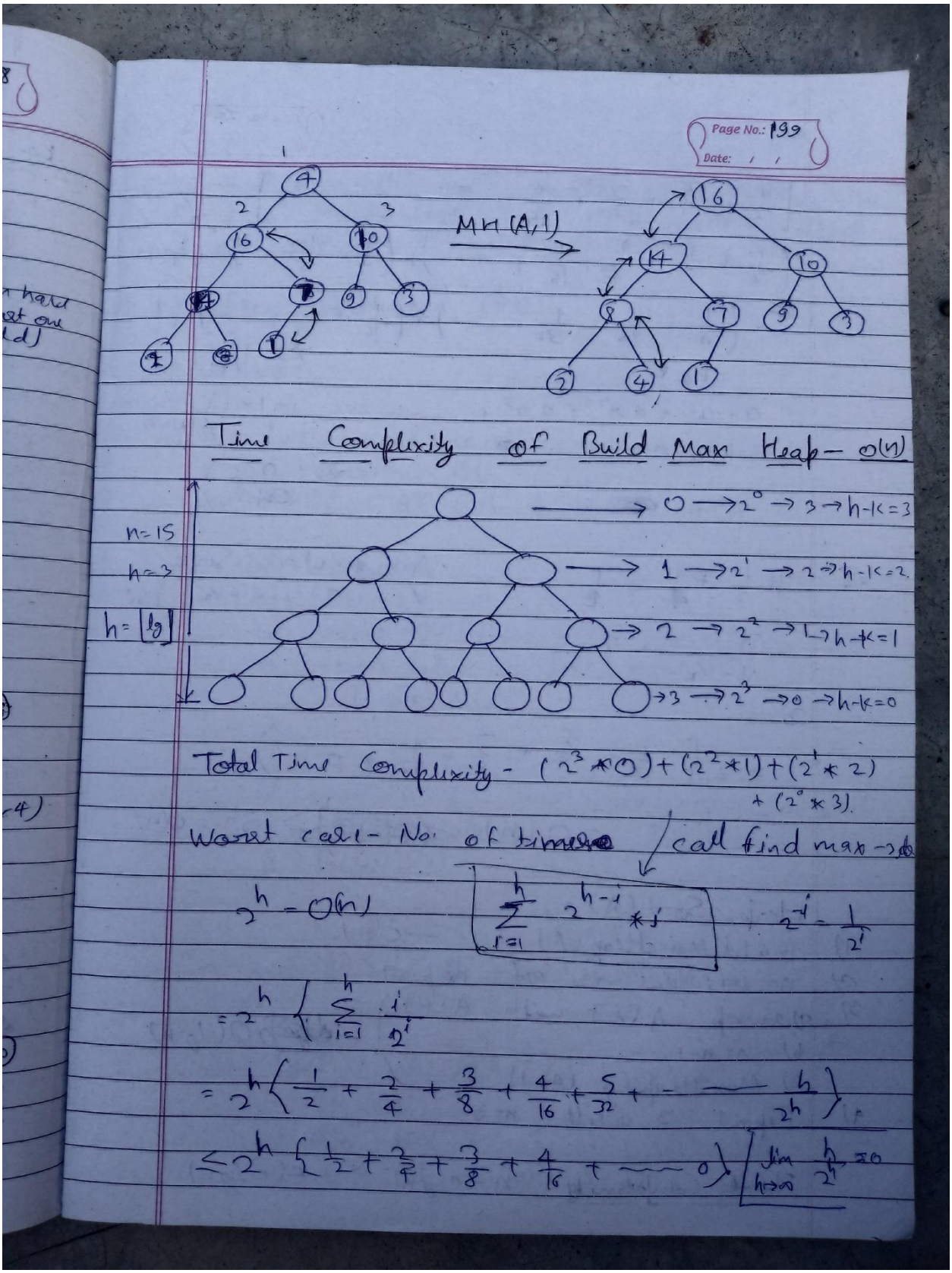
For ~~a tree~~ Complete Binary Tree
of no. of Node - n
then index of -
- internal nodes - 1 to $\lfloor n/2 \rfloor$ (which have at least one child)
- leaf nodes - $\lfloor n/2 \rfloor + 1$ to n

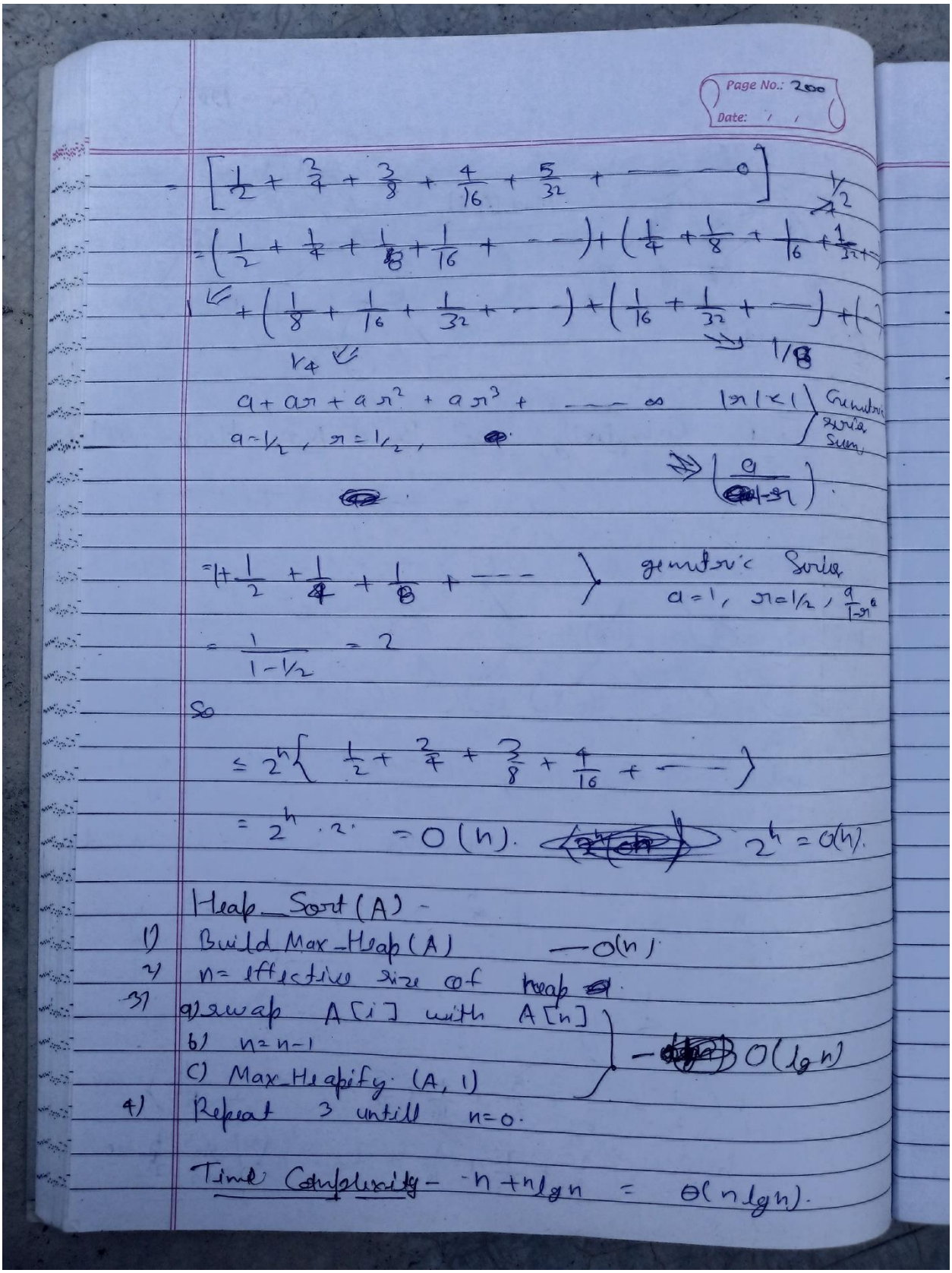
BuildMaxHeap (A)
1) for i = $\lfloor n/2 \rfloor$ to 1
2) Max_Heapify (A, i) (n = A.length)



MH (A,4)




Page No.: 199
Date: / /
MH(A,1)
Time Complexity of Build Max Heap - O(n)
n=15
h=3
h=⌊lg⌋
0 → 2^0 → 3 → h-i<=3
1 → 2^1 → 2 → h-i<=2
2 → 2^2 → 1 → h-i<=1
3 → 2^3 → 0 → h-i<=0
Total Time Complexity - (2^3 * 0) + (2^2 * 1) + (2^1 * 2) + (2^0 * 3)
Worst case - No. of times / call find max
2^h = O(n)
Σ_{i=1}^{h} 2^{h-i} * i
2^{-i} = 1/2^i
= 2^h { Σ_{i=1}^{h} i/2^i
= 2^h { 1/2 + 2/4 + 3/8 + 4/16 + 5/32 + ---- h/2^h }
≤ 2^h { 1/2 + 2/4 + 3/8 + 4/16 + ---- 0 }
lim_{h→∞} h/2^h = 0



$$= \left[\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \frac{5}{32} + \cdots \infty\right] \quad \times \frac{1}{2}$$

$$= \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots\right) + \left(\frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \cdots\right)$$

$$+ \left(\frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \cdots\right) + \left(\frac{1}{16} + \frac{1}{32} + \cdots\right) + (\cdots)$$

r_4 $\quad$ $1/8$

$a + ar + ar^2 + ar^3 + \cdots \infty \quad |r|<1$ } Geometric series sum

$a = 1/2, \; r = 1/2,$

$\Rightarrow \left(\frac{a}{1-r}\right)$.

$= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots$ } geometric Series $a = 1, \; r = 1/2, \; \frac{a}{1-r}$

$= \frac{1}{1-1/2} = 2$

So

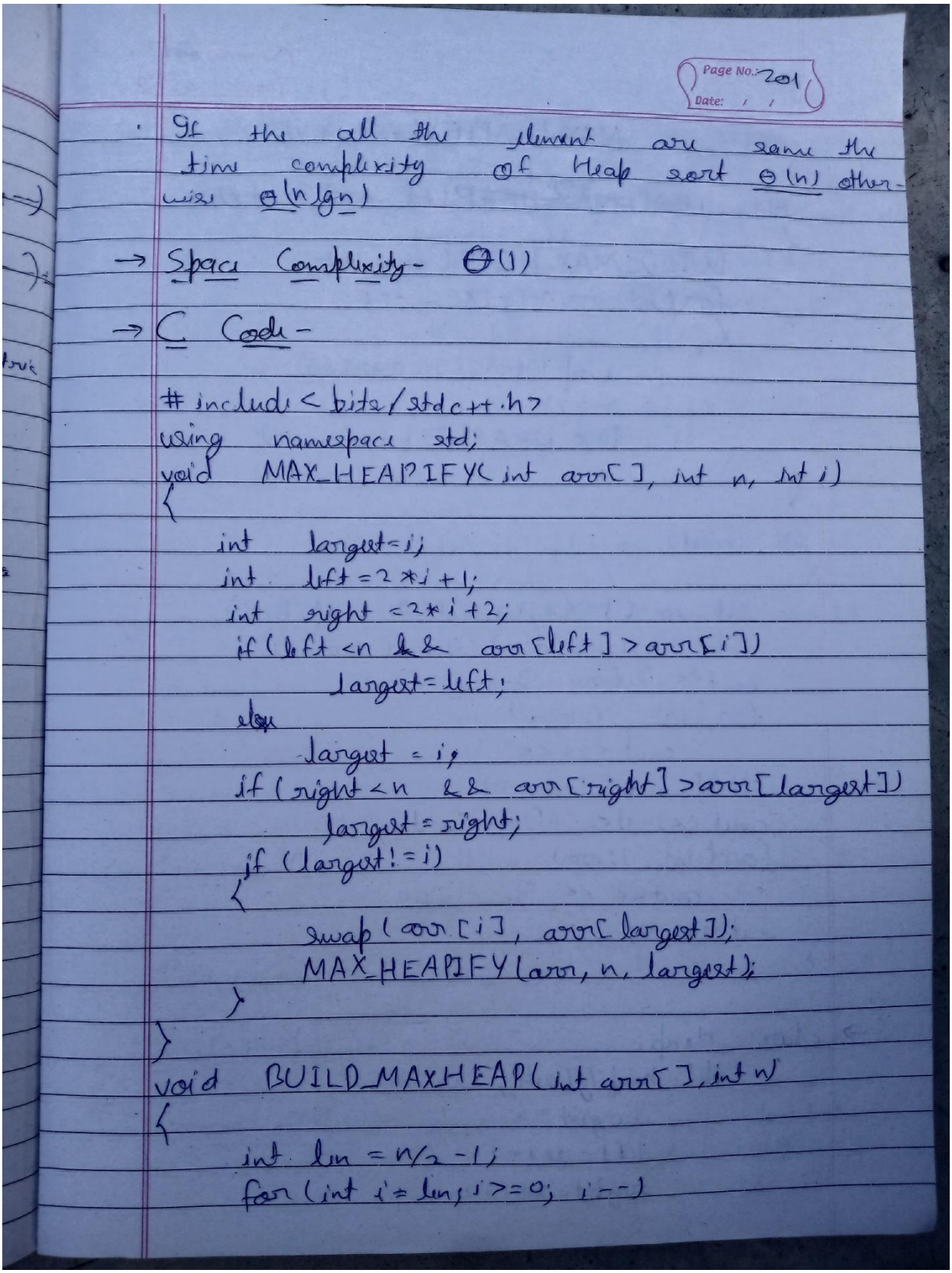
$\leq 2^h \left\{\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots\right\}$

$= 2^h \cdot 2 = O(h)$. $\quad 2^h = O(h)$.

Heap_Sort(A) -

1) Build_Max_Heap(A) — $O(n)$
2) n = effective size of heap.
3) a) swap A[i] with A[n]
   b) n = n-1
   c) Max_Heapify(A, 1)
   } — $O(\lg n)$
4) Repeat 3 until n = 0.

Time Complexity - $n + n \lg n = \Theta(n \lg n)$.



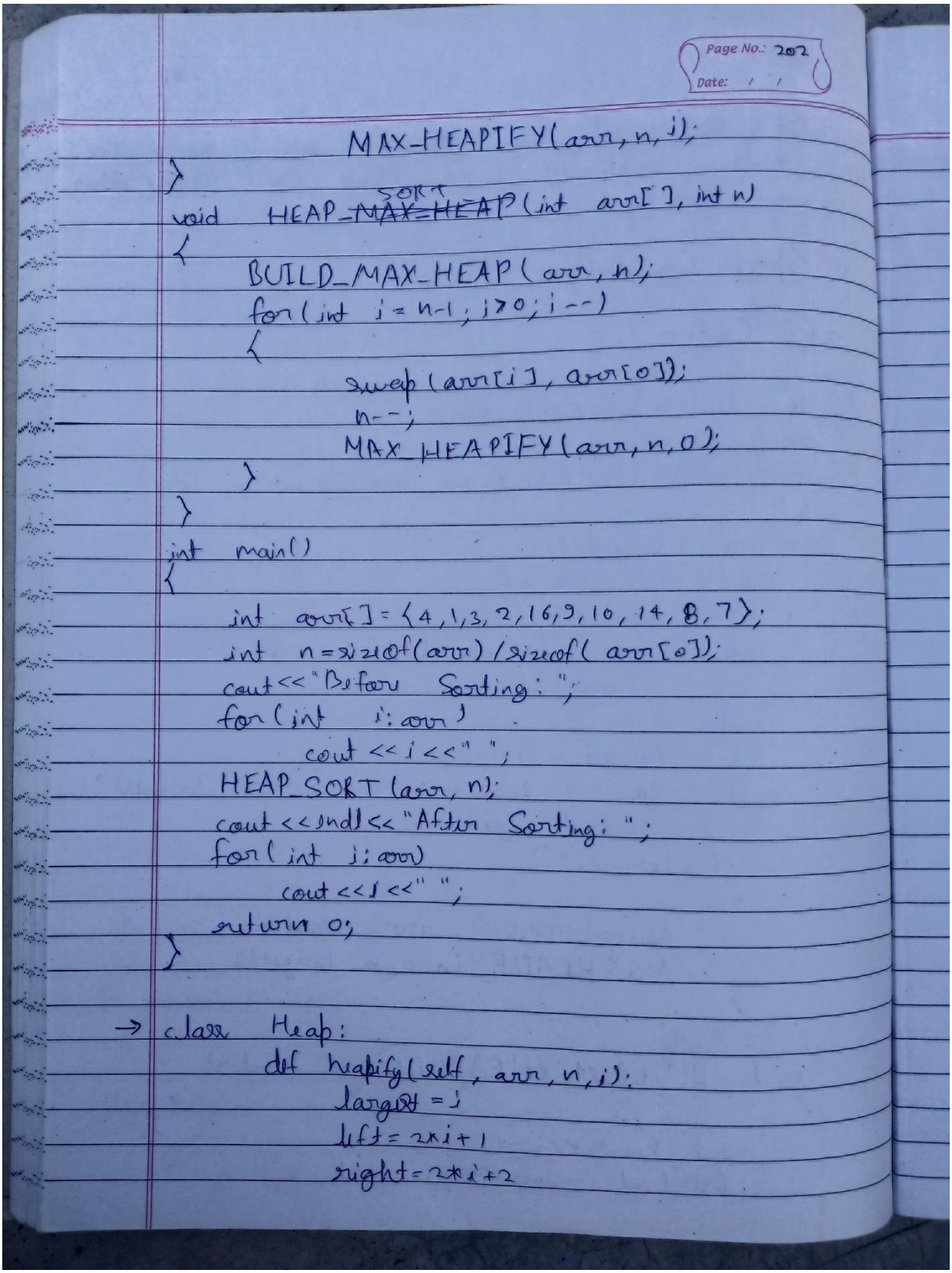
Page No.: 201
Date: / /

- If the all the element are same the time complexity of Heap sort $\Theta(n)$ otherwise $\Theta(n \lg n)$

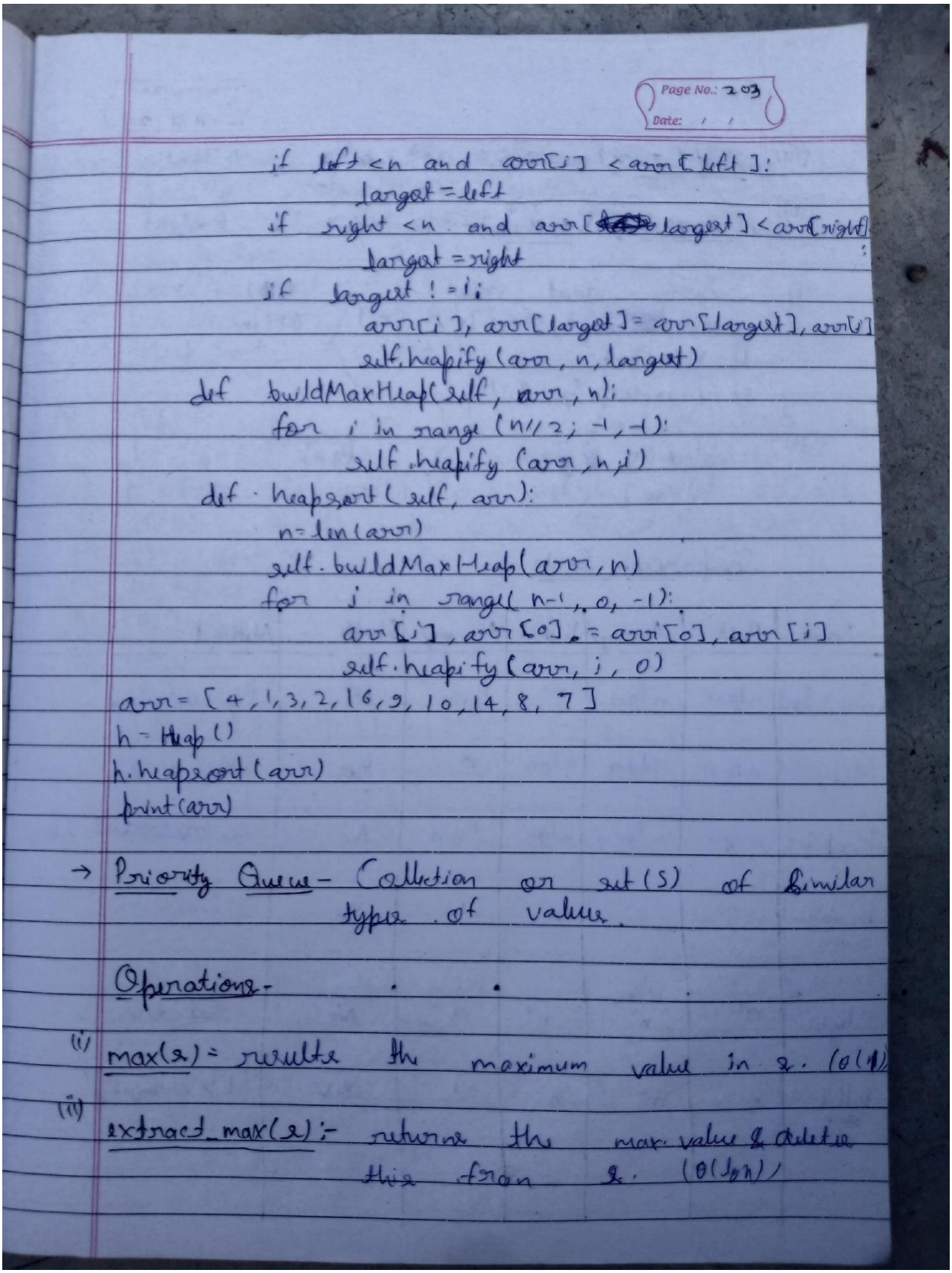
→ Space Complexity - $\Theta(1)$

→ C Code -

```
#include <bits/stdc++.h>
using namespace std;
void MAX_HEAPIFY(int arr[], int n, int i)
{
    int largest=i;
    int left=2*i+1;
    int right=2*i+2;
    if(left<n && arr[left]>arr[i])
        largest=left;
    else
        largest=i;
    if(right<n && arr[right]>arr[largest])
        largest=right;
    if(largest!=i)
    {
        swap(arr[i], arr[largest]);
        MAX_HEAPIFY(arr, n, largest);
    }
}
void BUILD_MAXHEAP(int arr[], int n)
{
    int len=n/2-1;
    for(int i=len; i>=0; i--)
```



```
        MAX_HEAPIFY(arr, n, i);
    }
    void HEAP_SORT(int arr[], int n)
    {
        BUILD_MAX_HEAP(arr, n);
        for(int i = n-1; i>0; i--)
        {
            swap(arr[i], arr[0]);
            n--;
            MAX_HEAPIFY(arr, n, 0);
        }
    }
    int main()
    {
        int arr[] = {4,1,3,2,16,9,10,14,8,7};
        int n = sizeof(arr)/sizeof(arr[0]);
        cout<<"Before Sorting: ";
        for(int i: arr)
            cout<<i<<" ";
        HEAP_SORT(arr, n);
        cout<<endl<<"After Sorting: ";
        for(int i: arr)
            cout<<i<<" ";
        return 0;
    }

→ class Heap:
      def heapify(self, arr, n, i):
          largest = i
          left = 2*i+1
          right = 2*i+2
```



Page No.: 203
Date: / /

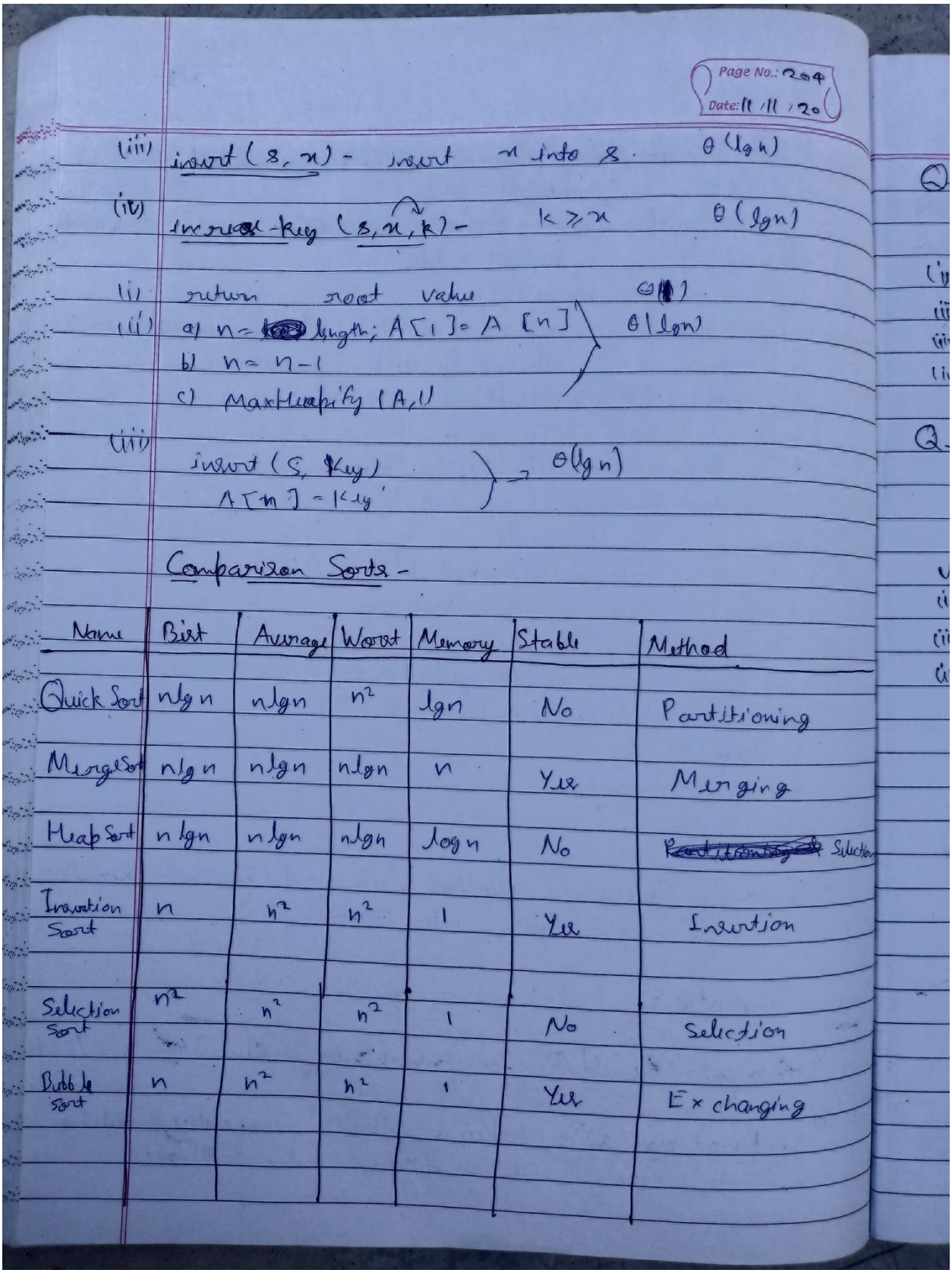
```
if left < n and arr[i] < arr[left]:
    largest = left
if right < n and arr[largest] < arr[right]:
    largest = right
if largest != i:
    arr[i], arr[largest] = arr[largest], arr[i]
    self.heapify(arr, n, largest)
def buildMaxHeap(self, arr, n):
    for i in range(n//2, -1, -1):
        self.heapify(arr, n, i)
def heapsort(self, arr):
    n = len(arr)
    self.buildMaxHeap(arr, n)
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        self.heapify(arr, i, 0)
arr = [4, 1, 3, 2, 16, 9, 10, 14, 8, 7]
h = Heap()
h.heapsort(arr)
print(arr)
```

→ Priority Queue - Collection or set(s) of similar type of value.

Operations -

(i) max(x) = results the maximum value in x. (O(1))

(ii) extract_max(x) :- returns the max. value & delete this from x. (O(log n))



Page No.: 204
Date: 11/11/20

(iii) insert (S, x) - insert x into S. $\Theta(\lg n)$

(iv) increase-key (S, x, k) - $k \geq x$ $\Theta(\lg n)$

(i) return root value

(ii) a) n = length; A[1] = A[n]
b) n = n - 1
c) MaxHeapify (A, 1)
$\Theta(\lg n)$

(iii) insert (S, key)
A[m] = key
$\Theta(\lg n)$

Comparison Sorts -

Name	Best	Average	Worst	Memory	Stable	Method
Quick Sort	$n \lg n$	$n \lg n$	n^2	$\lg n$	No	Partitioning
Merge Sort	$n \lg n$	$n \lg n$	$n \lg n$	n	Yes	Merging
Heap Sort	$n \lg n$	$n \lg n$	$n \lg n$	$\log n$	No	Selection
Insertion Sort	n	n^2	n^2	1	Yes	Insertion
Selection Sort	n^2	n^2	n^2	1	No	Selection
Bubble Sort	n	n^2	n^2	1	Yes	Exchanging

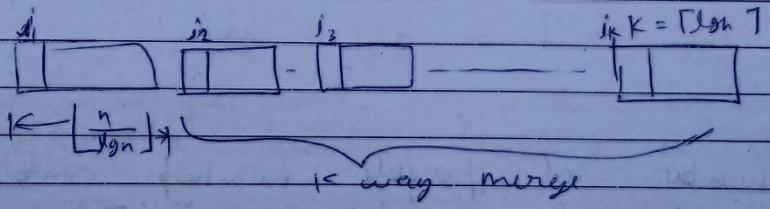
Page No.: 205
Date: / /

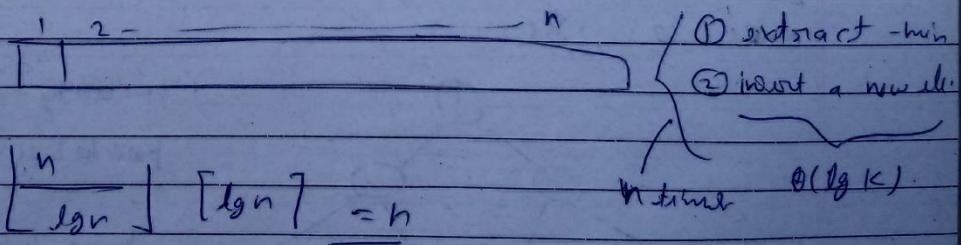
Q- Which one of the following in place sorting algorithms needs the minimum number of swaps?

(i) Quick Sort — $O(n^2)$
(ii) Insertion sort — $O(n^2)$
(iii) Selection Sort — $O(n)$ swaps (✓)
(iv) Heap Sort — $O(n \lg n)$

Q- Suppose there are $\lceil \lg n \rceil$ sorted lists of $\lfloor n / \lg n \rfloor$ elements each. The time complexity of producing a sorted list of all these elements is (Hint: Use a Heap data Structure)

(i) $O(n \lg \lg n)$ (✓)
(ii) $\Theta(n \lg n)$
(iii) $\Omega(n \lg n)$
(iv) $\Omega(n^{3/2})$





$\lfloor \frac{n}{\lg n} \rfloor \lceil \lg n \rceil = n$

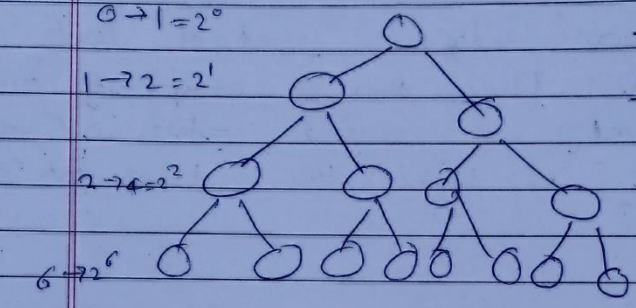
$\Theta(n \lg k) \quad k = \lceil \lg n \rceil$

$\Theta(n \lg \lg n)$

Page No.: 206
Date: / /

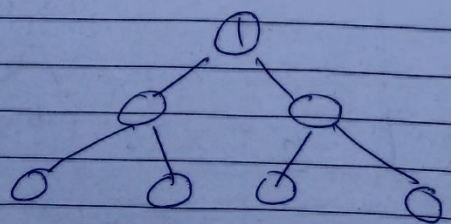
Q- In a heap with n element with the smallest element at the root. The 7th smallest element can be found in time.

(i) $O(n \lg n)$
(ii) $O(n)$
(iii) $O(\lg n)$
(iv) $O(1)$ (✓)



$2^7 - 1 = 127$ ← 7th smallest element will be one of these

Q- The number of possible min-heap containing each value from $\{1, 2, 3, 4, 5, 6, 7\}$ exactly once is, 80



Total possibility =
$= 2 \times 5 \times 4 \times 1 \times 2$
$= 80$

Page No.: 207
Date: / /

Q- An operator delete (i) for a binary heap data structure is to be designed to delete the item in the ith node. Assume that the heap is implemented in an array and i refers to the ith index of the heap. If the heap tree has depth d (number of edges on the path from the root to the farthest leaf). Then what is the time complexity to re-fix the heap efficiently after removal of the element?

(i) O(1)
(ii) O(d) but not O(1) (✓)
(iii) $O(2^d)$ but not O(d)
(iv) $O(d2^d)$ but not $O(2^d)$

delete O(height).
height = d.
O(d). ✓

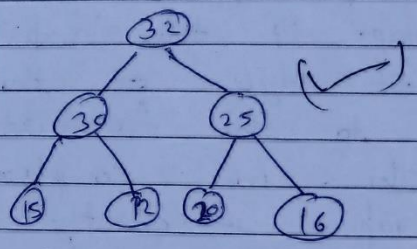
Q- The tighat lower bound on the number of comparisons in the worst case. For comparison-based sorting is of the order of -

(i) n
(ii) n^2
(iii) $n \lg n$ (✓)
(iv) $n \lg^2 n$.

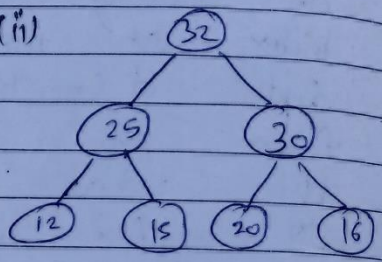
Page No.: 208
Date: / /

Q The elements 32, 15, 20, 30, 12, 25, 16 are inserted one by one in the given order into a Max Heap. The resultant Max Heap is –

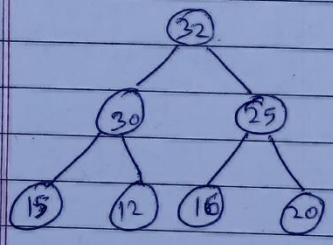
(i)



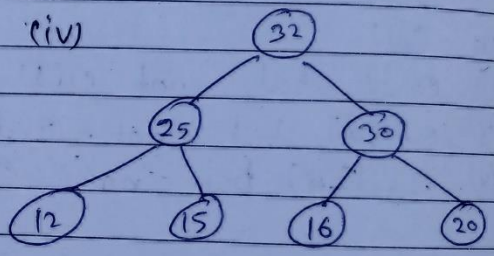
(ii)



(iii)

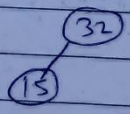


(iv)

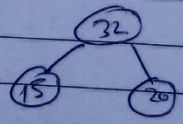


Insert - 32

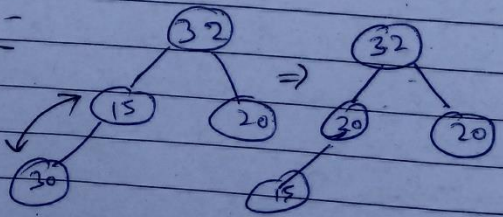
Insert 15 -



Insert 20 -

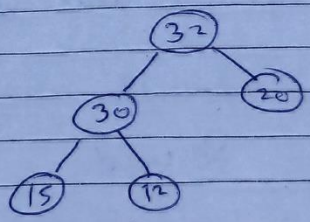


Insert 30 -

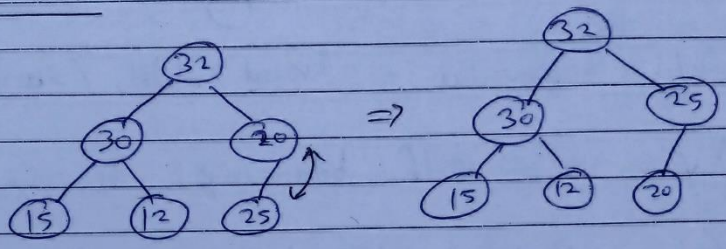


Page No.: 209
Date: / /

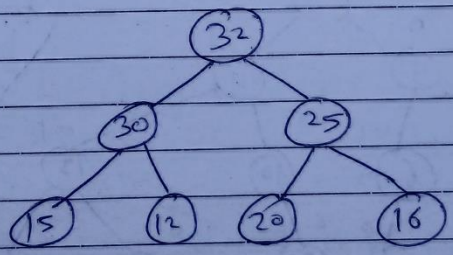
Insert 12 -



Insert 25 -



Insert 16 -



Q. A priority queue is implemented as a Max-Heap Initially, it has 5 elements. The level-order traversal of heap is - 10, 8, 5, 3, 2. Two new element 1 and 7 are inserted into the heap - in that order. The level-order traversal of the heap after the insertion of the element is -

(i) 10, 8, 7, 3, 2, 1, 5 (✓)
(ii) 10, 8, 7, 2, 3, 1, 5
(iii) 10, 8, 7, 1, 2, 3, 5
(iv) 10, 8, 7, 5, 3, 2, 1