Stacks

A stack is a container where we can
push O(1)
pop O(1) } the data at the top.
Top O(1)

Top() → To see what is present at the top.

pop() → remove the topmost element

Why use stack?
→ In a book rag we use stack (stack of books)
→ In a canteen plates stack (stack of plates)
→ Undo operation in our PC's

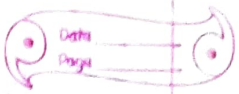
Stack Implementation

Arrays vectors linked list

Stack Array :

Problem: fixed size array

Stack is full → no more push
Stack is empty



Array (fixed)

Stack

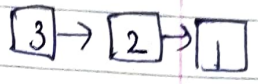
push
pop
top

Vector (Dynamic Data)

Push
pop
top.

Linked list Dynamic

Push to head.



Top → element at the head

Pop → remove the head from first node & move the head

Push → push at the head.

To be keep in mind

Stack is full

Stack is empty

Stack using linkedlist

```
template <typename T>
class Stack:

template <typename T>
class Node {
    public:
        T data;
        Node <T> * next;

        Node (T d) {
            data = d;
        }
};
```

```
template <typename T>
class Stack {
    Node<T> *head;
public:
    Stack() {
        head = NULL;
    }
    void push(T data) {
        node<T> *n = new Node<T>(data);
        n->next = head;
        head = n;
    }
    bool empty() {
        return head == NULL;
    }
    T top() {
        return head->data;
    }
    void pop() {
        if(head != NULL) {
            Node<T> *temp = head;
            head = head->next;
            delete temp;
        }
    }

int main(){
    Stack<char> S;
    S.push('n');
    S.push('e');
    S.push('l');
    S.push('l');
```

```
    while (!s.empty())
        cout << s.top();
        s.pop();
    }

return 0;
}
```

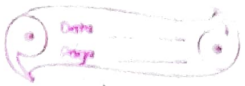
Stack using vector

```
# include <vector>

template <typename T>
  class stack {
    vector <T> arr;

  public:
      void push (T data) {
        arr.pushback (data);
        }

      void pop () {
        arr.pop_back();
      }

        T top() {
            int lasti = vsize()-1;

            return arr[lasti];
          }

      bool empty() {
          return arr.size == 0;
        }
};
```

```
int main() {
    stack <int> s;
    s.push(1);
    s.push(2);
    s.push(3);
    s.push(5);
    s.push(10);

    while (!s.empty()) {
        cout << s.top();
        s.pop();
    }
```

Stack STL

```
#include <stack>

int main() {
    stack <string> books;
    books.push("C++");
    books.push("Java");
    books.push("Python");

    while (!books.empty()) {
        cout << books.top();
        books.pop();
    }

    return 0;
}
```



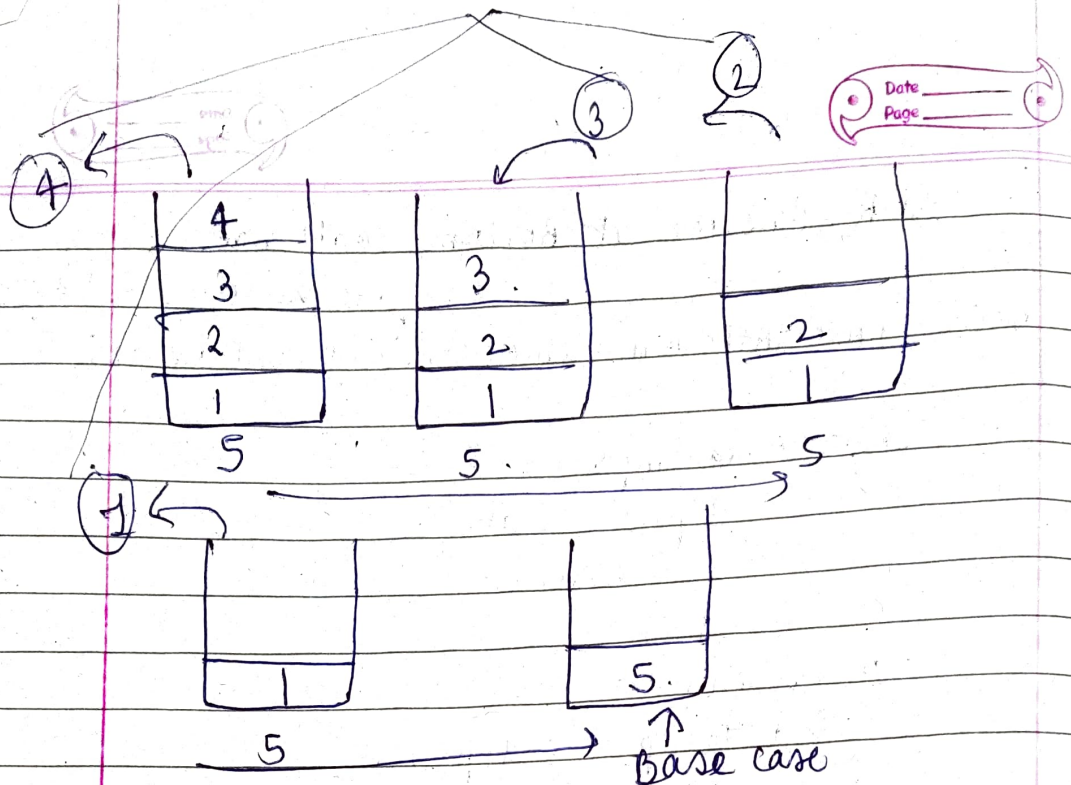
Stack Insert at Bottom Challenge

→ to modify the original stack

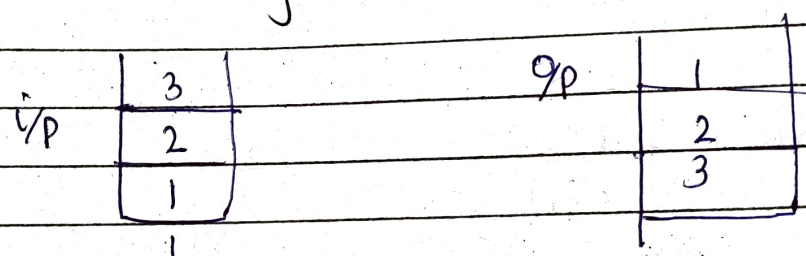
Solve it recursively

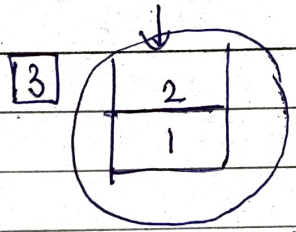
```
void insertAtBottom (stack<int>&s, int data) {
    if (s.Empty()) {
        s.push(data);
        return;
    }
    // rec case
    int temp = s.top();
    s.pop()
    insert at Bottom(s, data);
    s.push(temp);
}

int main() {
    stack<int> s;
    s.push(1);
    s.push(2);
    s.push(3);
    s.push(4);
    insertAtBottom(s, 5);

    while(!s.empty()){
        cout << s.top();
        s.pop();
    }
return 0;
}
```

in a temp variable

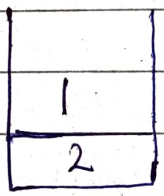


Recursively reverse a stack



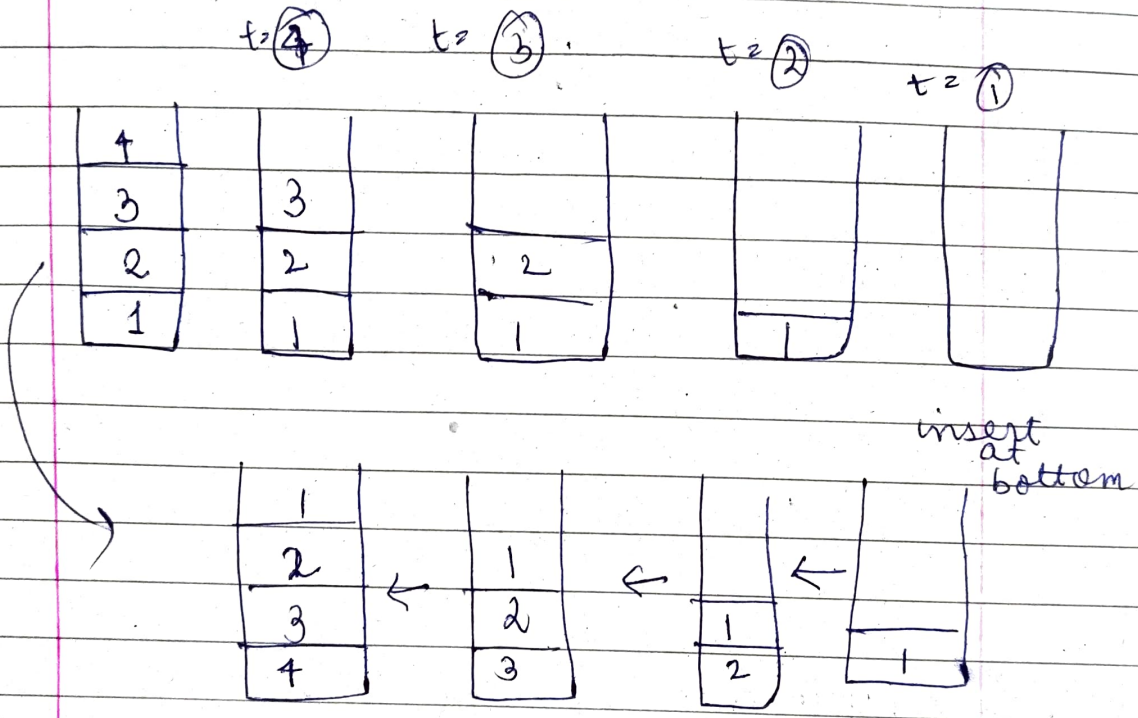


Reverse the smaller stack.

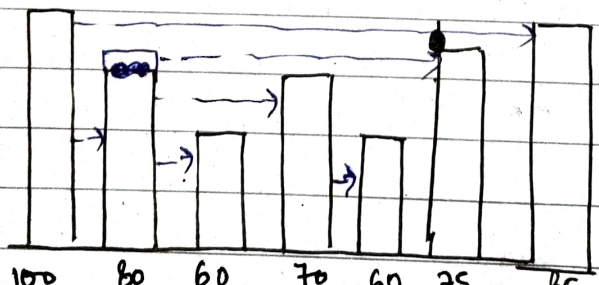


Now insert 3 at the bottom of the stack.

```
void reverse(stack<int> &s){
    if(s.empty())
        return;

    int t = s.top();
    s.pop();
    reverse(s);
    insertAtBottom(s, t);
}
```



Stock Span problem



Price	100	80	60	70	60	75	85
Days	0	1	2	3	4	5	6
span	1	1	1	2	1	4	6

Given a list of prices for stock for n no of days. Calc. span of stock price for each day.

span - the no of days for which the curr. price is highest

for 70
prev highest was 80.
so span of 70 = idx of (70) - idx (80)
= 3 - 1
= 2.

for 60
so span of 60 = idx of (60) - idx (80)
= 1

Span of day zero = 1

Brute force Approach: for every day iterate in left direc^n. find the first highest day & now.
i - idx of prev highest

$O(N^2)$

Efficient way: Take a stack. push the first day in the stack along with indices.

(6, 85)	span = 6 - 0 = 6.
~~5, 75~~	span = 5 - 1 = 4
~~4 60~~	span = 4 - 3 = 1
~~3, 70~~	span = 3 - 1 = 2
~~(2, 60)~~	span = 2 - 1 = 1
~~(1, 80)~~	span = 1 - 0 = 1
(0, 100)	span = 1

If the curr val being pushed is less than the top of stack so span would be calc. w.r.t the top.

When top is smaller than current element then pop the top & push the curr element along with the idx.

Corner case : If stack becomes empty consider curr = 185 so we do index + 1

```
void StockSpan (int prices[], int n, int span[]){
    Stack<int> s;  // indices of the days.
    s.push(0);
    span[0] = 1;

for (int i=1; i<n-1 ; i++)
// loop for rest of the days.
    int currPrice = price[i];
    // topmost element i.e. higher than curr
    while (!s.empty() && price[s.top()] <= currPrice)
        s.pop();
    }
    if (!s.empty()) {
        int prevHighest = s.top();
        span[i] = i - prevHighest; }
    else {
    }   span[i] = i+1;
```

```
        // push element into stack
        S.push(i);
    }
}

int main(){
    int prices[] = {100, 80, 60, 70, 60, 75, 85};
    int n = sizeof(prices)/sizeof(int);
    int span[10000] = {0};
    stockspan(prices, n, span);
    for(int i=0; i<n; i++) cout << span[i];
```

TC
O(N)
↓
Each element is pushed once & poped once.

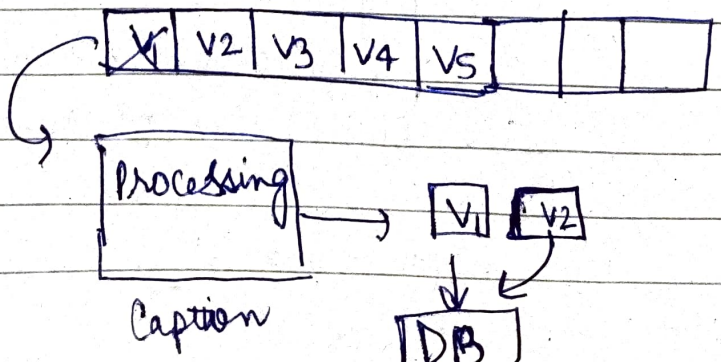
Queues

Stack → lifo (Last In first Out)
Queue → fifo (first in first out)

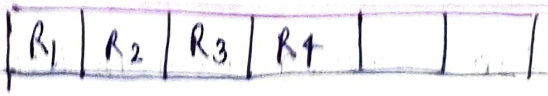
A queue in Bank (P1 P2 P3 P4) ← Push, → Pop

Operations
1) Push O(1) — Enqueue
2) Pop (removing from front) O(1) — dequeue
3) Front O(1)

Applications
1) Building system for youtube



Queue of Request when there is so much load on load balancer



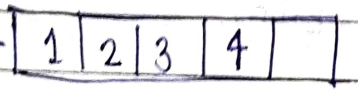
Queue Implementation

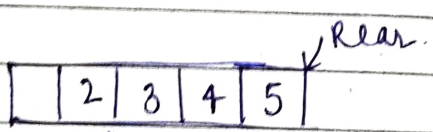
Implement a queue

Queue{
CS = 0
mS =

if (CS == mS) queue is full

1) Array





Push : O(1)
Pop : O(1)
front : O(1)

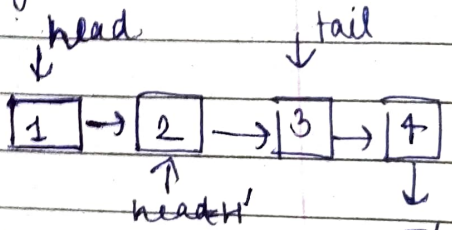
Insertion happens at (Rear + 1)
Removal → (front)

We need to maintain 2 pointer.

We don't move elements one step further because it would take lot of time.

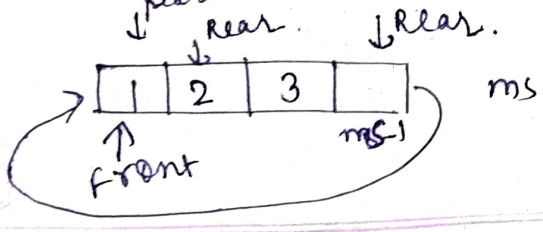
2) Linkedlist

Dynamic nodes



Insertion at tail is done

Insertion at tail : O(1)
Deletion at head : O(1)



Queue using Circular Array

```
class Queue {
    int * arr;
    int cs;
    int ms;
    int front;
    int rear;

public:
    Queue (int defaultSize = 5) {
        cs = defaultsize;
        arr = new int [ms];
        cs = 0;
        front = 0;
        rear = ms-1;
    }
void push (int data) {
    if (! full () )
    {
    // take rear to next index
        rear = (rear + 1) % ms;
        arr [rear] = data;
        cs++;
    }

}
bool full (){
    return cs == ms;
}
```

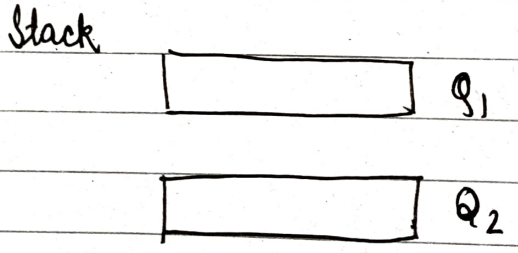
```
bool empty(){
    return cs==0;
}

void pop() {
    if(!empty()) {

        // take front pointer forward.

        front = (front+1) % ms;
        cs--;
    }
}

int qfront() {
    return arr[front];
}
};

int main(){

    Queue q(7);
    q.push(1);
    q.push(2);
    q.push(3);
    q.push(4);
    q.pop();
    q.push(5);

    while(!q.empty())
        cout << q.front();
    q.pop();
```

Queue STL

```
#include <queue>

int main(){

    queue<int> q;
        q.push(1);
        q.push(2);
        q.push(3);

    while(!q.empty()){
        cout << q.front() << " "; }.
    q.pop();
```

Stack using 2 queues

Implement a stack class which uses 2 queues internally as a data structure.

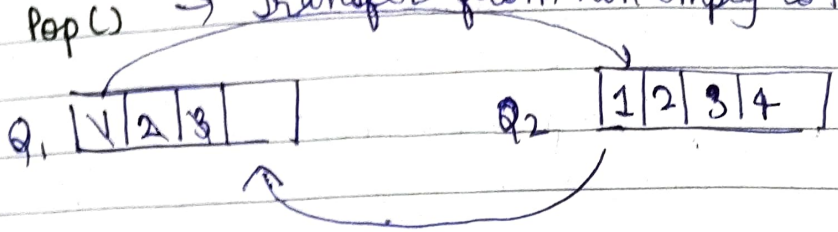


Push()
Pop()
top()

O(1) Push()

pick one empty Q → Q_1 (push it into Q_1)

O(N) Pop() → Transfer from non empty to the empty queue



Using queues interchangebly.

O(N) Top()

Stack using 2 queues

```
class Stack {
    queue <int> q1, q2;
public:
    void push (int x) {
        if (! q.empty()) {          // will insert in the non-empty q
            q1.push(x);
        }
        else {
            q2.push(x);
        }
    void pop() {
        if (q1.empty())
        // shift the elements from q2 to q1
```

Q1 | 1 | 2 | 3 |

Q2 | 4 | 2 | 3 | 4 |

If after removing 4 q2 becomes empty then break

```
while (!q2.empty()){
    int front = q2.front();
    q2.pop();

    if (q2.empty()) {
        break;
    }
    q1.push(front);
}
else {
    // shift the elements from q1 to q2.
    while (!q1.empty())
        int front = q1.front();
        q1.pop();
        if (q1.empty()){
            break;
        }
        q2.push(front);
    }
}

int top() {
    if (q1.empty()) {
    // shift the elements from q2 to q1.
    while (!q2.empty()){
        int front = q2.front();
        q2.pop();
        q1.push(front);
```

```
bool empty () {
    return q1.empty() && q2.empty();
}

    if (q2.empty()) {
        return front;
    }
  }
}
else {
    while (!q1.empty()) {
        int front = q1.front();
        q2.push(front);
        q1.pop();

        if (q1.empty()) {
            // top element of the stack
            return front;
        }
    }
}
};

int main() {
    Stack s;
    s.push(1);
    s.push(2);
    s.pop();
    s.push(3);
    s.push(4);
    s.push(5);
    while (!s.empty())
        cout << s.top() << " ";
    s.pop();
```