STACK & QUEUE - SET 3 SOLUTIONS

Note: All solutions are written in C++.

Question 1:

- **Problem link**: https://leetcode.com/problems/min-stack/
- **Difficulty level**: Easy

Solution:

```
class MinStack {
public:
    // our stack doesn't contain actual values
    stack<long long int> s;
    long long int minEle;
    MinStack() {
        minEle = 0;
    }

    void push(long long int val) {
        if(s.empty())
        {
            s.push(val);
            minEle = val;
        }
        else if(val >= minEle)
        {
            s.push(val);
        }
        else if(val < minEle)
        {
            s.push(2*val - minEle);
```

```
            minEle = val;
        }
    }

    void pop() {
        if(s.top() < minEle)
        {
            minEle = 2*minEle - s.top();
            s.pop();
        }
        else
        {
            s.pop();
        }
    }

    int top() {
        return (s.top() < minEle) ? minEle : s.top();
    }

    int getMin() {
        return minEle;
    }
};
```

Complexity:

- Time: O(1)
- Space: O(n)

Question 2:

- **Problem link**: https://leetcode.com/problems/next-greater-element-i/
- **Difficulty level**: Easy

Solution:

```
class Solution {
public:
    vector<int> nextGreaterElement(vector<int>& nums1, vector<int>&
nums2) {
        // (num, neextGreaterElement)
        unordered_map<int, int>mp;
        stack<int> s;
        s.push(nums2[0]);
        vector<int> res;
        for(int i = 1; i < nums2.size(); i++)
        {
            if(s.empty())
            {
                s.push(nums2[i]);
                continue;
            }
            while(!s.empty() && s.top() < nums2[i])
            {
                mp.insert({s.top(), nums2[i]});
                s.pop();
            }
            s.push(nums2[i]);
        }
        for(int i = 0; i < nums1.size(); i++)
        {
            if(mp.find(nums1[i]) != mp.end())
            {
                res.push_back(mp[nums1[i]]);
            }
            else res.push_back(-1);
        }
        return res;
```

```
    }
};
```

Complexity:

- Time: O(n)
- Space: O(n)

Question 3:

- **Problem link**: https://practice.geeksforgeeks.org/problems/3-data-structures/0/?category[]=priority-queue&category[]=priority-queue&page=1&query=category[]priority-queuepage1category[]priority-queue
- **Difficulty level**: Easy

Explanation: the solution is quite straightforward where we just declare a priority queue and a queue, and based on the input operation, we push a number wither to the queue or the priority queue. And then we pop elements from the queue and the priority queue until they are empty.

Solution:

```
int main()
{
      int test;
      cin>>test;

      while(test--)
      {
          int n;
          cin>>n;
```

```
        priority_queue<int> pq;
        queue<int> q;
        stack<int> s;

        while(n--)
        {
            int op;
            cin>>op;
            int num;
            cin>>num;
            if(op == 1)
                pq.push(num);
            else if(op == 2)
                q.push(num);
            else
                s.push(num);
        }
        while(!pq.empty())
        {
            cout<<pq.top()<<" ";
            pq.pop();
        }
        while(!q.empty())
        {
            cout<<q.front()<<" ";
            q.pop();
        }
        while(!s.empty())
        {
            cout<<s.top()<<" ";
            s.pop();
        }
        cout<<endl;
    }
    return 0;
```

```
}
```

Complexity:

- Time: O(N)
- Space: O(N)

Question 4:

- **Problem link**: https://practice.geeksforgeeks.org/problems/easy-string2212/1/?category[]=Stack&category[]=Stack&difficulty[]=0&page=1&query=category[]Stackdifficulty[]0page1category[]Stack
- **Difficulty level**: Easy

Explanation: to solve the above problem we start a loop till the end of the string and for every iteration, increment a count till the character at ith position matches the following character.

Solution:

```
 #include <iostream>
using namespace std;

void printRLE(string s)
{
    for (int i = 0; s[i] != '&#092;&#048;'; i++) {

        int count = 1;
        while (s[i] == s[i + 1]) {
            i++;
            count++;
        }
```

```
            cout << s[i] << count << " ";
        }

        cout << endl;
}
```

Complexity:

- Time: O(|S|)
- Space: O(1)

Question 5:

- **Problem link**: https://leetcode.com/problems/design-circular-deque/
- **Difficulty level**: Medium

Explanation:

Use a vector to implement the circular deque. Initialise variables to track the front element, the rear element and the number of elements present in the deque. Use the 'count' variable to decide whether the deque is empty or full at any time. To insert at front, move the 'front' pointer back and put the element at this position. To insert at the back, move the 'rear' pointer forward and put the element at this position. Do the opposite for deletion operations. To return the front and rear elements, just return the elements at 'front' and 'rear' positions, if it exists.

Solution:

```
class MyCircularDeque {
public:
    vector<int> deque;
    int size;
    int front;
```

```
    int rear;
    int count;

    /** Initialize your data structure here. Set the size of the
deque to be k. */
    MyCircularDeque(int k) {
        size = k;
        front = 0;
        rear = size-1;
        count = 0;
        deque = vector<int> (size, -1);
    }

    /** Adds an item at the front of Deque. Return true if the
operation is successful. */
    bool insertFront(int value) {
        if(isFull())
            return false;

        front = (front-1+size)%size;
        deque[front] = value;
        count++;
        return true;
    }

    /** Adds an item at the rear of Deque. Return true if the
operation is successful. */
    bool insertLast(int value) {
        if(isFull())
            return false;

        rear = (rear+1)%size;
        deque[rear] = value;
        count++;
        return true;
```

```
    }

    /** Deletes an item from the front of Deque. Return true if the
operation is successful. */
    bool deleteFront() {
        if(isEmpty())
            return false;

        deque[front] = -1;
        front = (front+1)%size;
        count--;
        return true;
    }

    /** Deletes an item from the rear of Deque. Return true if the
operation is successful. */
    bool deleteLast() {
        if(isEmpty())
            return false;

        deque[rear] = -1;
        rear = (rear-1+size)%size;
        count--;
        return true;
    }

    /** Get the front item from the deque. */
    int getFront() {
        if(isEmpty())
            return -1;
        return deque[front];
    }

    /** Get the last item from the deque. */
    int getRear() {
```

```
        if(isEmpty())
            return -1;
        return deque[rear];
    }

    /** Checks whether the circular deque is empty or not. */
    bool isEmpty() {
        if(count == 0)
            return true;
        return false;
    }

    /** Checks whether the circular deque is full or not. */
    bool isFull() {
        if(count == size)
            return true;
        return false;
    }
};
```

Complexity:

- Time: Each of the operations of the circular deque takes O(1) time.
- Space: O(k)

Question 6:

- **Problem link**: https://leetcode.com/problems/decoded-string-at-index/
- **Difficulty level**: Medium

Explanation:

If we have a decoded string like `appleappleappleappleappleapple` and an index like `K = 24`, the answer is the same if `K = 4`.

In general, when a decoded string is equal to some word with `size` length repeated some number of times (such as `apple` with `size = 5` repeated 6 times), the answer is the same for the index `K` as it is for the index `K % size`.

We can use this insight by working backwards, keeping track of the size of the decoded string. Whenever the decoded string would equal some `word` repeated `d` times, we can reduce `K` to `K % (word.length)`.

Solution:

```
class Solution {
public:
    string decodeAtIndex(string S, int K) {
        long size = 0;
        int N = S.size();

        // Find size = length of decoded string
        for (int i = 0; i < N; ++i) {
            if (isdigit(S[i]))
                size *= S[i] - '0';
            else
                size++;
        }

        for (int i = N-1; i >=0; --i) {
            K %= size;
            if (K == 0 && isalpha(S[i]))
                return (string) "" + S[i];

            if (isdigit(S[i]))
                size /= S[i] - '0';
            else
                size--;
        }
        return "";
    }
```

};

Complexity:

- Time: O(n)
- Space: O(1)

Question 7:

- **Problem link**: https://practice.geeksforgeeks.org/problems/circular-tour-1587115620/1/?category[]=Queue&category[]=Queue&difficulty[]=1&page=1&query=category[]Queuedifficulty[]1page1category[]Queue
- **Difficulty level**: Medium

Explanation: A queue data structure is used for an efficient approach to this problem. We first enqueue first petrol pump to the queue, we keep enqueueing petrol pumps till we either complete the tour, or the current amount of petrol becomes negative. If the amount becomes negative, then we keep dequeuing petrol pumps until the queue becomes empty.
Instead of creating a separate queue, we use the given array itself as a queue. We maintain two index variables start and end that represent the rear and front of the queue.

Solution:

```
class MyCircularDeque {
public:
    vector<int> deque;
    int size;
    int front;
    int rear;
    int count;
```

```
MyCircularDeque(int k) {
    size = k;
    front = 0;
    rear = size-1;
    count = 0;
    deque = vector<int> (size, -1);
}

    if(isFull())
        return false;

    front = (front-1+size)%size;
    deque[front] = value;
    count++;
    return true;
}

bool insertLast(int value) {
    if(isFull())
        return false;

    rear = (rear+1)%size;
    deque[rear] = value;
    count++;
    return true;
}

bool deleteFront() {
    if(isEmpty())
        return false;

    deque[front] = -1;
    front = (front+1)%size;
```

```
        count--;
        return true;
    }

    bool deleteLast() {
        if(isEmpty())
            return false;

        deque[rear] = -1;
        rear = (rear-1+size)%size;
        count--;
        return true;
    }

    int getFront() {
        if(isEmpty())
            return -1;
        return deque[front];
    }

    int getRear() {
        if(isEmpty())
            return -1;
        return deque[rear];
    }

    bool isEmpty() {
        if(count == 0)
            return true;
        return false;
    }

    bool isFull() {
        if(count == size)
            return true;
```

```
            return false;
        }
};
```

Complexity:

- Time: O(N)
- Space: O(1)

Question 8:

- **Problem link**: https://leetcode.com/problems/shortest-subarray-with-sum-at-least-k
- **Difficulty level**: Hard

Explanation:

Maintain a deque of pairs. Each element of this deque will be a pair {sum, index} where sum is the partial sum of the subarray nums[0 ... index]. Traverse the array from left to right. Calculate the sum of all elements till the current element. Delete all those elements from the back of the deque whose 'sum' is greater than the current sum. If the current sum itself is greater than k, then, the subarray from start till now may be an eligible subarray. Check if this subarray is the shortest. Next, check elements from the start of the deque. If the difference between the current sum and the 'sum' in the pair at the front of the deque is greater than or equal to 'k', then, the subarray nums[index+1 ... i] is an eligible subarray. Check if this subarray is the shortest and pop that pair from deque. If there are no eligible subarrays after traversing the entire array, then return -1;

Solution:

```
class Solution {
public:
    int shortestSubarray(vector<int>& nums, int k) {

        int n = nums.size();
        int sum = 0;
        deque<pair<int,int>> dq;

        int shortestLength = INT_MAX;

        for(int i=0; i<n; i++)
        {
            sum += nums[i];

            while(!dq.empty() && dq.back().first >= sum)
                dq.pop_back();

            if(sum >= k)
                shortestLength = min(shortestLength, i+1);

            while(!dq.empty() && sum - dq.front().first >= k)
            {
                shortestLength = min(shortestLength,
i-dq.front().second);
                dq.pop_front();
            }

            dq.push_back({sum, i});
        }

        if(shortestLength == INT_MAX)
            shortestLength = -1;

        return shortestLength;
    }
```

```
};
```

Complexity:

- Time: O(n)
- Space: O(n)