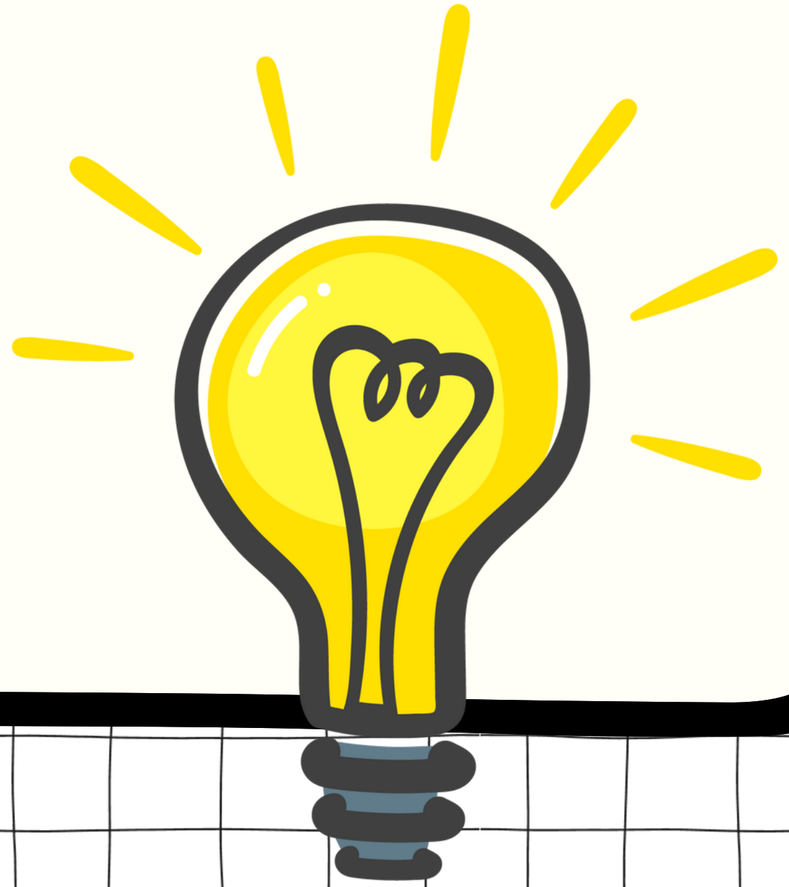
Arrays & Sliding Window



Presented by IECSE Manipal

Your Hosts

Anuj Kamath

2nd Year, CSE AIML

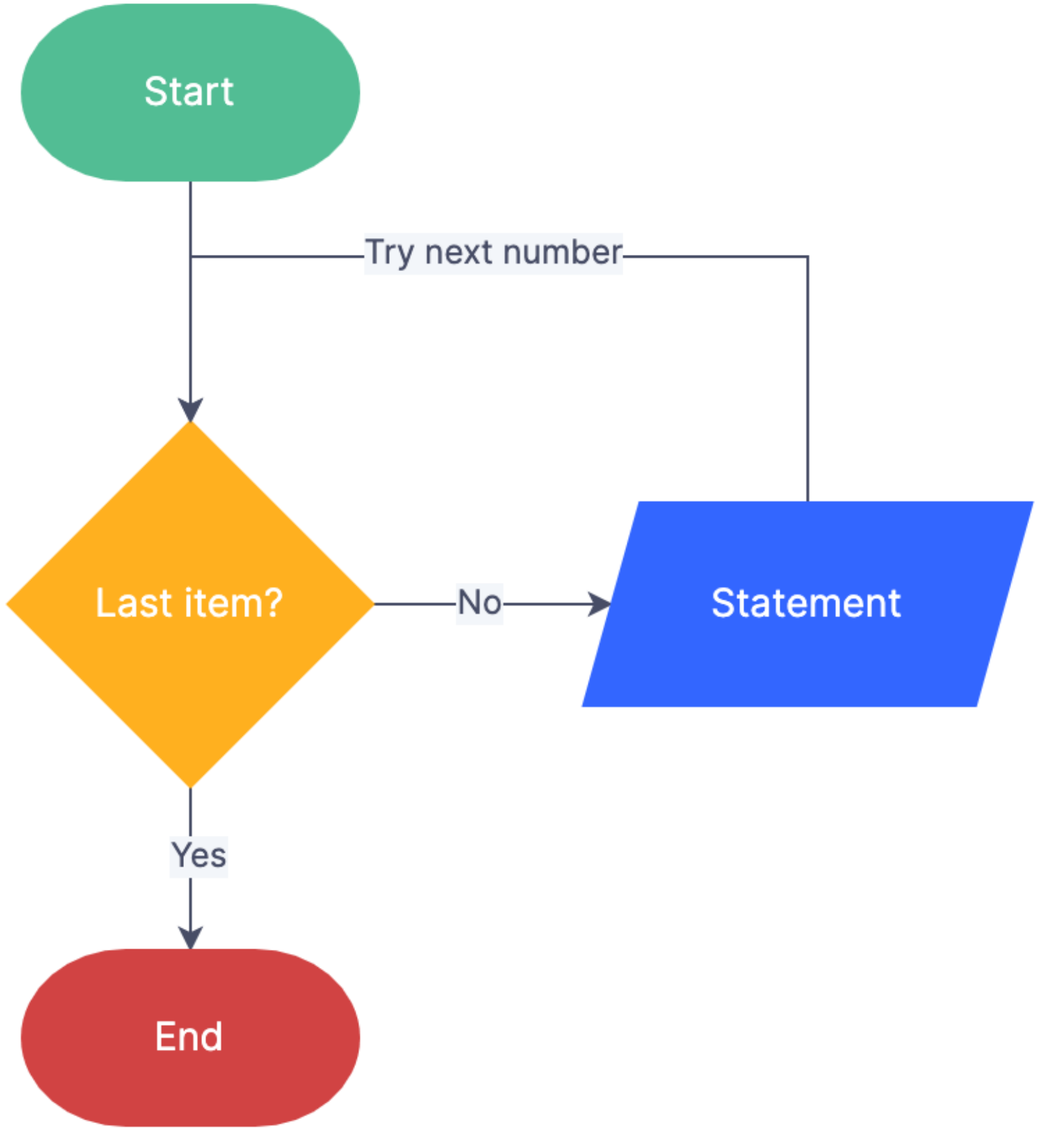
Pragadeeshwar Kannan

2nd Year, CCE

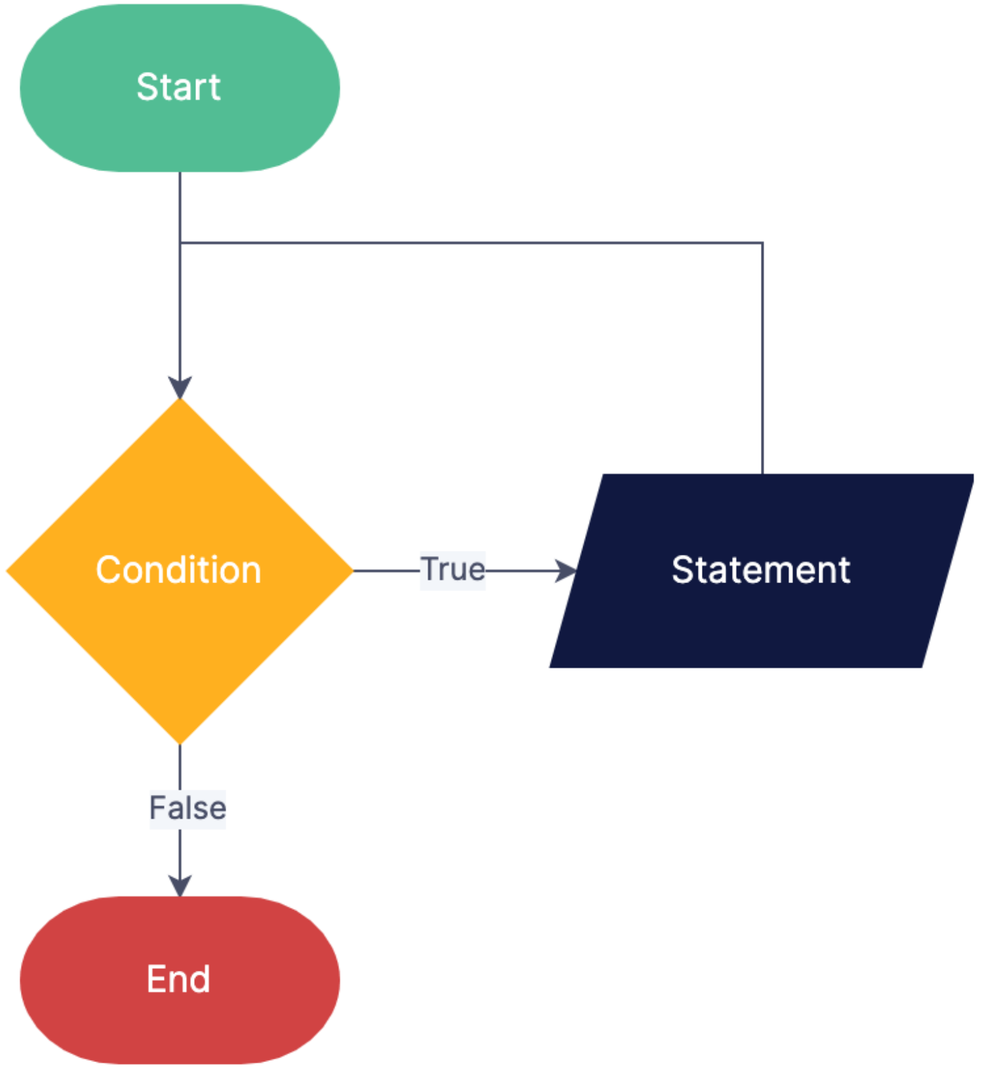
Topics

- Basics of Data Structure
  - Arrays
  - Vectors
  - Stacks
  - Queues
  - Hashmap
- Two Pointers
- Sliding Windows
- LeetCode Questions

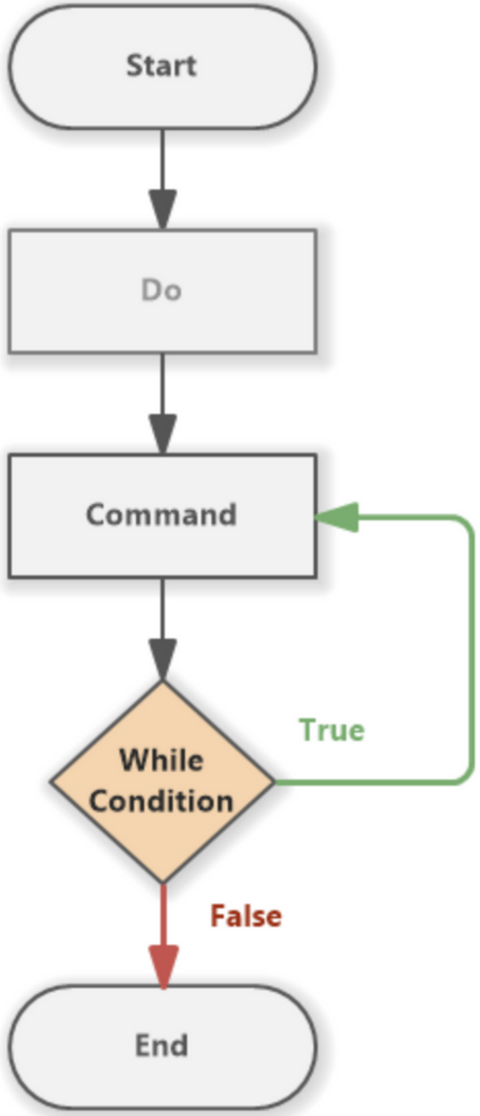
Loops: For Loop



Loops: While Loop

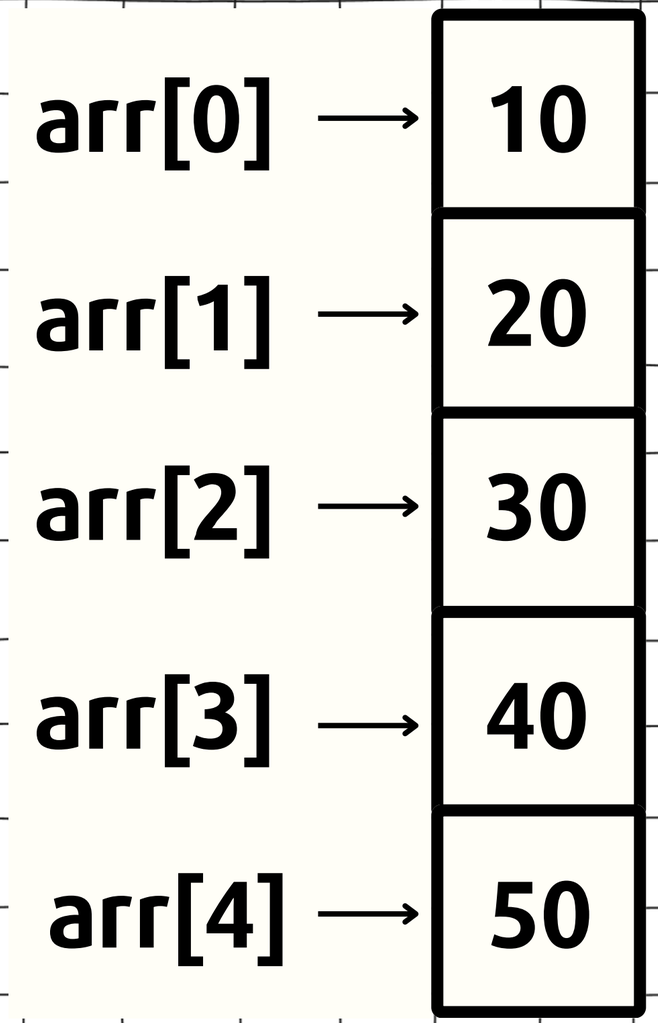


Loops: Do-While Loop

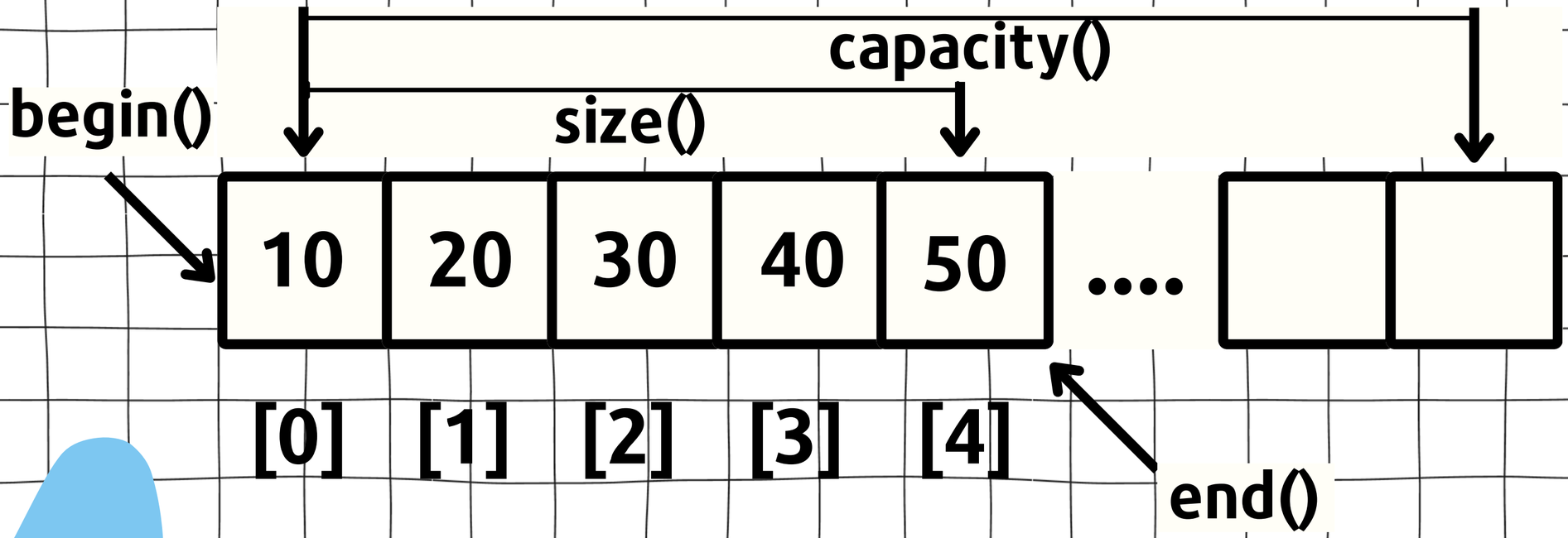


Arrays

10	20	30	40	50
[0]	[1]	[2]	[3]	[4]



Vectors



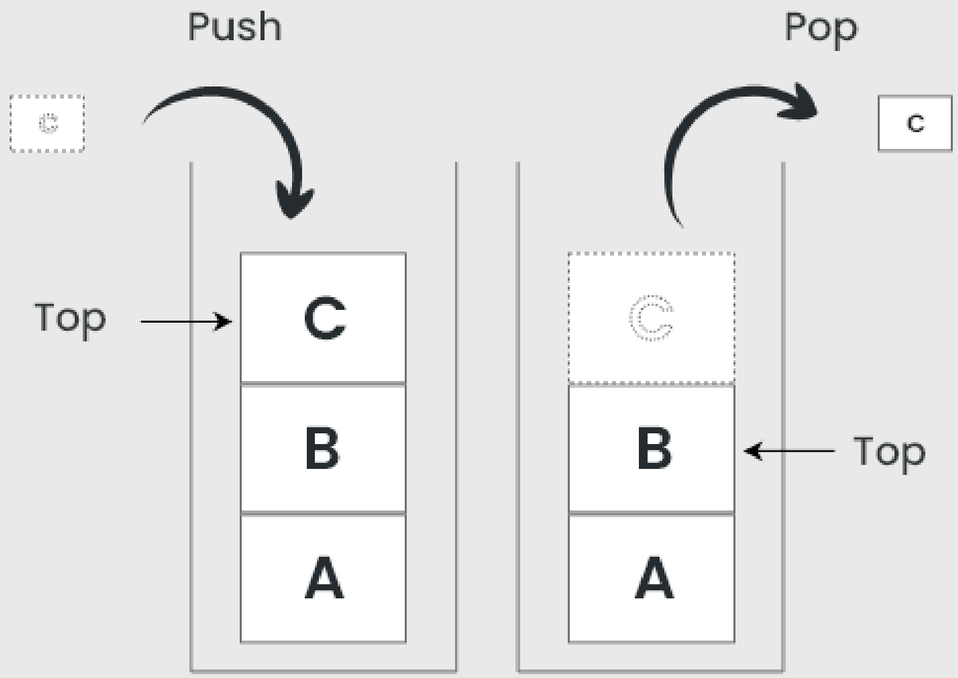
Some Important Data Structures

Stack

A stack is a linear data structure that follows the principle of Last In First Out (LIFO). This means the last element inserted inside the stack is removed first.

Stack

Data Structure

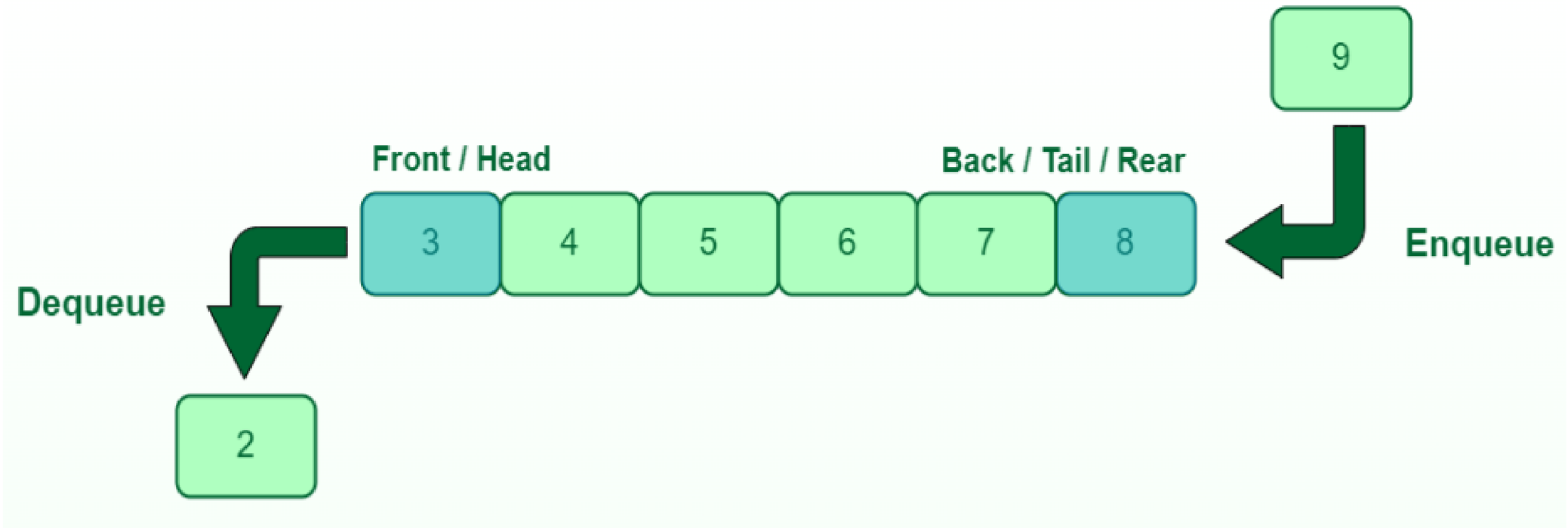


Applications of Stacks

- Undo and Redo in softwares
- Opening multiple window in the same application

Queue

A Queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.



Queue Data Structure

APPLICATIONS OF QUEUE

Dequeue

Or a Double-Ended Queue

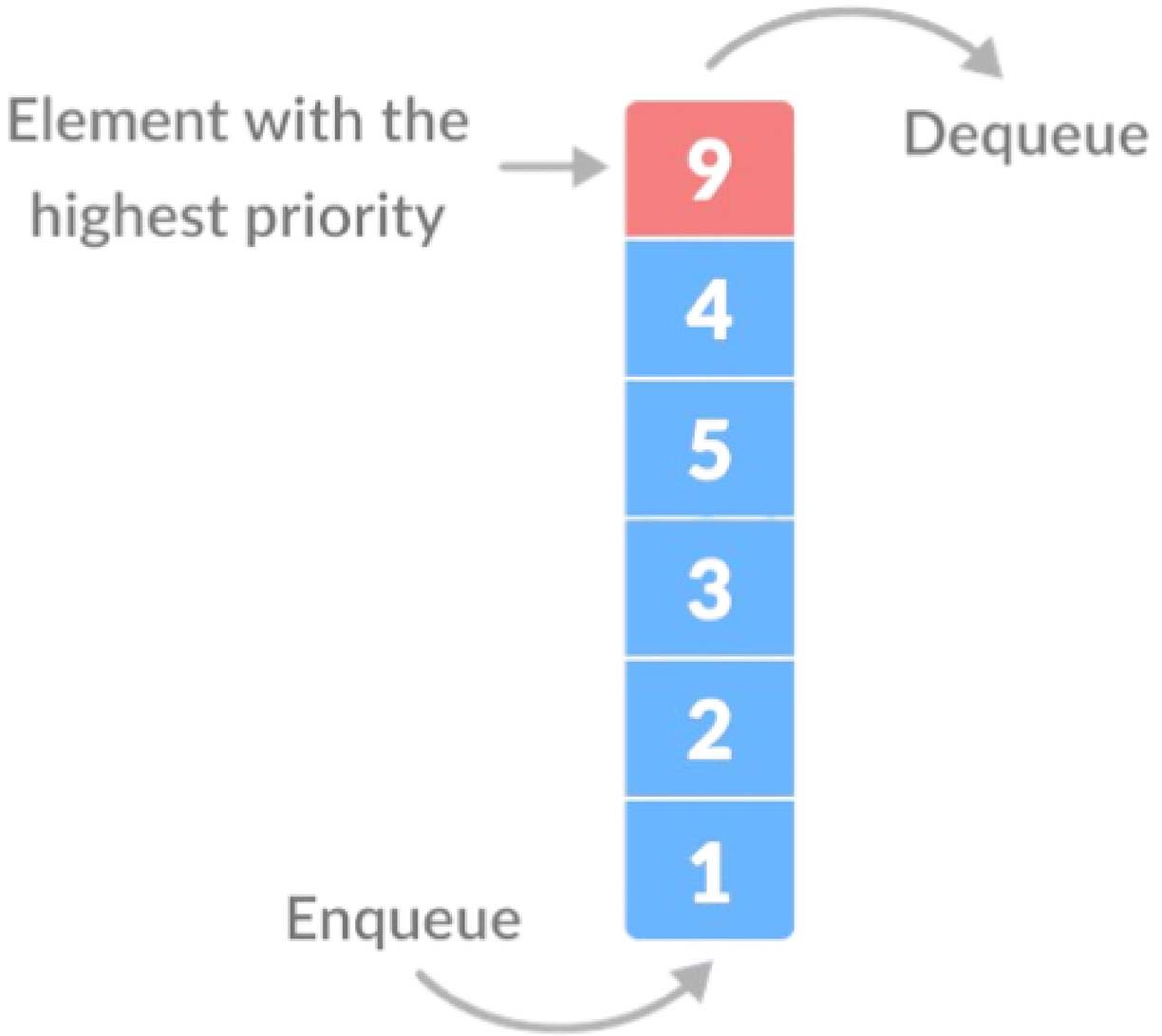
Deque is a generalized version of Queue data structure that allows insert and delete at both ends.

There are 4 processes in a DEQueue

- Enqueue at the front
- Enqueue at the rear
- Dequeue at the front
- Dequeue at the rear

Priority Queue

A priority queue is a special type of queue in which each element is associated with a priority value.

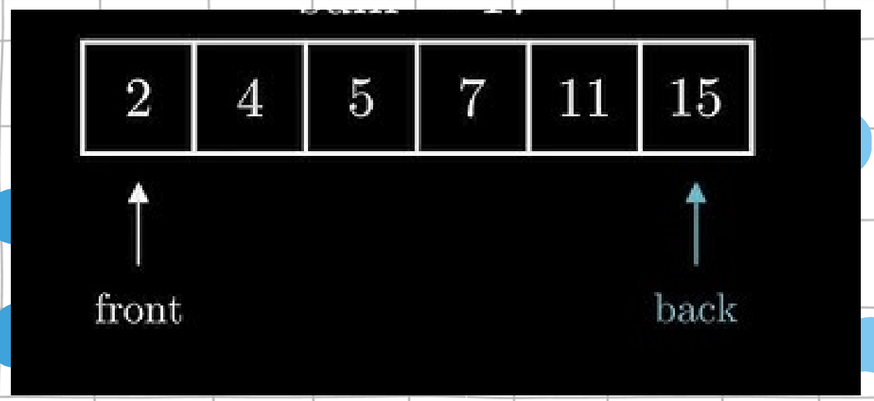

Element with the highest priority
9
4
5
3
2
1
Dequeue
Enqueue

What is the use of priority queue when we can just sort an Array?

HashMaps

Maps store pairs of data, of varying or same type, like a dictionary in Python. Each pair (<key, value>) is identified by a unique key. They may be ordered or unordered.

Two Pointers Technique



Two Pointers Technique

The Two Pointers technique involves maintaining two pointers, often initialized at different ends of an array or list. These pointers move towards each other or in a specific pattern, helping efficiently solve problems like finding pairs, detecting a subarray, or searching in a sorted array.

Given a sorted array A and an integer target,find if there exists 2 integers A[i] and A[j] such that A[i] + A[j] = 6, where i!=j.

1. i=0 j=4

sum=7 i.e.>6

2. i=0 j=3

sum=5 i.e.<6

3. i=1 j=3

sum=6 i.e.==6

LeetCode

Best Time to Buy and Sell Stock

You are given an array prices where prices[i] is the price of a given stock on the ith day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0

Input: prices = [7,1,5,3,6,4]
Output: 5

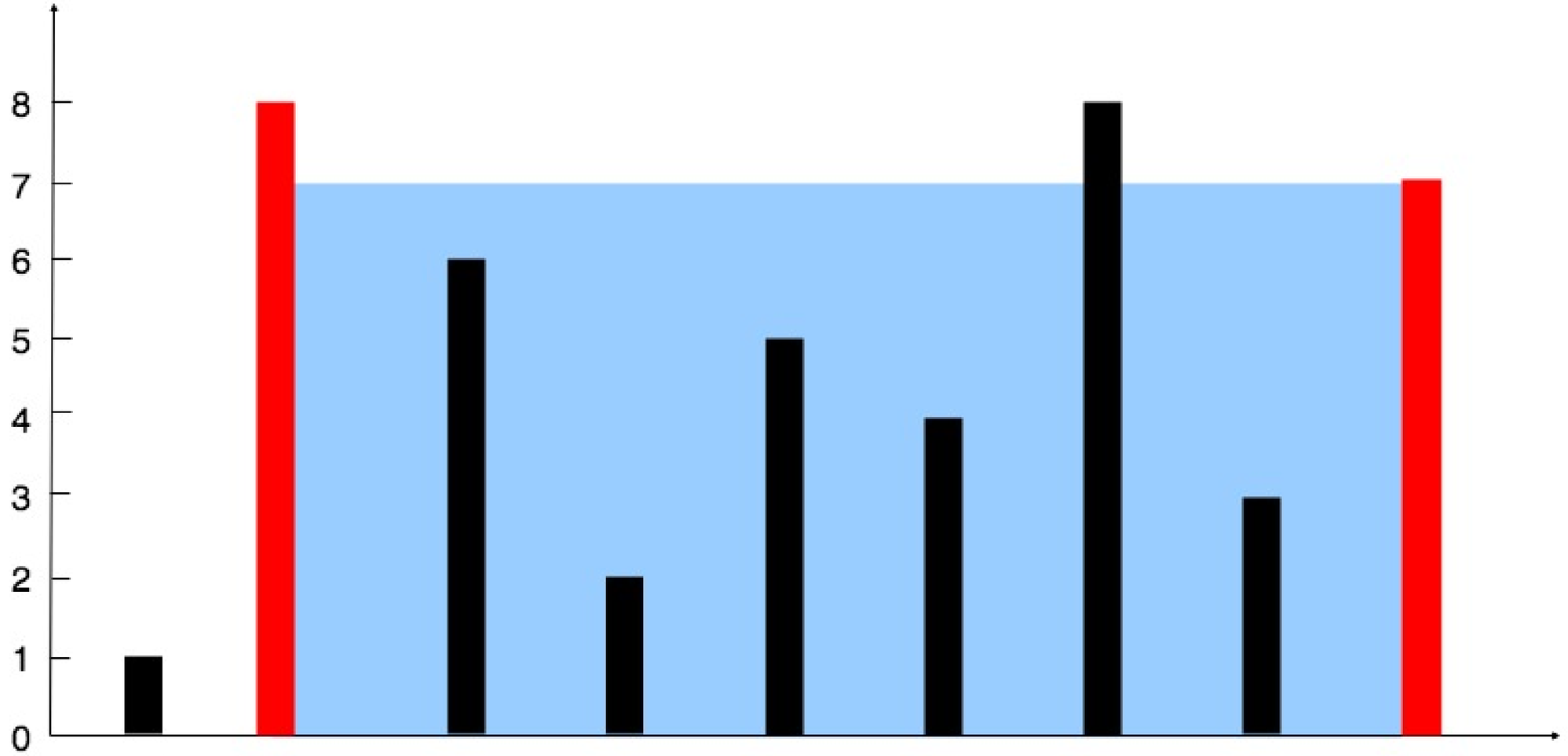
Input: prices = [7,6,4,3,1]
Output: 0

Solution: Best Time to Buy and Sell Stock

```cpp
int maxProfit(vector<int> &prices){
    int l = 0, r = 1, ans = 0;
    while (r < prices.size()){
        if (prices[l] > prices[r]){
            l = r;
            r = r + 1;
        }
        else{
            ans = max(ans, prices[r] -
prices[l]); r++;
        }
    }
    return ans;
}
```

LeetCode

Container With Most Water



Input: height = [1,8,6,2,5,4,8,3,7]

Output: 49

Solution: Container With Most Water

```cpp
class Solution {
public:
    int maxArea(vector<int>& height) {
        int right=height.size()-1;
        int left=0;
        int maxWater=min(height[right],height[left])*(right-left);
        while(left<right-1)
        {
            if(height[left]>height[right])
                right--;
            else
                left++;
            if(min(height[right],height[left])*(right-left)>maxWater)
                maxWater=min(height[right],height[left])*(right-left);
        }
        return maxWater;
    }
};
```

LeetCode

3-Sum

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

Input: nums=[-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]

Input: nums = [0,1,1]
Output: []

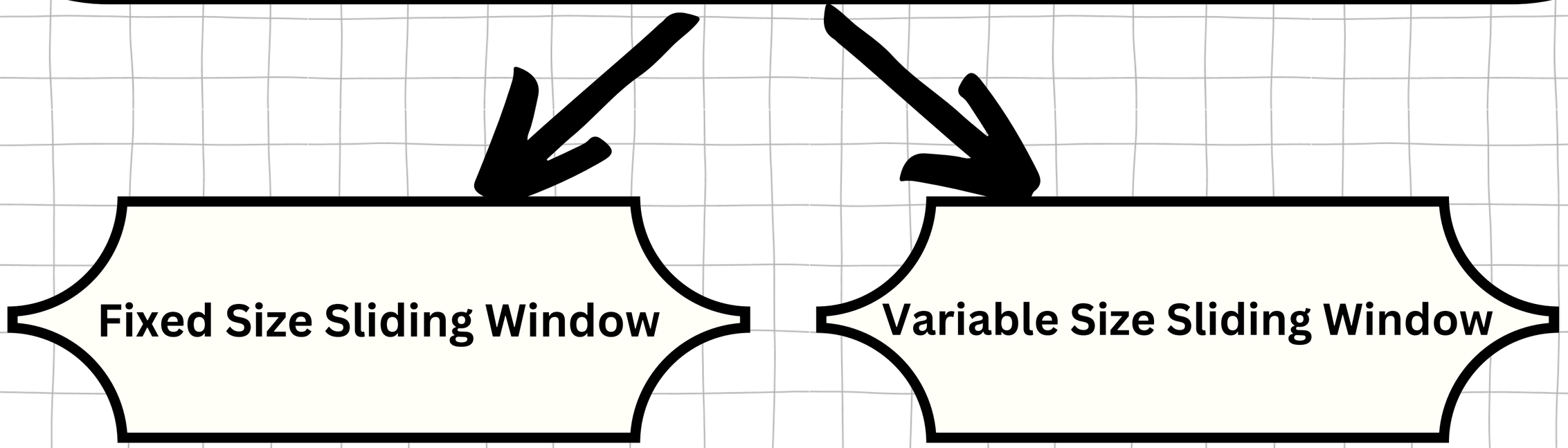
Solution: 3-Sum

```cpp
vector<vector<int>> triplet(vector<int> &num, int n){
    vector<vector<int>> ans;
    sort(num.begin(), num.end());
    for (int i=0; i<n; i++){
        if (i>0 && num[i] == num[i-1]) continue;
        int j = i+1;
        int k = n-1;

        while (j<k){
            int sum = num[i]+num[j]+num[k];
            if (sum<0){
                j++;
            }
            else if (sum>0){
                k--;
            }
            else {
                vector<int> temp = {num[i], num[j],
num[k]};         ans.push_back(temp);
                j++;
                k--;
                while (j<k && num[j] == num[j-1]) j++;
                while(j<k && num[k] == num[k+1]) k--;
            }
        }
    }
    return ans;
}
```

Sliding Window Technique

There are two types of Sliding Window Technique



LeetCode

Maximum Average Subarray I

Fixed Sliding Window

You are given an integer array nums consisting of n elements, and an integer k.
Find a contiguous subarray whose length is equal to k that has the maximum average value and return this value. Any answer with a calculation error less than 10-5 will be accepted.

Input: nums = [1,12,-5,-6,50,3], k = 4
Output: 12.75000

Input: nums = [5], k = 1
Output: 5.00000

Solution: Maximum Average Subarray I

```cpp
#include <bits/stdc++.h>
class Solution {
public:
    double findMaxAverage(vector<int>& nums, int k)
{       double ans  = 0;
        double sum = 0;
        for(int i =0; i < k;i++)
        {
            sum += nums[i];
        }
        ans = sum;
        for(int i = k ;i < nums.size();i++)
        {
            sum += nums[i];
            sum -= nums[i-k];
            ans = max(ans,sum);
        }
        return ans/k ;
    }
};
```

LeetCode

Sliding Window Maximum

Fixed Sliding Window

You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Return the max sliding window

Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [3,3,5,5,6,7]

Solution: Sliding Window Maximum

```cpp
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        int n=nums.size();
        priority_queue<pair<int,int>> pq;
        vector<int> ans(n-k+1);
        for (int i=0; i<k; i++)
            pq.push({nums[i], i});

        ans[0]=pq.top().first;
        for(int i=k; i<n; i++){

            while(!pq.empty() && pq.top().second<=(i-k))
                pq.pop(); //Pop up element not in the window

            pq.push({nums[i], i});
            ans[i-k+1]=pq.top().first;//Max element for this
window  }
        return ans;
    }
};
```

LeetCode

Longest Repeating Character Replacement

Variable Sliding Window

You are given a string s and an integer k. You can choose any character of the string and change it to any other uppercase English character. You can perform this operation at most k times.

Return the length of the longest substring containing the same letter you can get after performing the above operations.

Input: s = "ABAB", k = 2
Output: 4

Input: s = "AABABBA", k = 1
Output: 4

Solution: Longest Repeating Character Replacement

```cpp
class Solution {
public:
     int characterReplacement(string s, int k) {
        unordered_map<char, int> alphabets;
        int ans = 0;
        int left = 0;
        int right = 0;
        int maxf = 0;

        for (right = 0; right < s.size(); right++)
{           alphabets[s[right]]+= 1;;
            maxf = max(maxf, alphabets[s[right]]);

            if ((right - left + 1) - maxf > k) {
                alphabets[s[left]] -= 1;
                left++;
            } else {
                ans = max(ans, (right - left + 1));
            }
        }
        return ans;
    }
};
```

LeetCode

Longest Substring Without Repeating Characters

Variable Sliding Window

Given a string s, find the length of the longest substring without repeating characters.

Input: s = "abcabcbb"
Output: 3

Input: s = "bbbbb"
Output: 1

Solution: Longest Substring Without Repeating Characters

```cpp
class Solution {
public:
    int lengthOfLongestSubstring(string s)
    {
        if(s.size()==0)
        return 0;
        unordered_map<char,int> mp;
        int ans = 1;
        int low = 0;
        for(int i = 0;i<s.size();i++)
        {   if(mp.find(s[i])!=mp.end())//if any character is repeated
            {
                int pos = mp[s[i]];
                while(low<=pos)
                { //remove every characters from the map till the current index
                    mp.erase(s[low]);
                    low++;
                }
                mp[s[i]] = i;//store the current index
            }
            else mp[s[i]] = i;
            ans = max(ans,int(mp.size()));
        }
        return ans;
    }
};
```

More Questions

76. Minimum Window Substring (leetCode POTD Question)

567. Permutation in String

904. Fruit Into Baskets

More Questions

Thank You!

People who give feedback -> 🗿