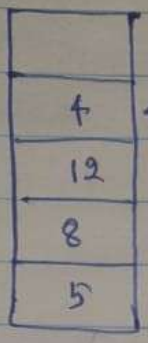
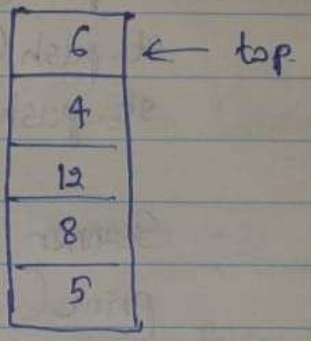
DSA - 2022 Regular.

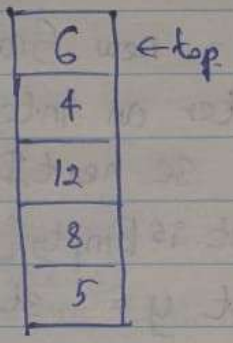
Question 1



(i) myStack.push(6)

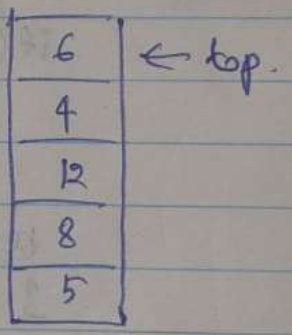


(ii) myStack.peek()
returns 6 →

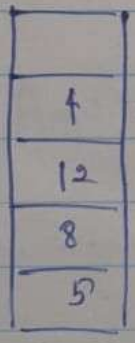


(iii) myStack.push(2)

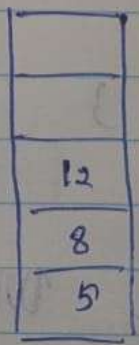
error → Stack is full
Cannot push element to the stack. →



(iv) myStack.pop();



(v) myStack.pop.



(b.) class StackApp {
public static void main (String [] args) {

StackX st = new StackX(7);

st.push(20);
st.push(43);
st.push(12);
st.push(37);

Scanner sc = new Scanner (System.in);
print("Enter an Integer: " ~~+ sc.nextInt();~~
int i = sc.nextInt();
while (! st.isEmpty()) {
int y = st.pop();

if (y == i) {
System.out.println(" The number is exists in the array;")
return true; break;
}
else {
return false;
}

}
}
}

(c). (1) True. When It comes to stack, we can do the insert and remove from the top only, and ~~it has~~ stack ~~class~~ has stackArray size when we creating a stack. We cannot insert any value when the stack is full. Queue is also fixed sized. we cannot insert any element to the QueueArray when the no of Items of the queueArray equal to it maxSize.

And It leads to wasted space. In some cases array implementation may result in wasted space. If the capacity of the array is significantly larger than the no of Items stored in the array.

(ii). false

Circular queue allow to insert items from the rear and remove done from the front.
After an element is removed front is incremented by one.
After remove an element rear is increment by one.

(d.)

	Linear Queue	Circular Queue
boolean isEmpty()	if(nItems==0) return true; else return false;	if (nItems==0) return true; else return false;
boolean isFull().	if (rear == maxSize-1) return true; else return false;	if (nItems == maxSize) return true; else return false.

Question 2

(a) (i) first → [100] ← (insertFirst(100));

(ii) first → [50] → [100]

(iii) first → [10] → [50] → [100]

(iv) first → [10] → [50] → [100] → [200]

(v) first → [10] → [50] → [100] → [200]

first → [50] → [100] → [200]

(b). public int countVacantBeds() {
int count = 0;
Link current = first;
while (! curre

(b) (i) public int CantVacantBeds() {
int count = 0;
PatientBed current = first;

while (current ! = null) {
if (current. vacant == 1) {
count++;
}
else {
current = current. next;
}
return count;
}
}

(ii) findVacant() → first vacant PatientBed.
no vacant beds → return NULL.

class Main {
public static void main (String [] args) {

patientBed vacantP = new LinkList
LinkList List = new LinkList();

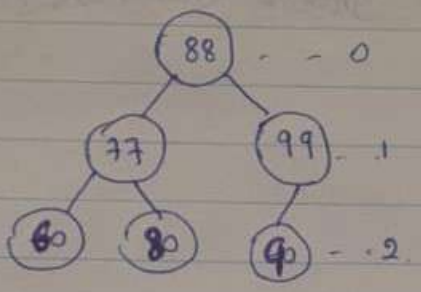
patientBed vacantBed = List. findVacant();

if (vacantBed ! = null) {
vacantBed. assignBed();
System. out. println (" Assigned Bed No: " + vacantBed.
bedNo);

else {
System.out.println("Patient beds are not available");
}
}
}

(c) (i) 88, 99, 77, 80, 90, 60.

~~Left to Right~~

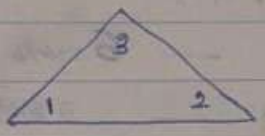


(ii) ~~height~~

no of depths = height

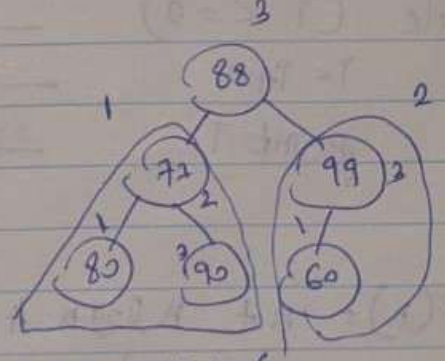
$h = 2$

(iii) Post Order



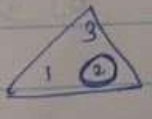
left, Right, Root

```
private void postOrder (Node localNode){
        postOrder (localNode.leftChild);
        postOrder (localNode.rightChild);
        localNode.displayNode();
}
```



80 90 77 60 99 88

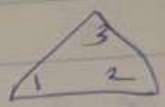
(iv)
```
public class displayGreaterThanRoot(){
        displayGreaterThanRoot(root);
}
```



(iv) public void displayGreaterThanRoot(Node localNode){

```
if (root == null){
    System.out.println("Tree is empty");
    return;
}
else{
    System.out.println("Node greater than root");
    inOrder(root);
}
```

}



Question 3

(a) (i) int j = 0 — 1 = 1

for (i = 1 to 5) — 6 6 5 } = 17

(*) print i — 1 × 5 = 5

while (j < 5) — 5 × +4 = 10

j = j+1 — 2 × 5 = 10

print j — 1 × 5 = 5

Sum → 48

i = 2 ×
i = 3 ×
i = 0 ×
i = 5 ×

(ii) int i = 1 — 1

while (i <= 0) — IF

i = i + 2 — x

print i — x

2

(b) (i) $T(n) = n + n\log n + n^2 + 5$

$O(n^2)$

(ii) $T(n) = 2n + 10$

$O(n)$

(iii) $T(n) = 3^n + n! + 4n^2 + n^n - 8$

$= O(n^n)$

(c) (i) $T(n) = 2T(n/2) + O(n)$

(ii) In Merge Sort Algorithm partition is always balanced.

(d) A = {8, 2, 10, 3, 5}

MERGESORT (A, p, r)

8, 1, 8

1 < 8

1. If p < r

2. q = [(p+r)/2] = (1+8)/2 = 4

3. MERGESORT (A,

(d) A = {8, 2, 10, 3, 5}

PARTITION (A, p, r)

5, 1, 8

1. x = A[r]
   x = A[5] = 5

2. i = p-1
   i = 1-1 = 0

3. for j = p to r-1
   for j = 1 to (5-1) 4

   j=1

4. If A[j] ≤ x
   A[1] ≤ 5
   8 ≤ 5 ✗

(i) j=2

4. if A[2] ≤ 5
   2 ≤ 5 ✓

5. i = i+1
   i = 0+1 = 1

6. exchange A[i] with A[j]
   A[1] ↔ A[2]
   8 ↔ 2

2	8	10	3	5

j=4

7. exchange A[i+1] with A[r]
   A[3] with A[5]
   10 ↔ 5.

2	3	5	8	10

8. return i+1
   2+1 = 3
   return 3

j=3

4. if A[3] ≤ 5
   10 ≤ 5 ✗

j=4

4. if A[4] ≤ 5
   3 ≤ 5 ✓

5. i = i+1
   i = 1+1 = 2

6. exchange A[i] with A[j]
   A[2] ↔ A[4]
   8 ↔ 3

2	3	10	8	5

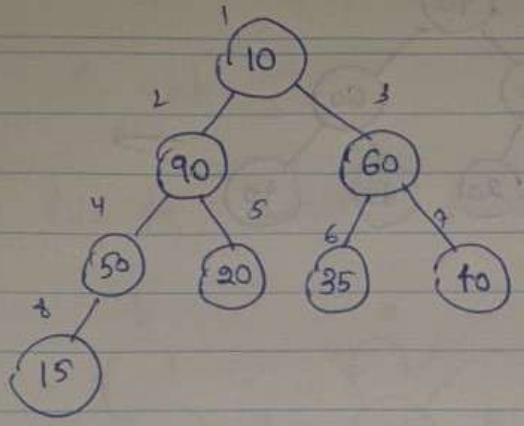
Question 4

(a)(i) for i = ⌊A.length/2⌋ downto 1

In this BUILD_MAX_HEAP function only calling for the parent nodes (non-leaf nodes)

(11) A

10	90	60	50	20	35	40	15



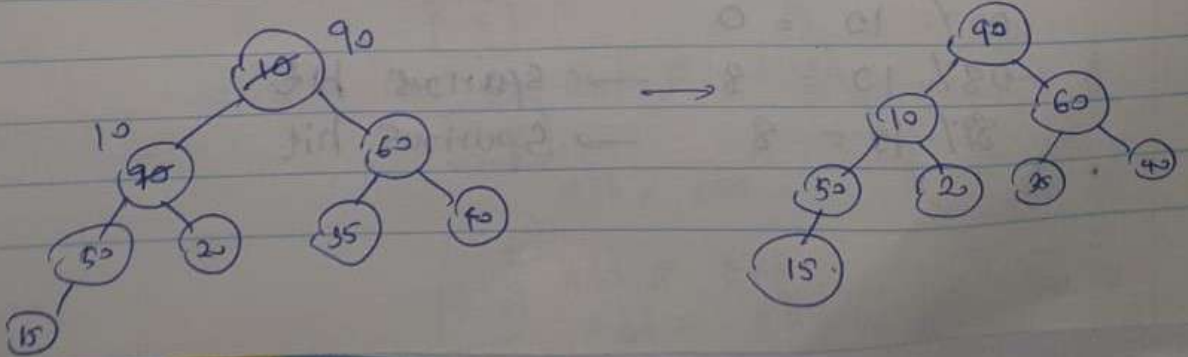
left child(i) = 2i. 2×1 = 2

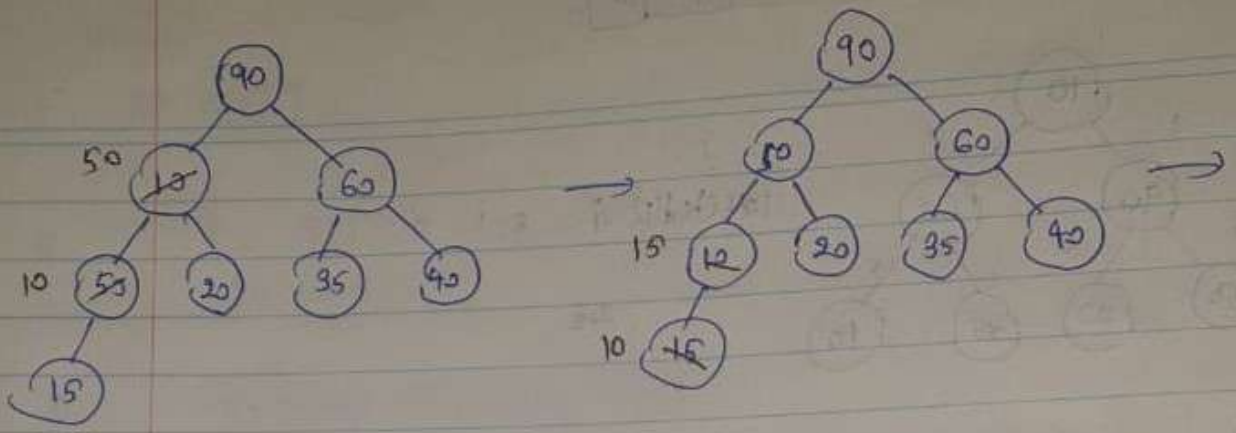
2×2

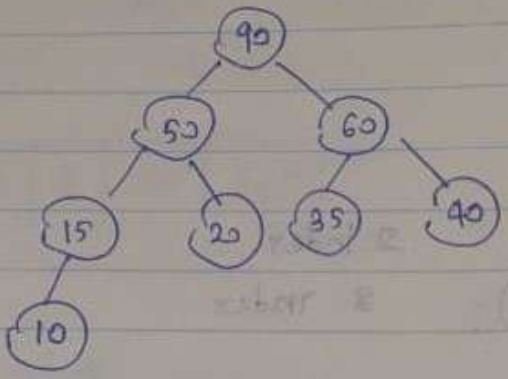
8, 1

MAX_ HEAPIFY (A, i)

1. l = LEFT_CHILD(i); 2 index.
2. r = RIGHT CHILD(i); 3 index
3. if l ≤ heap_size[A] and A[l] > A[i]
   2 ≤ 8 && 90 > 60 ✓
4. largest = l
   largest = 2
5. else largest = i; ×
6. if r ≤ heap_size[A] and A[r] > A[largest]
   3 ≤ 8 && 60 > 90 ×
7. largest = r; ×
8. if largest ≠ i
   2 ≠ 1. ✓
9. exchange A[i] ↔ A[largest]
   10 ↔ 90
10. MAX_HEAPIFY (A, largest)
    MAX_HEAPIFY (B, 2)







(b.) modulo $q = 10$.

$T = 90288320088$ $\quad p = 28$

$P \% q = 28 \% 10 = 8$

~~902 % 10 = 2~~

~~028 % 10 = 8~~ → valid

90 % 10 = 0

02 % 10 = 2

28 % 10 = 8 → valid

88 % 10 = 8 → Spurious hit.

83 % 10 = 3

32 % 10 = 2

28 % 10 = 8 → valid

80 % 10 = 0

00 % 10 = 0

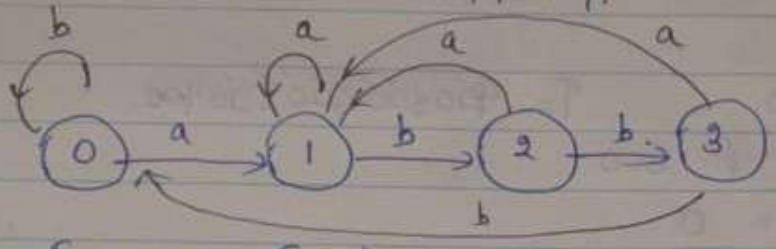
08 % 10 = 8 → Spurious hit

88 % 10 = 8 → Spurious hit

Valid hits = 2
Spurious hits = 3 //

① P = abb

No of states = Characters inside pattern + 1
= 3 + 1 = 4.



{a, b}

T = a ↵ T = a b a x T = a b a x T = a b b a
T = b x T = a b ↵ T = a b b ↵ T = a b b b

② Native - String - Matcher (T, P)

abb = P

1. n = length [T] 10
2. m = length [P] 3
3. for s = 0 to n − m (7)

if P[1..m] = T[s+1 .. s+m]

s = 0 abb ≠ aAb → 2
s = 1 abb ≠ Abb →
s = 2 abb ≠ bbb →
s = 3 abb ≠ bba →
s = 4 abb ≠ baa →
s = 5 abb ≠ aaa → 2
s = 6 abb ≠ aab → 2
s = 7 abb ≠ aba → 2

Comparisons = 2+1+1+1+1+2+2+2 = 12

2022 - June

Question 1

(1) modulo q = 100. T = 900800600300900

P = 600

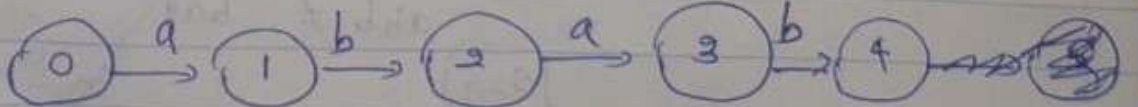
600 % 100 = 0

900 % 100 = 0 → Spurious hit
008 % 100 = 8
080 % 100 = 80
800 % 100 = 0 → Spurious hit
006 % 100 = 6
060 % 100 = 60
600 % 100 = 0 → valid
003 % 100 = 3
030 % 100 = 30
300 % 100 = 0 → Spurious hit
009 % 100 = 9
090 % 100 = 90
900 % 100 = 0 → Spurious hit

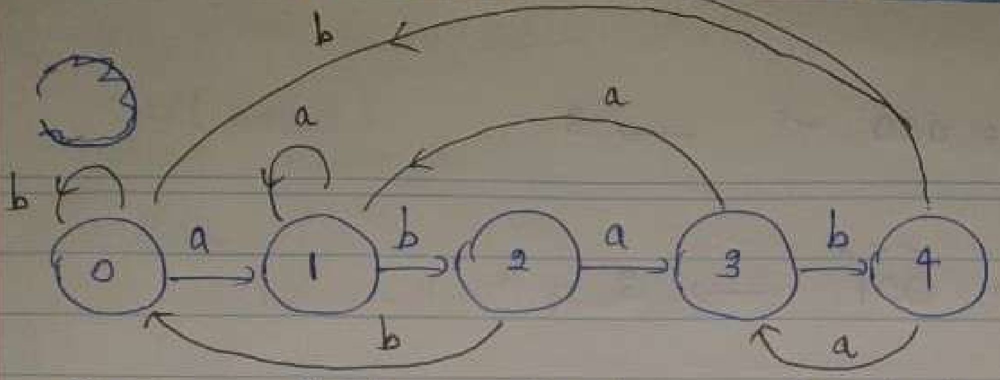
valid = 1
spurious = 4

(2) P = abab

No of states → Characters inside pattern + 1
= 4 + 1 = 5



(2)



T= a ✓ T= ab~~b~~a x T= aba ✓ T= abaa x T= abab[a

T= b x T= ab ✓ T= abb ✓ T= abab ✓ T= ababb

a[bb aba[a ✓ T= ab[a[b[b

T= ab[a[b[a

(3) Naive-String-Matcher (T, P)

1. n ← length(T) = 9
2. m ← length(P) = 3

3. for s ← 0 ~~s~~to n-m
   s ← 0 to 9-3=6

4. do if [1.. m] = T[s+1 .. s+m]

s=0

[1..3] = T[1..3]

000 = 100 x → 1

s=1

000 = 000 ✓ → 3

s=2

000 = 001 → 3

s=3

000 = 010 → 2

s=4

000 = 100 x → 1

S=5

000 = 000 ✓ ⟶ 3

S=6

000 = 001 ⟶ 3

(a) 3+3+3+3+2+1+1

12+4 = 16

(b) Text size = n

Pattern size = m.

~~In best case there is no any spurious hits or valid hits~~

In best case shifts should be 1.

Total number of best shifts = 2

Total number of comparisons = 2 = total number of shifts

= n-m+1

The complexity = O(n-m+1)

Question 2

(1) Pow(x, n) = x^n Pow(x, n-1) = $x^{(n-1)}$

$x^n = x \times x^{n-1}$

Pow(x, n) = $x \times x^{n-1}$

Pow(x, n) = x × Pow(x, n-1)

(b). int Pow (int x, int n) {

```
    if(n==0)
        return 1
    else
        return (x* pow (x, (n-1)));
}
```

(c). int Pow (int x, int n) { $\longrightarrow T(n)$

```
    if (n==0)        ⟶ c1
        return 1     ⟶ c2
    else
        return (x*  pow (x, (n-1)));
                ⟶ c3        ⟶ T(n-1)
}
```

$n>0$

recurrence equation $T(n) = c_1 + c_3 + T(n-1)$

$= c + T(n-1)$

$= O(n)$

(2).

Sum ← 0	— 1	→ 1
for i ← n downto 0	= (n+1)	→ 3n+5
Sum = sum+1	= (n+2)	→ 2n+2
	(n+2)	
	2(n+1).	→ 5n+8

(3).

1	2	3	4	5	6.
28	15	1	30	0	8.

e.g. 1, 6

Procedure QUICKSORT (A, p, r)

1. if p < r 1 < 6
2. then q ← PARTITION (A, p, r) 1, 6
3. QUICKSORT (A, p, q-1) A, 1,
4. QUICKSORT (A, q+1, r)

Procedure PARTITION (A, p, r)

1. $x \leftarrow A[r]$
   $x \leftarrow A[6]$
   $x \leftarrow 8$
   $x = 8$

2. $i \leftarrow p-1$
   $i = 1-1 = 0$

3. for $j \leftarrow p$ to $r-1$
   for $j = 1$ to 5

   j=1

4. do if $A[j] \leq x$
   $A[1] \leq 8$
   $28 \leq 8$ ✗

   j=2

   do if $A[j] \leq x$
   $A[2] \leq 8$
   $15 \leq 8$ ✗

   j=3

   do if $A[j] \leq x$
   $A[3] \leq 8$
   $1 \leq 8$ ✓

5. then $i \leftarrow i+1$
   $i = 0+1 = 1$

6. exchange $A[i] \leftrightarrow A[j]$
   $A[1] \leftrightarrow A[3]$
   $28 \leftrightarrow 1$

1	16	28	30	0	8

7. ~~exchange~~ $A[i+1] \leftrightarrow A[r]$

$A[2] \leftrightarrow A[6]$

$15 \leftrightarrow 8$

1	8	28	30	0	15

8. ~~return i+1~~

return 2

4. ~~exchange~~ ~~A~~ for $j=4$

do if $A[1] \leq x$

$A[4] \leq 8$

$30 \leq 8$ ×

4 $j=5$

do if $A[i] \leq x$

$A[5] \leq 8$

$0 \leq 8$ ✓

5. then $i = i+1$

$i = 1+1 = 2$

6. exchange $A[i] \leftrightarrow A[j]$

$A[2] \leftrightarrow A[5]$

$15 \leftrightarrow 0$.

1	0	28	30	15	8

7. exchange $A[i+1] \leftrightarrow A[r]$

$A[3] \leftrightarrow A[6]$

$28 \leftrightarrow 8$

1	0	8	30	0	

8. return i+1

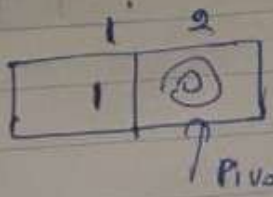
return 3

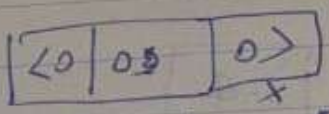
1	0	8	30	15	28

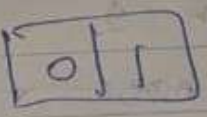
diagrammatic way.

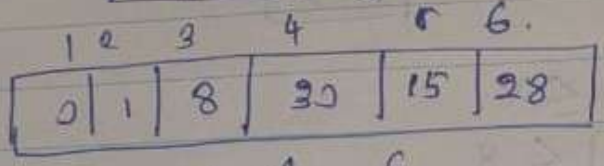
③ QUICKSORT (A, p, q-1)

A, 1, 2



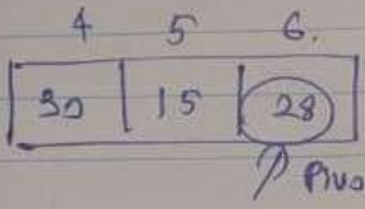


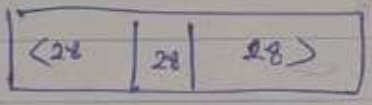


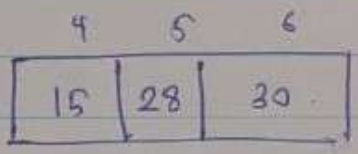


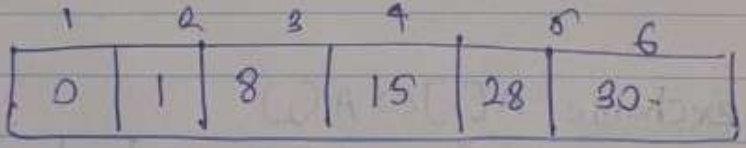
A, 4, 6

④ QUICKSORT (A, q+1, r)

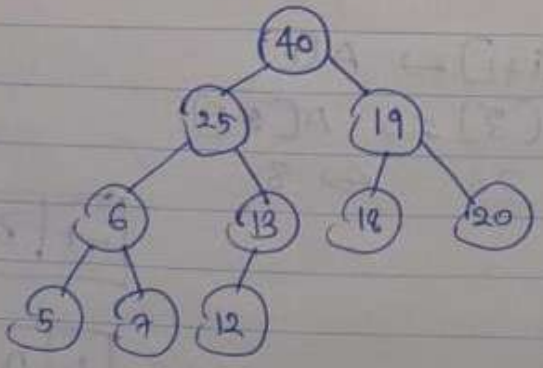








⑤ <40, 25, 19, 6, 13, 18, 20, 5, 7, 12>

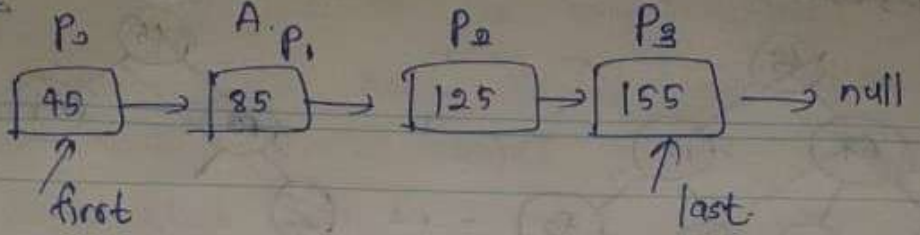


No

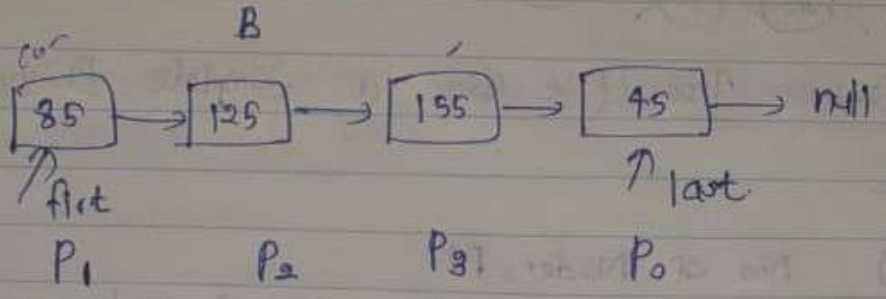
Because the root 6 and 19 are not the maximum values when it compared with their right and left child nodes.

Question 3

① A



B



A LinkList

```
LinkList P3 = last;
LinkList P0 = first;
LinkList P1 = P0.next;
LinkList P2 = P1.next;
LinkList P3 = P2.next;
P3.next = null
```

B LinkedList

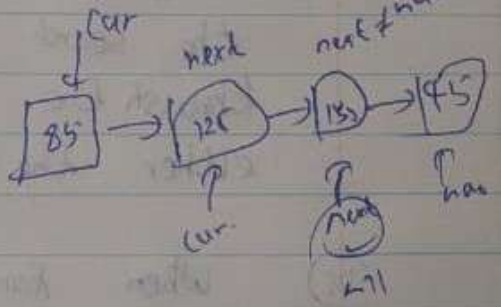
```
first = P1
P1.next = P2
P2.next = P3
P3.next = P0
P0 = last.
P0.next = null.
```

② public deleteLast(int key) {

```
Link cur = first;
```

③ public boolean deleteLast() {

```
Link cur = first;
```

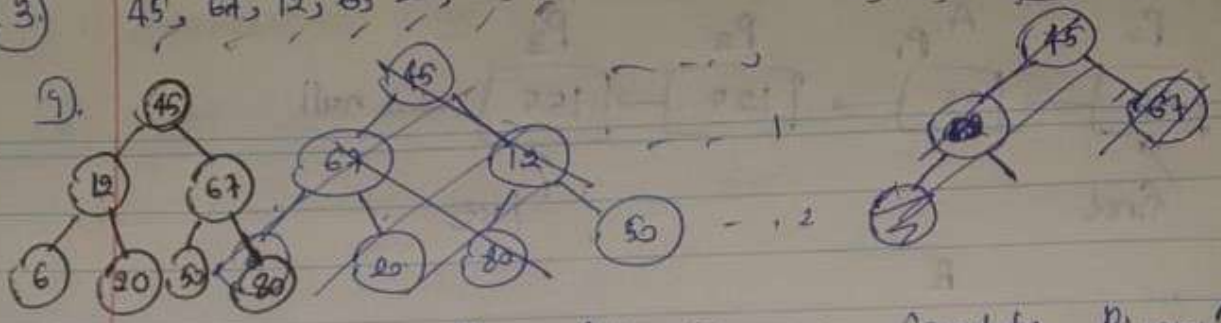



```
while (current.next.next != null) {
    current = current.next;
}
current.next = null;
}
```

(3) 45, 67, 12, 6, 20, 80, 50.

6, 12, 20, 45, 50, 67, 80

(i).



(ii). Full Binary Tree (it also a Complete Binary Tree)

(iii) 2

(iv) No of Nodes = N

Searching is depend on the No of Nodes inside the binary tree.

$T(N) = O(\log N)$

(v) This is a balanced case then it's $T(n) = O(\log N)$. → best case

(Worst case → $O(N)$)

(v). edges = N

$O(N)$

worst case

best case → target value at the head

$O(1)$

(In this case target value is located somewhere within the linked list or not present at all, so, the search operation would need to traverse through the linked list, examing each link until either the target value is find or not).

(vi) When for a binary search Tree, the average and worst case time complexity for searching is $O(\log N)$ in a balanced tree. This means that as the number of Nodes increases, runtime also increase. This logarithmic growth allows binary search tree to efficiency search for values even with large N values.

In Linked List, the worst-case time complexity for searching is O(N). Number of Link ↑ the running times grows linearly

The linear growth makes Linked Lists less efficient for search when dealing with large N values compared to binary search trees.

∴ Binary Search tree is faster than linked list when we searching a value

Question 4

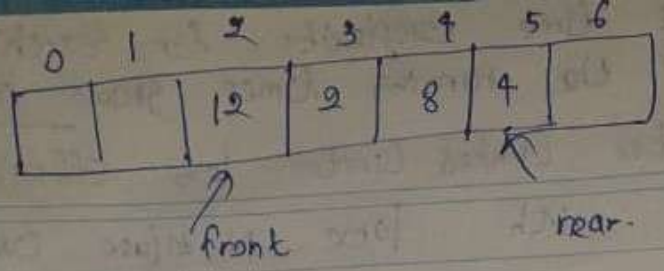
(1.)

Stack.	Queue.
Follow LIFO principle	Follow FIFO principle.
Size is Fixed and linear data Structure.	Size is Fixed and linear data structure.

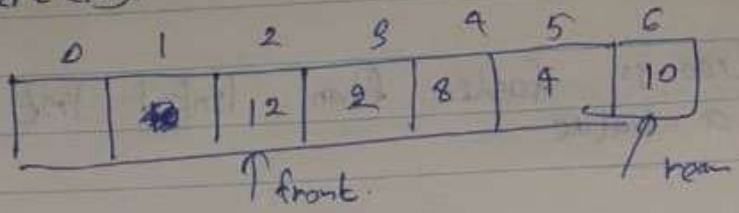
(2.)

```
public void deleteMiddle() {
    int count = 0;

    while (s.isEmpty()) {
        s1.push(s.pop());
        count = count+1;
    }
    int middle = count/2;

    for (i = 0; i < middle; i++) {
        s.push(s1.pop());
    }
    s.push(s1.pop());

}
```

3.

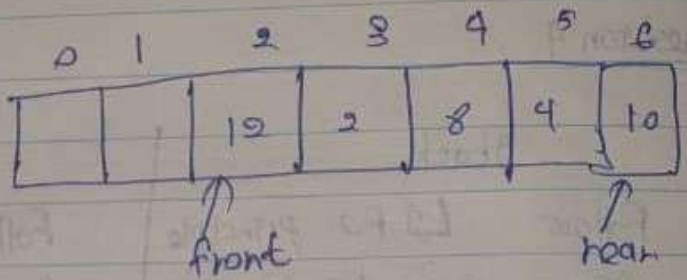


(I) Insert (10)

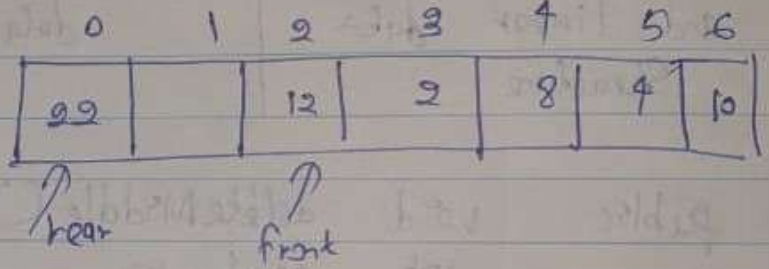


(II) peek Front ()

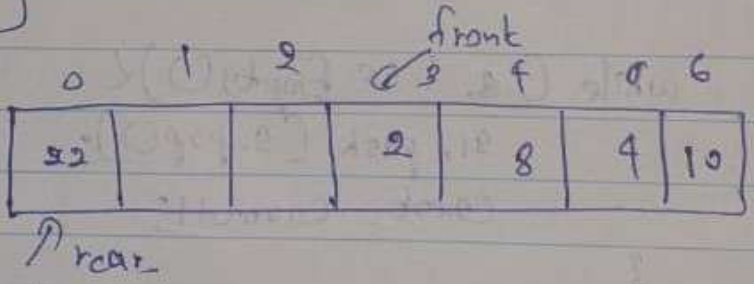
return 12



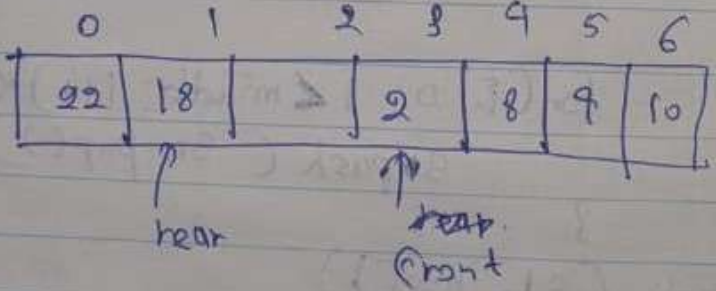
(III) Insert (22)



(IV) remove ()



(V) insert (18)



(5) public void calmean ( ) {
if (nItems == 0) {
System.out.println (" Queue is empty ");
}
else {

(5) public void calculate Mean And Insert ( ) {

Queue tempQueue = new Queue ();
double sum = 0.0;
int count = 0;

while (! queue. isEmpty ( )) {

(5) public void calMeanAndInsert (Queue <Double> q) {

Queue <Double> tempQueue = new LinkedList <> ();
double sum = 0.0;
int count = 0;

while (! q. isEmpty ( )) {
double value = q. remove ();
sum += value;
count++;
tempQueue. insert (value);
}

double mean = sum / count;
q. insert (mean);

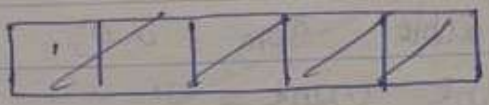
while (! tempQueue. isEmpty ( )) {
q. insert (tempQueue. remove ( ));
}
}

2. ~~Calculate~~
CalMeanAndInsert(2)

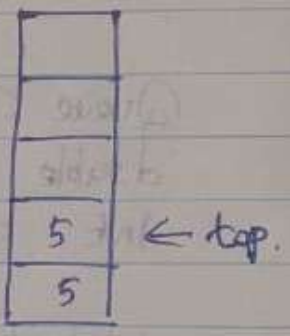
2019 - 3rd October

Question 01

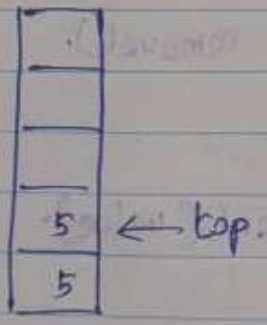
(a) (I) (i) s. push (5);



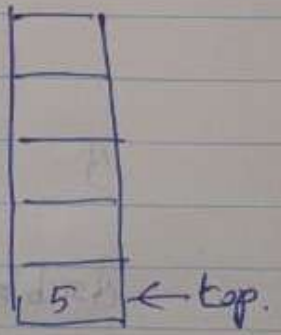
(ii) s. push (s. peek());
s. peek() → returns 5
s. push (5)



(iii) s. push (s. pop());
s. pop() → 5
s. push (5);



(iv) s. pop();

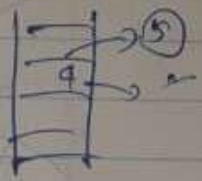


(II)
```
public void push (int j) {
    if (top == maxSize-1)        ——①
        System.out.println ("stack is full");
    else {
        stack[++top] = j;        ——②
    }
}
```

① and ② ~~are~~ edited.

(iii)

```
public StackMain{
    public static void main (String [] args){
        StackX s1 = new StackX(5);
        StackX s2 = new StackX(5);
        while (!s1.isEmpty()){
            int value = s1.pop();
            if (value%2 == 0)
                s2.push(value);
        }
    }
}
```

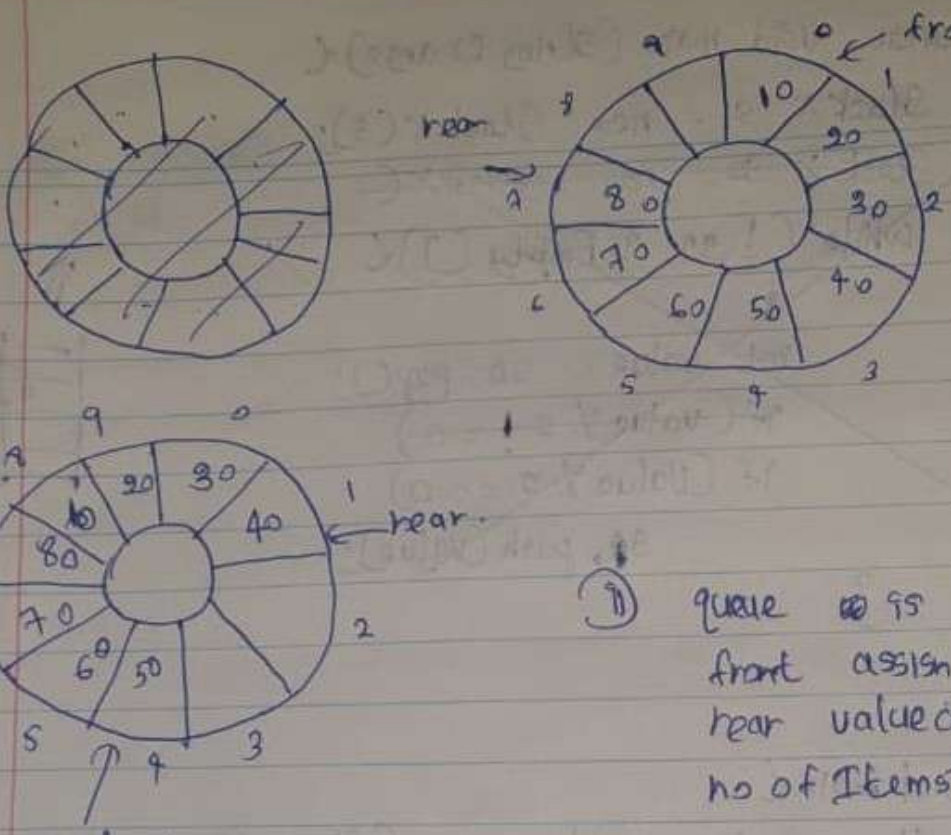


(iv)

(v)

```
public StackMain{
    public static void main (String [] args){
        StackX s1 = new StackX(5);
        StackX s2 = new StackX(5);
        while (!s1.isEmpty()){
            int value = s1.pop();
            if (value%2 == 0);
                s2.push(value);
        }
    }
    while (!s2.isEmpty()){
        s1.push(s2.pop());
    }
}
```

(b) (i)



(ii) queue is empty and front assign to 0 and rear value assign to -1 no of Items also 0

(iii)

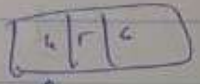
```
public int getSize(){ int nItems=0;
    if (isEmpty()){
        System.out.println(" No of Items 0");
    }
    else {
        while (!isEmpty()){
            int nItems=
        }
```

(iii)

```
public int getSize(){
    return nItems;
}
```

(c).

```
public Main{
    public static void main(String()args){
        Stack X myStack = new Stack X(5);
        Queue X myQueue = new myQueue X(5);
```



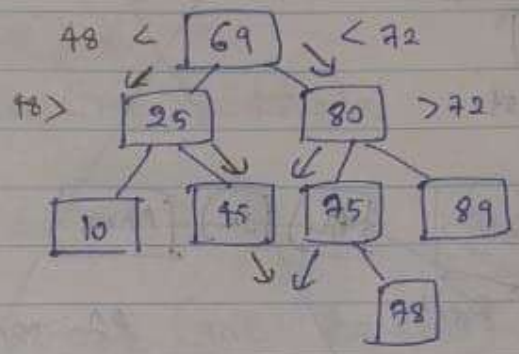
4, 5, 6 → ran

← stack

(c)
```
while(!myQueue.isEmpty()){
    myStack.push(myQueue.remove());
}
while(!myStack.isEmpty()){
    myQueue.insert(myStack.pop());
}
}
```

Question 2

(a) (i) (ii) 75



(iii) 45

(iv) non leaf → parent
25, 80, 75, 69

(v) 45

(vi) 3

(vii) No

(b) (i) Student kmark = studentMarks.maximum();
kmark.displayDetails();

(ii) Student Lmark = studentMarks.minimum();
Lmark.displayDetails();

(iii) all mark = studentMarks.descendingOrder();
System.out.println(all mark);

(d.) (i) public [redacted] isAvailable (int bookNo) {

[redacted] book cur = first;

while (! cur [redacted]) {

if (bookNo == cur.BookNo) {

return [redacted] cur;

}

else {

[redacted] cur = cur.next;

}

}

return false;

}

(ii) ~~public void lending() {~~

public void lending (int bookNo) {

book cur = first;

(d.) public book isAvailable (int bookNo) {

book cur = first;

while (cur != null) {

if (cur.BookNo == bookNo) {

return current;

}

else {

cur = cur.next;

}

}

return false;

}

(ii) Last Page.

(11) public void lending (int bookNo) {
Book book = isAvailable (bookNo)

if (book != null) {
book.numOfCopies--;
System.out.println ("Borrowed");
}
else {
System.out.println ("bookNo is not available");
}
}

Question 3

(a) O(1) In this case, 'addss' method is a loop it is running (Iterates) 100 times. The loop always executes fixed number of iterations. It takes a constant time. ∴ O(1) = T(n)

(b) (I) QUICKSORT (Arr, low, high)

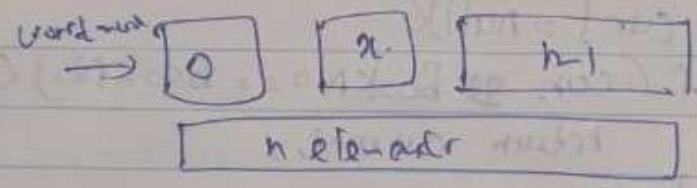
1. If low < high.
2. p = PARTITION(Arr, low, high).
3. QUICKSORT (Arr, low, p-1)
4. QUICKSORT (Arr, p+1, high)

Lg 3 and 4 lines are incorrect, a correct algorithm

(II)

QUICKSORT (Arr, low, high) ⟶ T(n).

1. If low < high ⟶ C_1
2. p = PARTITION (Arr, low, high) ⟶ $C_2 \cdot n$
3. QUICKSORT (Arr, low, p-1) ⟶ T(0).
4. QUICKSORT (Arr, p+1, high) ⟶ T(n-1)



$T(n) = T(0) + T(n-1) + C_2 \cdot n + C_1$

$T(n) = C_2 n + T(n-1)$

(III) $T(n) = T(n-1) + C_2 \cdot n$

$T(n-1) = T(n-2) + C_2 (n-1)$

$T(n-2) = T(n-3) + C_2 (n-2)$

$T(2) = T(1) + C_2 \cdot 2$

$T(1) = T(0) + C_2 \cdot 1$

$T(0) = 0$

$T(n) = c_2.1 + c_2.2 + \cdots + c_2(n-2) + c_2(n-1) + c_2.n$

$T(n) = c_2(1+2+\cdots+(n-2)+(n-1)+n)$

$S_n = \frac{n}{2}(a+l)$

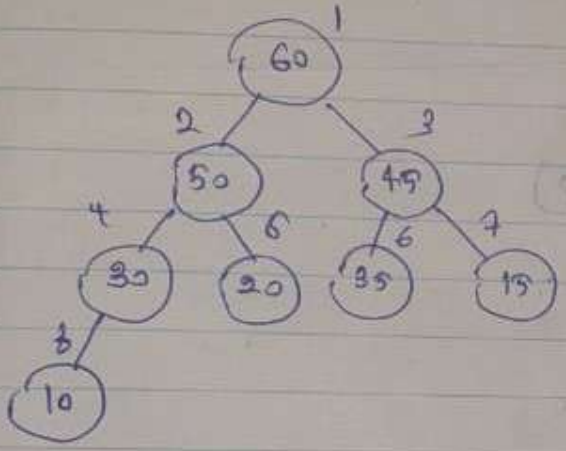
$S_n = \frac{n}{2}(1+n)$

$T(n) = c_2(n/2 + n^2/2)$

$T(n) = O(n^2)$

(c.i) Parent node should be greater than or equal to their child nodes.

(ii)



Yes. All the parent nodes are greater than their child nodes values.

(d) MIN_HEAPIFY(A,i)

```
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap_size and A[l] < A[i]
    smallest = l;
else
    smallest = i;
if r ≤ A.heap_size and A[r] < A[smallest]
    smallest = r;
if smallest ≠ i
    exchange A[i] with A[smallest]
    MIN_HEAPIFY(A, smallest)
```

```
HEAP_EXTRACT_MIN(A[1..n])
if A.heap_size >= 1
    min = A[1]
    A[1] = A[A.heap_size]
    A.heap_size = A.heap_size - 1
    MIN_HEAPIFY(A, 1)
    return min
```

(ii)
```
public void lending (int bookNo) {
    Book book = isAvailable (bookNo)

    if (book != null) {
        book.numofCopies--;
        System.out.println("Borrowed");
    }
    else {
        System.out.println("a bookNo is not available");
    }
}
```

Question 4

(a). Native String - Matcher (T, P)

1. n = T.length → 9
2. m = P.length → 3.
3. for s=0 to n-m.
   s=0 to 9-3=6
4. if P[1..n'] = T[s+1 .. s+m]
   - abb = aaa x → 2
   - bba = aaa → 1
   - bac = aaa → 1
   - acd = aaa → 2
   - cda = aaa → 1
   - daa = aaa → 1
   - aab = aaa → 3
5. then print "Pattern occurs with shift" s (2) 7

(i) 2+2+3+1+1+1+1 = 11

(ii). 3 valids shifts and 4 invalid shifts

(b). (i) 0 valid shifts and 6 invalid shifts

7.

(b) (i) $q = 100$.

$P = 100$

(ii). $P \% q = 100 \% 100 = 0$.

Text $T = 2034100520006$.

$203 \% 100 = 3$
$034 \% 100 = 34$
$341 \% 100 = 41$
$410 \% 100 = 10$
$100 \% 100 = 0$ → valid
$005 \% 100 = 5$
$052 \% 100 = 52$
$200 \% 100 = 0$ → spurious
$006 \% 100 = 6$

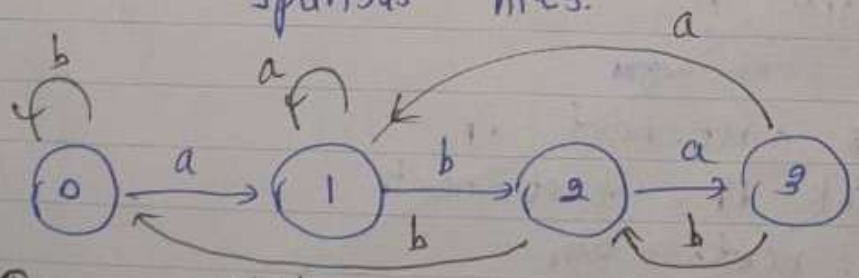
valid → 1
spurious hit → 1

(iii) Increasing the modulo value

(iv) Worst-case scenario occurs when the Rabin-Karp algorithm has,

All the hits valid
All the hits spurious.
All hits are combination of valid & spurious hits.

(c)



$T = a$
$T = b$ ×

$T = aba$ ✓
$T = ab$ ✓

$T = aba$ ✓
$T = a$ b b ×

$T = aba$ a
$T = ab$ a b

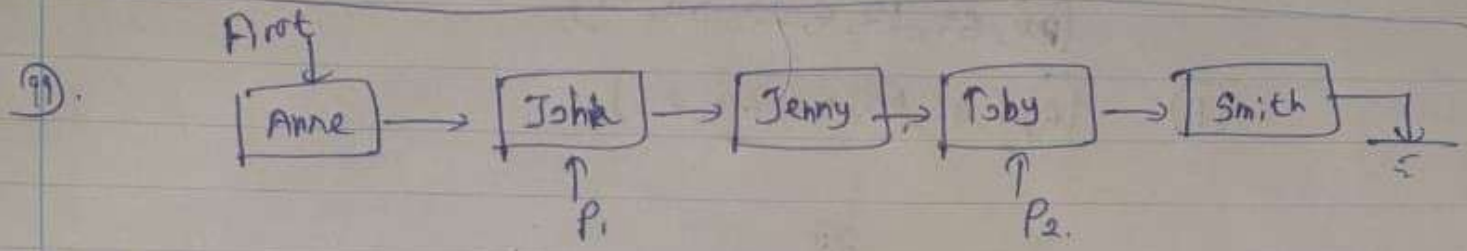
diagram A

Link first = Anne.
Link P1 = John
Link P2 = Tobby.

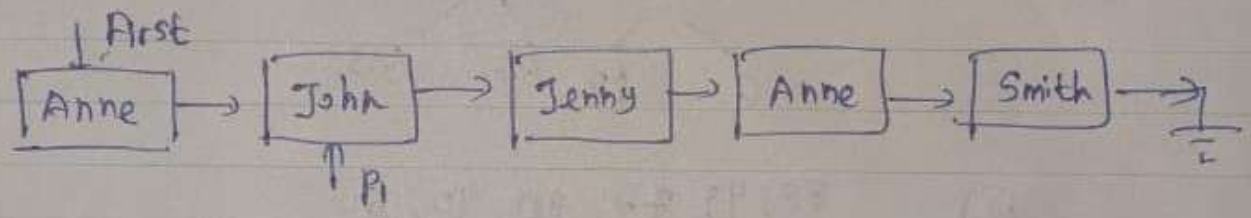
first. next = P1
P1. next = first. next. next.

diagram B

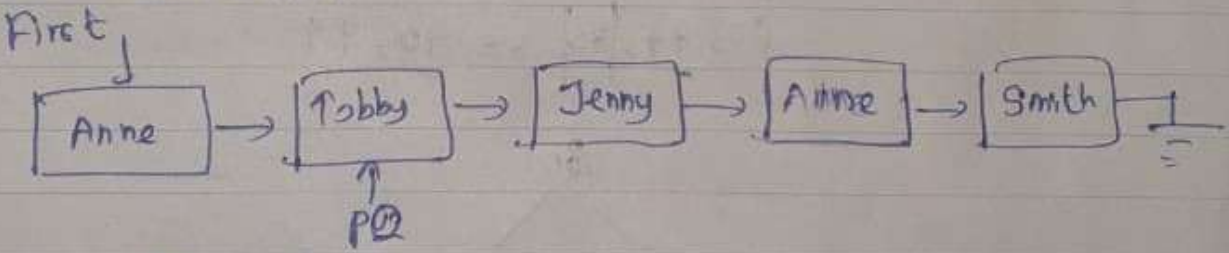
first = Anne.
first. next = Jenny;
P1. next = P2
P2. next = Smith.



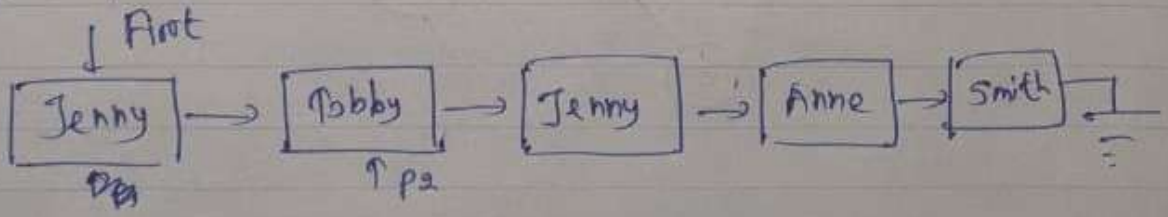
(1) P1. next. next = First.



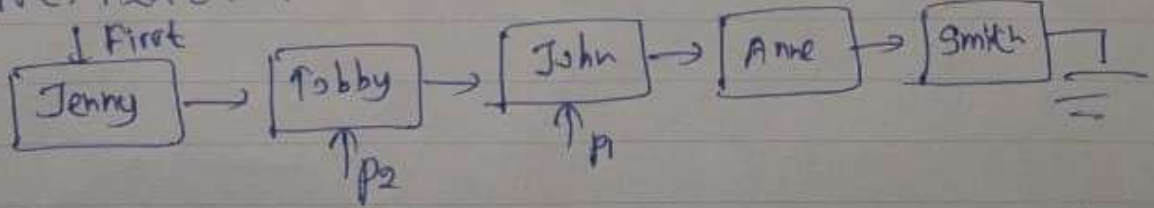
(2) First. next = P2.



(3) First = P2. next.



(4) First. next = P1



p2. next = NULL.