

1. Priority Queue:-

```
2. #include <stdio.h>
3. #define MAX 5
4.
5. int queue[MAX];
6. int priority[MAX];
7. int front = -1, rear = -1;
8.
9. int isFull() {
10.     return rear == MAX - 1;
11. }
12.
13. int isEmpty() {
14.     return front == -1 || front > rear;
15. }
16.
17. void enqueue(int value, int p) {
18.     if (isFull()) {
19.         printf("Queue Overflow! Cannot enqueue %d\n", value);
20.         return;
21.     }
22.
23.     if (front == -1) {
24.         front = 0;
25.         rear = 0;
26.         queue[rear] = value;
27.         priority[rear] = p;
28.         printf("Enqueued %d with priority %d into the
queue.\n", value, p);
29.         return;
30.     }
31.
32.     int i;
33.     for (i = rear; i >= front && priority[i] > p; i--) {
34.         queue[i + 1] = queue[i];
35.         priority[i + 1] = priority[i];
36.     }
37.
38.     queue[i + 1] = value;
39.     priority[i + 1] = p;
40.     rear++;
41.
```

```
42.         printf("Enqueued %d with priority %d into the queue.\n",
   value, p);
43.     }
44.
45.     void dequeue() {
46.         if (isEmpty()) {
47.             printf("Queue Underflow! Queue is empty.\n");
48.             return;
49.         }
50.
51.         int dequeuedValue = queue[front];
52.         front++;
53.
54.         if (front > rear) {
55.             front = -1;
56.             rear = -1;
57.         }
58.
59.         printf("Dequeued %d from the queue.\n", dequeuedValue);
60.     }
61.
62.     void display() {
63.         if (isEmpty()) {
64.             printf("Queue is empty.\n");
65.             return;
66.         }
67.
68.         printf("Queue elements: ");
69.         for (int i = front; i <= rear; i++)
70.             printf("%d ", queue[i]);
71.
72.         printf("\nPriorities: ");
73.         for (int i = front; i <= rear; i++)
74.             printf("%d ", priority[i]);
75.
76.         printf("\n");
77.     }
78.
79.     int main() {
80.         int choice, value, p;
81.
82.         while (1) {
```

```
83.         printf("\nPriority Queue Operations:\n");
84.         printf("1. Enqueue\n");
85.         printf("2. Dequeue\n");
86.         printf("3. Display\n");
87.         printf("4. Exit\n");
88.         printf("Enter your choice (1-4): ");
89.         scanf("%d", &choice);

90.
91.         switch (choice) {
92.             case 1:
93.                 printf("Enter value and priority: ");
94.                 scanf("%d %d", &value, &p);
95.                 enqueue(value, p);
96.                 break;
97.             case 2:
98.                 dequeue();
99.                 break;
100.            case 3:
101.                display();
102.                break;
103.            case 4:
104.                printf("Exiting program.\n");
105.                return 0;
106.            default:
107.                printf("Invalid choice! Please enter 1-4.\n");
108.        }
109.    }
110.
111.    return 0;
112. }
```

OUTPUT:-

```
4. Exit
Enter your choice (1-4): 1
Enter value and priority: 10 1
Enqueued 10 with priority 1 into the queue.

Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 1
Enter value and priority: 30 3
Enqueued 30 with priority 3 into the queue.

Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 3
Queue elements: 10 30
Priorities: 1 3

Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 1
Enter value and priority: 20 2
Enqueued 20 with priority 2 into the queue.

Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 3
Queue elements: 10 20 30
Priorities: 1 2 3

Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 2
Dequeued 10 from the queue.

Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 1
Enter value and priority: 20 2
Enqueued 20 with priority 2 into the queue.

Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 3
Queue elements: 10 20 30
Priorities: 1 2 3

Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 2
Dequeued 10 from the queue.

Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 3
Queue elements: 20 30
Priorities: 2 3

Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 1
```

2.Input restricted Queue:-

```
#include <stdio.h>
#define MAX 5

int deque[MAX];
int front = -1, rear = -1;

int isFull() {
    return rear == MAX - 1;
}

int isEmpty() {
    return front == -1 || front > rear;
}

void insertRear(int value) {
    if (isFull()) {
        printf("Deque Overflow! Cannot insert %d\n", value);
        return;
    }
    if (front == -1) {
        front = 0;
        rear = 0;
    } else {
        rear++;
    }
    deque[rear] = value;
    printf("Inserted %d at rear.\n", value);
}

void deleteFront() {
    if (isEmpty()) {
        printf("Deque Underflow! Deque is empty.\n");
        return;
    }
    int deleted = deque[front];
    front++;
    if (front > rear) {
        front = -1;
        rear = -1;
    }
    printf("Deleted %d from front.\n", deleted);
}

void deleteRear() {
    if (isEmpty()) {
        printf("Deque Underflow! Deque is empty.\n");
        return;
    }
```

```

    }
    int deleted = deque[rear];
    rear--;
    if (front > rear) {
        front = -1;
        rear = -1;
    }
    printf("Deleted %d from rear.\n", deleted);
}

void display() {
    if (isEmpty()) {
        printf("Deque is empty.\n");
        return;
    }
    printf("Deque elements: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", deque[i]);
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\nInput-Restricted Deque Operations:\n");
        printf("1. Insert at rear\n");
        printf("2. Delete from front\n");
        printf("3. Delete from rear\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice (1-5): ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert at rear: ");
                scanf("%d", &value);
                insertRear(value);
                break;
            case 2:
                deleteFront();
                break;
            case 3:
                deleteRear();
                break;
            case 4:
                display();
                break;
            case 5:
                printf("Exiting program.\n");
                return 0;
        }
    }
}

```

```

default:
    printf("Invalid choice! Enter 1-5.\n");
}
}

return 0;
}

```

OUTPUT:-

```

Input-Restricted Deque Operations:
1. Insert at rear
2. Delete from front
3. Delete from rear
4. Display
5. Exit
Enter your choice (1-5): 1
Enter value to insert at rear: 10
Inserted 10 at rear.

Input-Restricted Deque Operations:
1. Insert at rear
2. Delete from front
3. Delete from rear
4. Display
5. Exit
Enter your choice (1-5): 1
Enter value to insert at rear: 20
Inserted 20 at rear.

Input-Restricted Deque Operations:
1. Insert at rear
2. Delete from front
3. Delete from rear
4. Display
5. Exit
Enter your choice (1-5): 1
Enter value to insert at rear: 30
Inserted 30 at rear.

Input-Restricted Deque Operations:
1. Insert at rear
2. Delete from front
3. Delete from rear
4. Display
5. Exit
Enter your choice (1-5): 2
Deleted 10 from front.

Input-Restricted Deque Operations:
1. Insert at rear
2. Delete from front
3. Delete from rear
4. Display
5. Exit
Enter your choice (1-5): 3
Deleted 30 from rear.

Input-Restricted Deque Operations:
1. Insert at rear
2. Delete from front
3. Delete from rear
4. Display
5. Exit
Enter your choice (1-5): 4
Deque elements: 20

```

> OUTLINE

> TIMELINE

C PROGRAM

- a.exe
- hello.c
- hello.exe
- inputRQ.c**
- inputRQ.exe

Q

o

g

E³

⑧

④

W

Ln 6 Col 1 Spaces:4 UTF-8 CRLF {} C Win32

```

*Output Restricted Queue:-
#include <stdio.h>
#define MAX 5

int deque[MAX];
int front = -1, rear = -1;

int isFull() {
    return rear == MAX - 1;
}

int isEmpty() {
    return front == -1 || front > rear;
}

void insertFront(int value) {
    if (isFull()) {
        printf("Deque Overflow! Cannot insert %d\n", value);
        return;
    }
    if (front == -1) {
        front = 0;
        rear = 0;
    } else {

        for (int i = rear; i >= front; i--) {
            deque[i + 1] = deque[i];
        }
        rear++;
    }
    deque[front] = value;
    printf("Inserted %d at front.\n", value);
}

void insertRear(int value) {
    if (isFull()) {
        printf("Deque Overflow! Cannot insert %d\n", value);
        return;
    }
    if (front == -1) {
        front = 0;
        rear = 0;
    } else {
        rear++;
    }
    deque[rear] = value;
    printf("Inserted %d at rear.\n", value);
}

void deleteFront() {
    if (isEmpty()) {
        printf("Deque Underflow! Deque is empty.\n");
        return;
    }
}

```

```

    }
    int deleted = deque[front];
    front++;
    if (front > rear) {
        front = -1;
        rear = -1;
    }
    printf("Deleted %d from front.\n", deleted);
}

void display() {
    if (isEmpty()) {
        printf("Deque is empty.\n");
        return;
    }
    printf("Deque elements: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", deque[i]);
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\nOutput-Restricted Deque Operations:\n");
        printf("1. Insert at front\n");
        printf("2. Insert at rear\n");
        printf("3. Delete from front\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice (1-5): ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert at front: ");
                scanf("%d", &value);
                insertFront(value);
                break;
            case 2:
                printf("Enter value to insert at rear: ");
                scanf("%d", &value);
                insertRear(value);
                break;
            case 3:
                deleteFront();
                break;
            case 4:
                display();
                break;
            case 5:

```

```

        printf("Exiting program.\n");
        return 0;
    default:
        printf("Invalid choice! Enter 1-5.\n");
    }
}
return 0;
}

```

OUTPUT:-

```

C PROGRAM
└─ a.exe
└─ hello.c
└─ hello.exe
└─ inputRQ.c
└─ inputRQ.exe
└─ outputRQ.c
└─ outputRQ.exe

Output-Restricted Deque Operations:
1. Insert at front
2. Insert at rear
3. Delete from front
4. Display
5. Exit
Enter your choice (1-5): 1
Enter value to insert at front: 10
Inserted 10 at front.

Output-Restricted Deque Operations:
1. Insert at front
2. Insert at rear
3. Delete from front
4. Display
5. Exit
Enter your choice (1-5): 1
Enter value to insert at front: 20
Inserted 20 at front.

Output-Restricted Deque Operations:
1. Insert at front
2. Insert at rear
3. Delete from front
4. Display
5. Exit
Enter your choice (1-5): 4
Deque elements: 20 10

Output-Restricted Deque Operations:
1. Insert at front
2. Insert at rear
3. Delete from front
4. Display
5. Exit
Enter your choice (1-5): 2
Enter value to insert at rear: 30
Inserted 30 at rear.

Output-Restricted Deque Operations:
1. Insert at front
2. Insert at rear
3. Delete from front
4. Display
5. Exit
Enter your choice (1-5): 4
Deque elements: 20 10 30

Output-Restricted Deque Operations:
1. Insert at front
2. Insert at rear
3. Delete from front
4. Display
5. Exit
Enter your choice (1-5): 5
Exiting program.

PS C:\Users\mahme\Desktop\c program>

```

The terminal window shows three separate sessions of a C program demonstrating Restricted Deque Operations. The sessions show insertions at front and rear, deletions from front, and displaying the deque elements.

LEETCODE:-

```

#include <stdio.h>
#include <stdlib.h>

```

```

typedef struct {
    int k;
}

```

```

int size;
int arr[1000];
} KthLargest;

void sortDescending(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] < arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}

KthLargest* kthLargestCreate(int k, int* nums, int numsSize) {
    KthLargest* obj = malloc(sizeof(KthLargest));
    obj->k = k;
    obj->size = numsSize;
    for (int i = 0; i < numsSize; i++) {
        obj->arr[i] = nums[i];
    }
    sortDescending(obj->arr, obj->size);
    return obj;
}

int kthLargestAdd(KthLargest* obj, int val) {
    obj->arr[obj->size++] = val;
    sortDescending(obj->arr, obj->size);
    if (obj->size < obj->k)
        return obj->arr[obj->size - 1];
    return obj->arr[obj->k - 1];
}

void kthLargestFree(KthLargest* obj) {
    free(obj);
}

```

OUTPUT:-

The screenshot shows a LeetCode test result page. At the top, it says "Accepted" with a runtime of 0 ms. There are two test cases: Case 1 and Case 2, both of which passed. The input for Case 1 is ["KthLargest","add","add","add","add","add"] and the output is [null,4,5,5,8,8]. The expected output is also [null,4,5,5,8,8]. Below the input and output sections, there is a link to "Contribute a testcase".

Leedcode:-

```
#include <stdio.h>
void sortDescending(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] < arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}

int lastStoneWeight(int stones[], int stonesSize) {
    while (stonesSize > 1) {
        sortDescending(stones, stonesSize);
        int y = stones[0];
        int x = stones[1];

        if (x == y) {

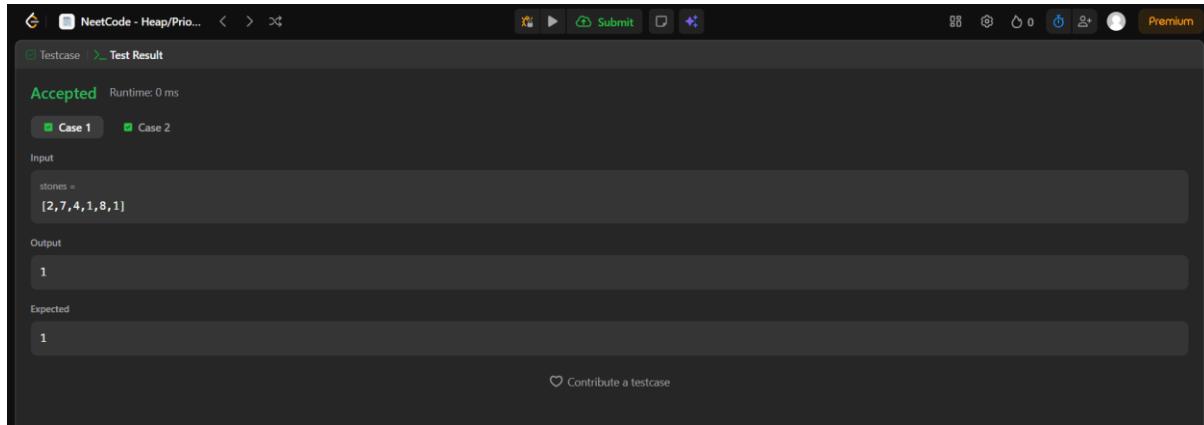
            for (int i = 2; i < stonesSize; i++) {
                stones[i - 2] = stones[i];
            }
            stonesSize -= 2;
        } else {

            stones[0] = y - x;
            for (int i = 2; i < stonesSize; i++) {
                stones[i - 1] = stones[i];
            }
            stonesSize -= 1;
        }
    }
}
```

```
}

return stonesSize == 0 ? 0 : stones[0];
}
```

OUTPUT:-



The screenshot shows a test result from NeetCode. It indicates the code has been accepted with a runtime of 0 ms. Two test cases are shown: Case 1 and Case 2, both of which have passed. The input for Case 1 is an array [2,7,4,1,8,1]. The output is 1, which matches the expected value. The interface includes tabs for Testcase and Test Result, and various submission and sharing buttons.

Leetcode:-

```
#include <stdio.h>

int leastInterval(char* tasks, int tasksSize, int n) {
    int freq[26] = {0};
    for (int i = 0; i < tasksSize; i++) {
        freq[tasks[i] - 'A']++;
    }
    int maxFreq = 0;
    for (int i = 0; i < 26; i++) {
        if (freq[i] > maxFreq)
            maxFreq = freq[i];
    }

    int maxCount = 0;
    for (int i = 0; i < 26; i++) {
        if (freq[i] == maxFreq)
            maxCount++;
    }

    int result = (maxFreq - 1) * (n + 1) + maxCount;

    if (result < tasksSize)
        result = tasksSize;

    return result;
}
```

OUTPUT:-

The screenshot shows a test results interface from NeetCode. At the top, it says "Accepted" with a runtime of 0 ms. Below that, there are three tabs: "Case 1" (selected), "Case 2", and "Case 3". The "Input" section shows "tasks = ["A", "A", "A", "B", "B", "B"]" and "n = 2". The "Output" section shows "8". The "Expected" section also shows "8". At the bottom right, there is a link to "Contribute a testcase".

```
Accepted Runtime: 0 ms
Case 1 Case 2 Case 3

Input
tasks =
["A", "A", "A", "B", "B", "B"]

n =
2

Output
8

Expected
8

Contribute a testcase
```