2024 - June

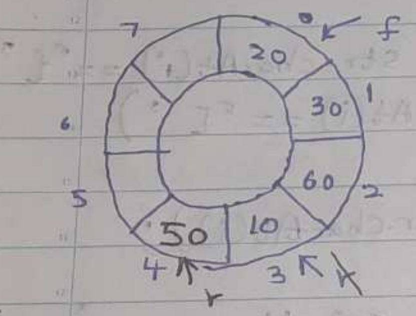
Q1

(a)

Data structure	Operation principle	Insertion	Deletion	Real-world Application.
Stack	LIFO	push() top	pop() top	Browser history
Queue	FIFO	insert() rear	remove() front	printer Queue

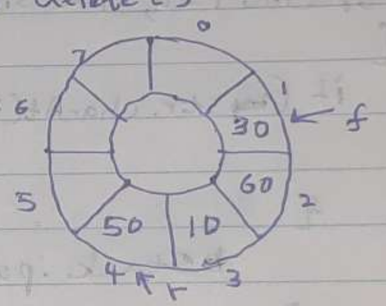
(b) Although the queue is not full we cannot insert more elements when the rear is at the end. (memory wastage)

(c) (i) insert (50)



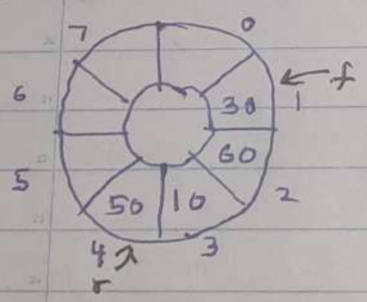
f → 0, r → 4, count → 5

(ii) delete ()



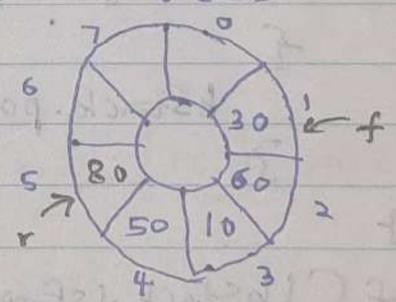
f → 1, r → 4, count → 4

(iii) peekfront ()



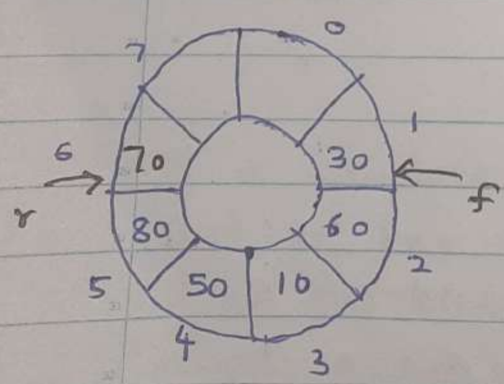
f → 1, r → 4, count → 4

(iv) insert (80)



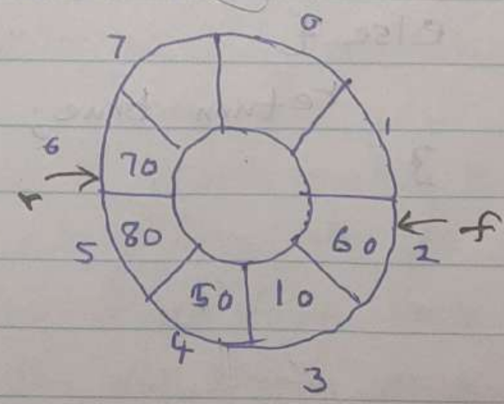
f → 1, r → 5, count → 5

(v) insert (70)



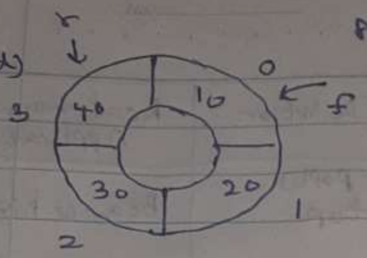
f → 1, r → 6, count → 6

(vi) delete ()



f → 2, r → 6, count → 5

(d)



Ayree

Rear - Front + 1 = 4
3 - 0 + 1 = 4
4 = 4

(e)
```
public boolean isBalanced( String str ) {
        int size = str.length();

        StackX bStack = new stackX (size);

        for (int i=0; i< size; i++ )
        {
            if ( str.charAt(i)=='(' || str.charAt(i) == '{'
                            || str.charAt(i) == '[' )
            {
                bStack.push( str.charAt(i));
            }
            else if ( str.charAt(i)==')' || str.charAt(i)
                == '}' || str.charAt (i) == ']' )
            {
                bStack.pop ();
            }
        }
        if (!bStack.isEmpty()) {
            return false;
        }
        else {
            return true;
        }
}
```

(f) public class BracketCheck {

```
    public static void main (String[] args)
    {
        ~~StackX st = new StackX (~~
        Scanner sc = new Scanner (System.in);
        System.out.print ("Enter string : ");
        String str = sc.nextLine ();
        int len = str.length();
        StackX st = new StackX (len);

        ~~st.~~

        Boolean res = st.isBalanced (str);
        if (res) {
            System.out.println ("Parantheses are balanced");
        }
        else {
            System.out.println ("Parantheses are imbalanced");
        }
    }
}
```

2

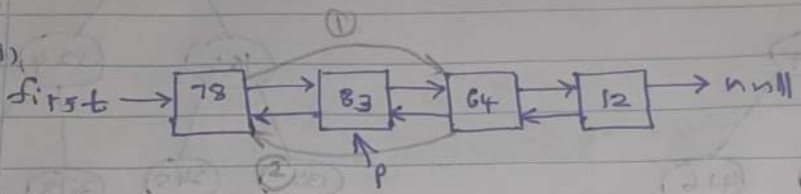
(a)(i) 72, 60, 100, 10

(ii) 100

(b)

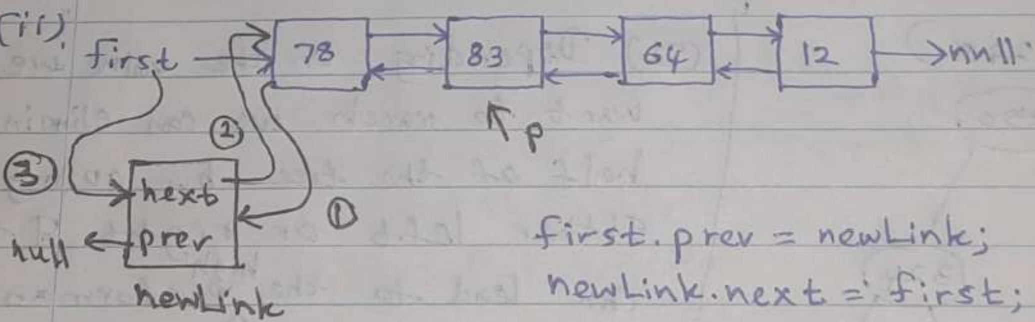
```
Link temp = first;
while (temp != NULL)
{
    if (temp.next == NULL)
        System.out.print (temp.ID)
    else
        temp = temp.next;
}
```

(c) (i)



first.next = p.next;
p.next.previous = first;

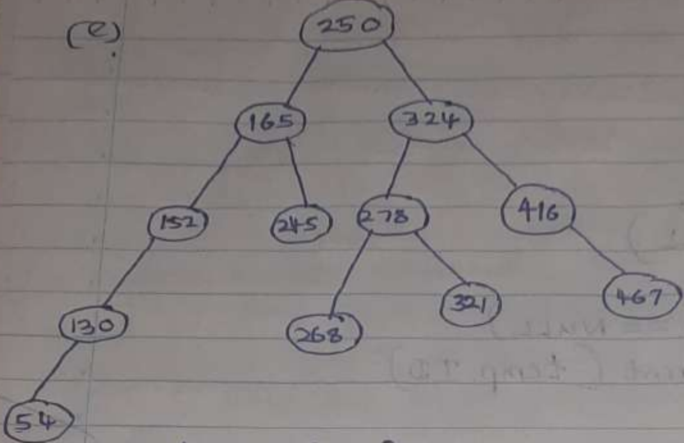
(ii)



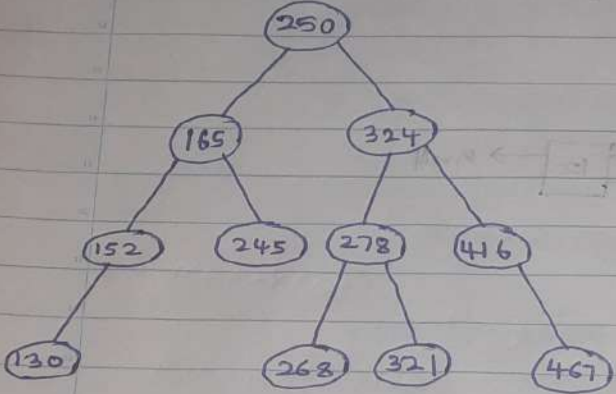
first.prev = newLink;
newLink.next = first;
~~newLin~~ first = newLink;

(d) In a complete binary tree, each node is either a leaf or has degree ≤ 2. Filling of the nodes must ~~In a full binary tree~~ be filled from left to right, which is not mandatory in full binary tree.

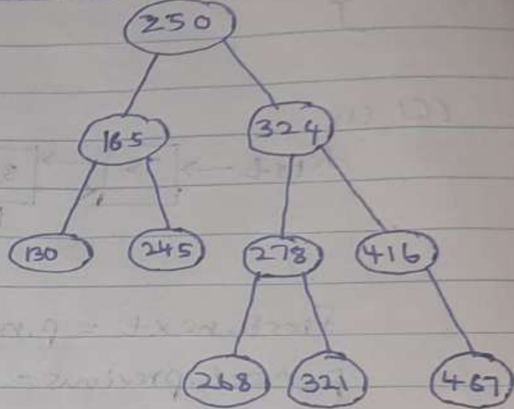
(e)



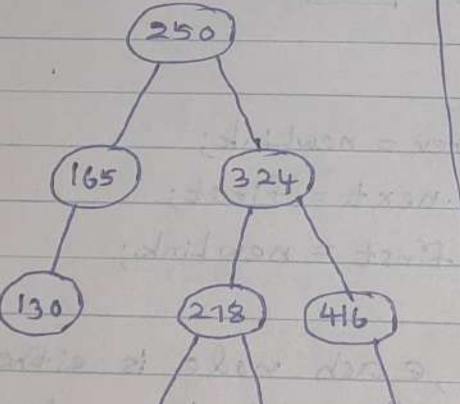
(f) (i) delete (54)



(ii) delete (152)



(iii) delete (245)



(g) Depending on the value we want to search we can eliminate half of the tree by going either left or right. It can lead to the high performance.

Q3 $(10, 8, 6, 4), 2, 0, -2$

(a) (i)

$i = 10$ — 1

while $i >= 0$ — 7 (6)

if $i > 5$ — $4+5$ (3)

print i — 1×3

$i = i - 2$ — 2×6

$T(n) = 1 + 7 + 9 + 3 + 12$

$T(n) = \underline{32}$

(ii)

for $i = 0$ to n — $n+1$ ($n+2$, $n+2$, $n+1$)

(n+1) for $j = 1$ to 5 — 6, 6

(5) $a = i + j$ — 2×5

print a — 1×5

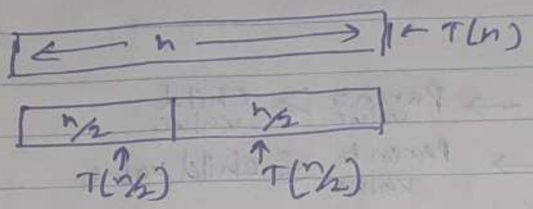
$T(n) = 3n + 5 + 17(n+1) + 10(n+1) + 5(n+1)$

$T(n) = 3n + 5 + 17n + 17 + 10n + 10 + 5n + 5$

$T(n) = 35n + 37$

(b) First we divide array into 2 equal parts. This is balanced partition. So it has best case only.

(c)



MergeSort(A, p, r) $\longrightarrow T(n)$

1. if $p < r$ $\rightarrow c_1$
2. $q = \lfloor (p+r)/2 \rfloor \rightarrow c_2$
3. MergeSort(A, p, q) $\rightarrow T(n/2)$
4. MergeSort(A, q+1, r) $\rightarrow T(n/2)$
5. Merge(A, p, q, r) $\longrightarrow c_3 n$

Merging time $\propto n$ $= cn$

$T(n) = \cancel{c_1} + \cancel{c_2} + 2T\left(\frac{n}{2}\right) + c_3 n$

$T(n) = 2T\left(\frac{n}{2}\right) + cn$

(d) $a = 2$, $b = 2$, $f(n) = cn$

$f(n)$ vs $n^{\log_b a}$

cn vs $n^{\log_2 2}$

n vs n $\rightarrow$ case 2

$T(n) = O(n^{\log_b a} \log_2 n)$

$T(n) = O(n \log_2 n)$

	1	2	3	4
A	4	6	7	9

(a)

(i) Partition (A, 1, 4)

(1) $x = A[4] = 9$

(2) $i = 0$

(3) for $j = 1$ to 3

(4) if $A[1] \leq 9$

(5) $i = 1$

(6) $A[1] \leftrightarrow A[1]$

(3) for $j = 2$ to 3

(4) if $A[2] \leq 9$

(5) $i = 2$

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

(3) for $j = 3$ to 3

(4) if $A[3] \leq 9$

(5) $i = 3$

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

(7) $A[4] \leftrightarrow A[4]$

(8) return 4

(ii)

Q4 (a) max-heap property → Parent value ≥ child value

min-heap property → Parent value ≤ child value

(b) (i) Traverse through all the parent nodes

(ii) Apply Heapify() for all the violating nodes one by one

(c)

T: | 2 | 0 | 3 | 0 | 4 | 0 | 3 | 0 | 5 | 0 |

Shift	Pattern	Result	# of Comparisons
s=0	3 0	I.S.	1
s=1	3 0	I.S.	1
s=2	3 0	V.S.	2
s=3	3 0	I.S.	1
s=4	3 0	I.S.	1
s=5	3 0	I.S.	1
s=6	3 0	V.S.	2
s=7	3 0	I.S.	1
s=8	3 0	I.S.	1
			11

Invalid shifts – 7

Valid shifts – 2

(d) $q = 100$, $P = 30$

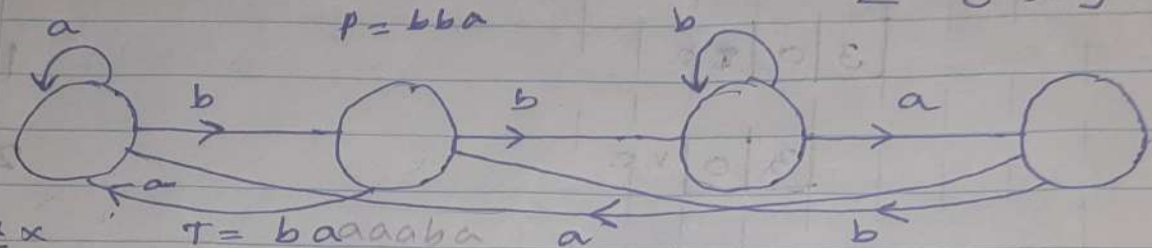
$p \% q = 30 \% 100 = 30$

$20 \% 100 = 20$
$03 \% 100 = 3$
$30 \% 100 = 30 \leftarrow$ v.h.
$04 \% 100 = 4$
$40 \% 100 = 40$
$03 \% 100 = 3$
$30 \% 100 = 30 \leftarrow$ v.h
$05 \% 100 = 5$
$50 \% 100 = 50$

valid hits $\rightarrow 2$ //

$\Sigma = \{a, b\}$

(e) $P = bba$



$T = \underline{a}$ × $T = b\underline{a}aaaba$

$T = bb\underline{a}$ ✓ $T = bba\underline{a}$

$T = \underline{b}$ ✓ $T = b\underline{b}$ ✓

$T = bb\underline{b}bba$ $T = bba\underline{b}$

Q2

(a)(i) 72, 60, 100, 10

(ii) 100

(b)
```
Link temp = first;
while (temp != NULL)
{
    if (temp.next == NULL)
        System.out.print (temp.ID)
        Break;
    else
        temp = temp.next;
}
```

(C) (i)