# CLASS-25

why / when to use stack

functions

## Next Greater Element

[1, 3, 2, 4] → right side first element greater than $i^{th}$ element

⇒ {3, 4, 4, -1}

↳ No greater exist on right side

[6, 8, 0, 1, 3] → -1

⇒ [8, -1, 1, 3, -1]

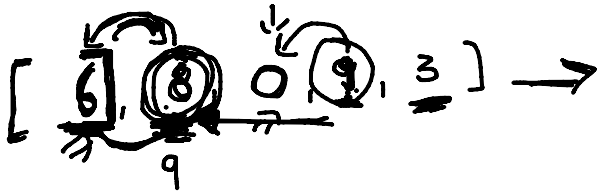
Brute Force solution:

```
vector<int> ans(n);
For(i=0; i<n; i++)
{   int temp = -1;
    for(j=i+1; j<n; j++)
    {   if(a[j] > a[i])
        {   temp = a[j]; break; }
    }
    ans[i] = temp;
}
```

T.C = ? $O(n^2)$

$n + n-1 + \cdots + 1 = n^2$

optimized

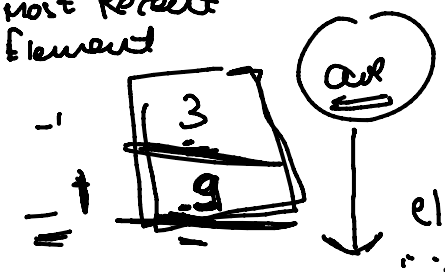


ans[6] = 9

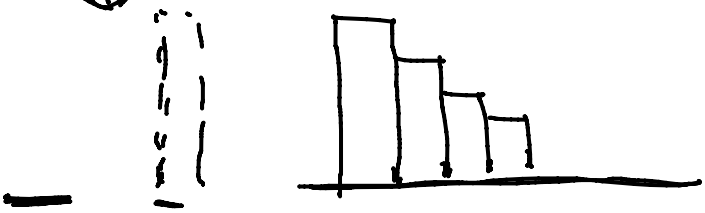
ans[0] = 9

ans[8] = 9

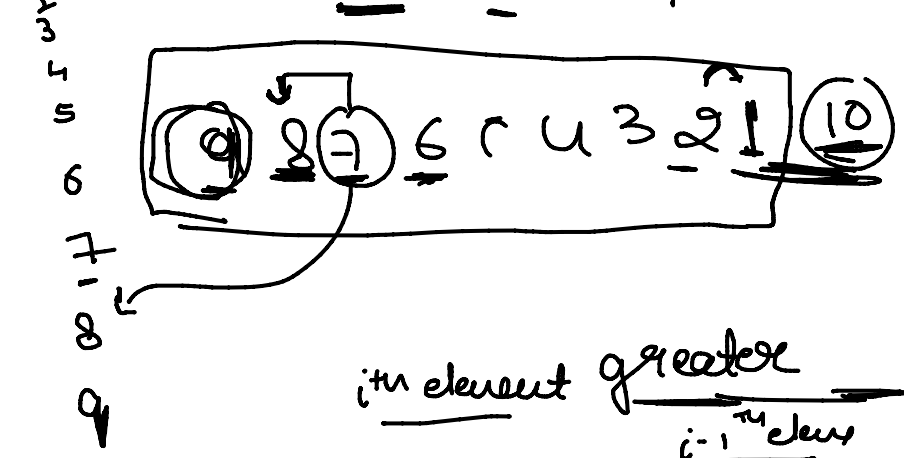
top → most Recent Element



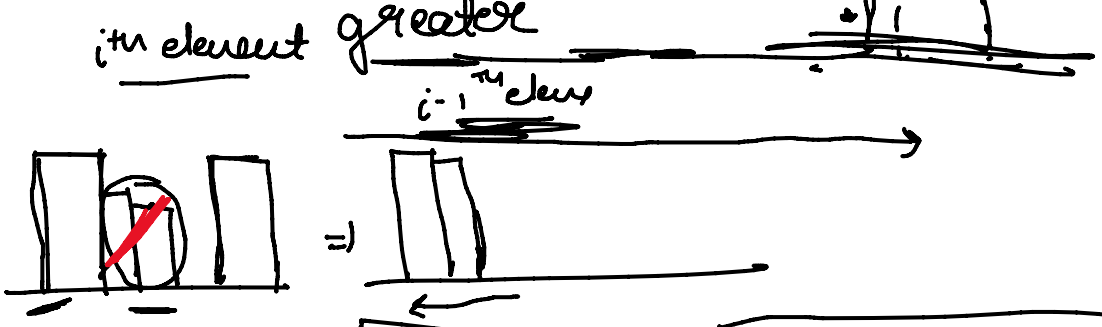
elements are X



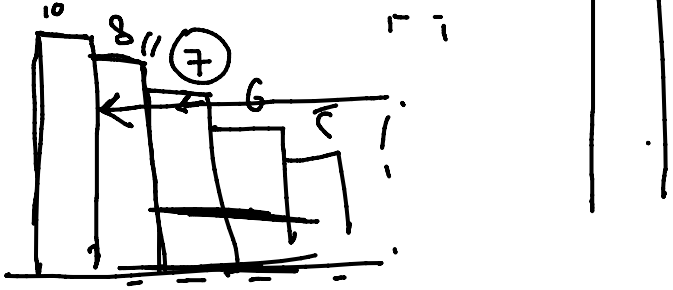
Pattern



ith element greater



building 1 elements



stack <int> s;

stack → index

```
stack<int> s;
s.push(0);  // index
vector<int> ans(n,-1);

for(i=1; i<n; i++)
{   while(!st.empty() and nums[i] > nums[s.top()])
    {   ans[s.top()] = nums[i];
        s.pop();
    }
    s.push(i);
}
```

stack → | index |
| ans |

Removed all buildings smaller than i'th ele

---

Daily Temperatures → Next Greater Element

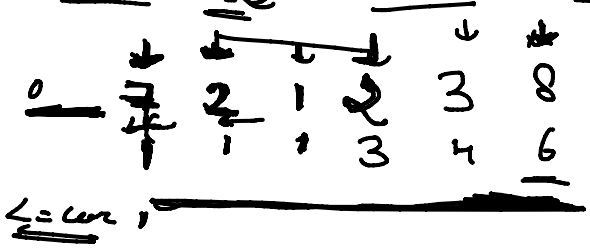
[73, 74, 75, 71, 69, 72, 76, 73]
 1   1   4   2   1   1   0   0

x = 6 - 2 = 4

ith element → Next Greater Element

```
stack<int> s;
s.push(0);
vector<int> ans(n,0);
for(i=1; i<n; i++)
{   while(!st.empty() and nums[i] > nums[s.top()])
    {   ans[s.top()] = i - s.top();
        s.pop();
    }
    s.push(i);
}
```

# 739 Stock Span Problem

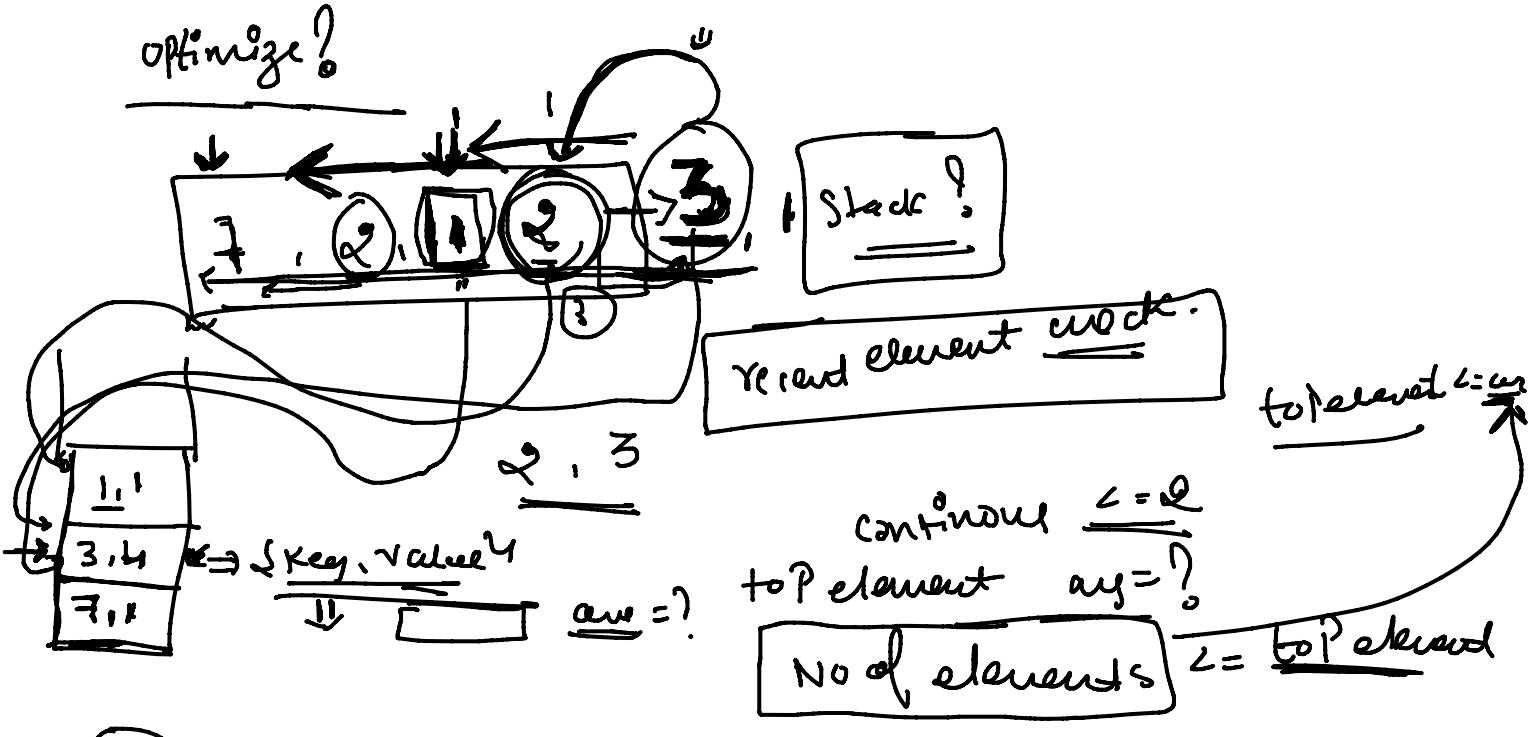



## Brute Force

```
for (i=0; i<n; i++)
{ int ans=1;
  for(j=i-1; j>=0; j--)
  { if (num[i]>num[j])
      break;
    ans++;
  }
}
```

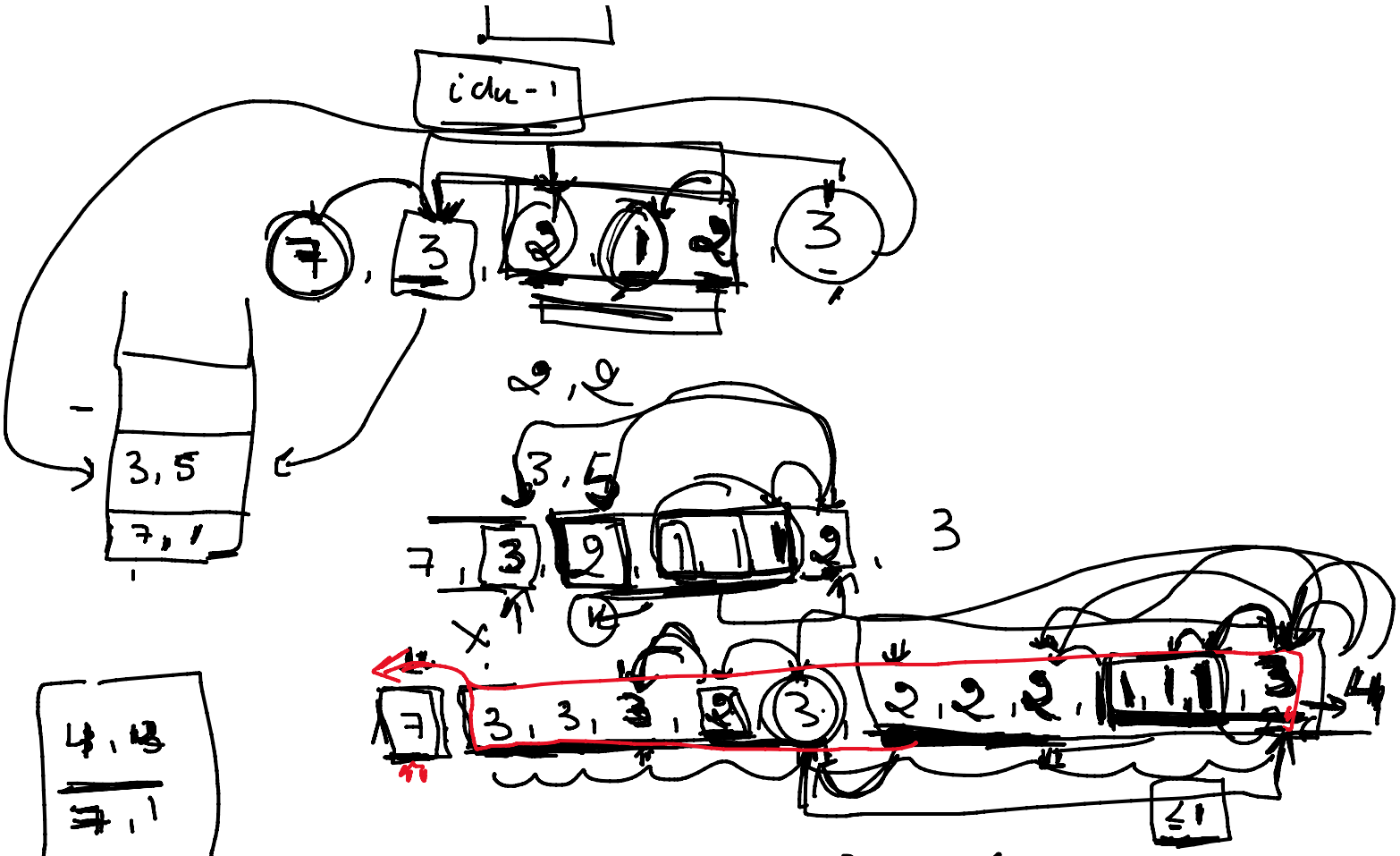
O(n²)

optimize?




?

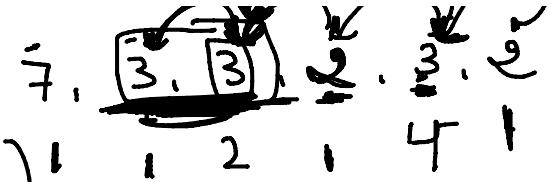
curelement




7 > 3 stack < pair < int, int >> s

↳ friend

```
int next ( int price )
{   int ans = 1;
    while ( !s.empty() and price >= s.top().first )
        {   ans += s.top().second;
            s.pop()
        }
    s.push({price, ans})
}
return ans;
```




(2,1)
(3,4)
(7,1)

---

# Implement Queue using stacks

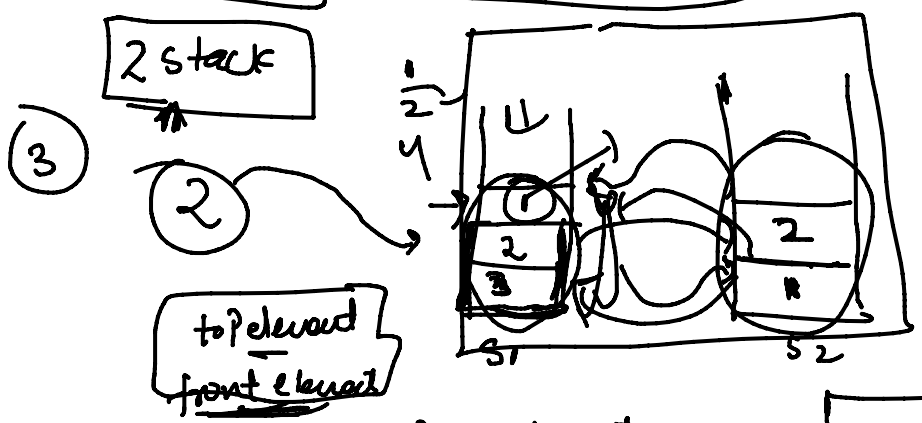
Queue → front()
→ push()
→ pop()
⇒ using Array FIFO

Stack (famous) → LIFO
→ Topelement (last inserted element)
→ push()
→ pop()

1 stack

Stack and queue

2 stack




topelement
frontelement

1, 2
1, 2, 3

T.C = O(n)

Push element O(2n)

Element Push O(2n)

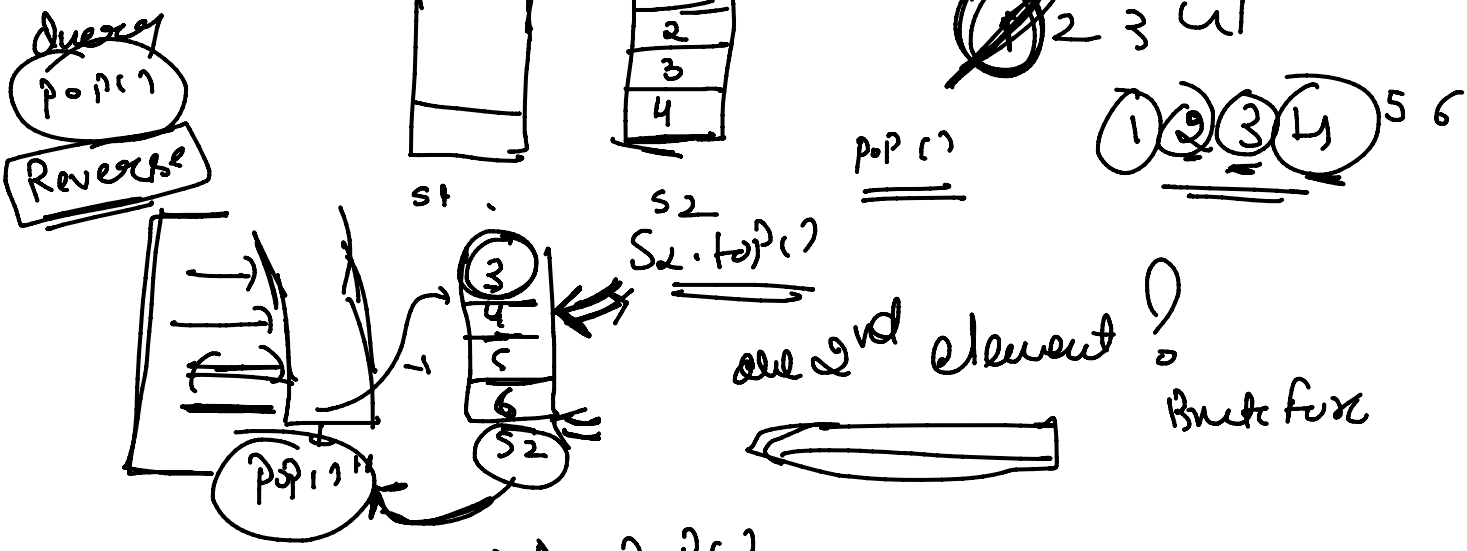
pop() → O(1)
push() → O(n)
top() → O(1)

O(n)

top() → O(1)

2nd Way :- Pop operation costly Push O(1)

queue




one 2nd element ?

Brute for

Pop()

```
while pop()
{ if(S2.empty())
  { // Shift all elements from S1 to
        S2 until only 1 element left in S1

  }
  else
  { Direct Pop from S2;
  }
```

O(n) 1

log n

(1) Making Push operation costly

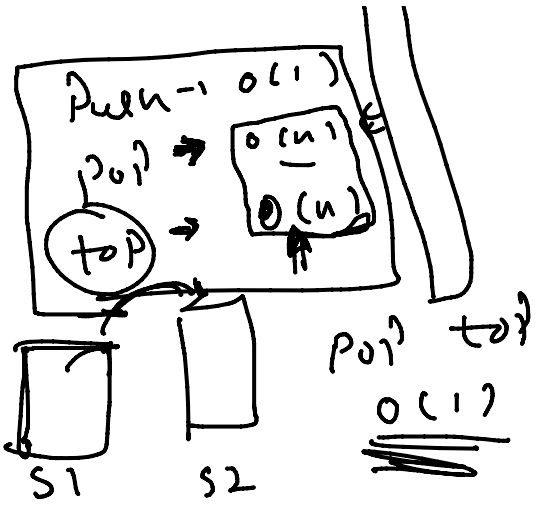
```
Stack<int> S1, S2;
void Push(int x)
{ // empty S1 and shift to S2
```

```
{ // empty S1 and shift to S2
    while (!S1.empty())
    { S2.push(S1.top());
      S1.pop();
    }
    S1.push(x);
    // Bring Back all S2 to S1
    while (!S2.empty())
    { S1.push(S2.top());
      S2.pop()
    }
}

void pop()
{ if (S1.empty())
    cout << "Queue is empty";
  S1.pop();
}

int front()
{ x=>
  =
  return S1.top();
}
```

Method 2 :- Make Pop costly




```
Void Push(int n)
{  S1.Push(n);
}
Void Pop()
{  if(S1.empty() and S2.empty())
   {  cout << "Underflow";
   }
   else if(S2.empty())
   {  while(!S1.empty())
      {  S2.Push(S1.top());
         S1.Pop();
      }
   }
   int k = S2.top();
   return k
}
```