Date _____ Page _____

Day - 140

Stack - 6

* **Get min. element from stack:**

⇒ We have to perform, three operations — in O(1) T.C.

push → O(1)
pop → O(1)
get min → O(1)

⇒ We can easily perform push & pop in O(1) T.C. & but we can't perform get min in O(1) easily.

⇒ So, we have to figure this out.

⇒ So, we will use an another stack & for every value in the original stack, we will maintain a min value for that corresponding stack value.

⇒ This will take O(1) T.C.

Code

```
stack<int> st1;
stack<int> st2;
void push(int x){
    if(st1.empty){
        st1.push(x);
        st2.push(x);
    }
```

Date Page

```
        else{
            st1.push(x);
            st2.push(min(x, st2.top()));
        }
}

int pop(){
    if(st1.empty()){
        return -1;
    else{
        int element = st1.top();
        st2.pop();
        st2.pop();
        return element;
    }
}

int getmin(){
    if(st1.empty())
        return -1;
    else{
        return st2.top();
    }
}
```

⇒ We have to solve it in constant space — we can't use another stack.

Date _____ Page _____

⇒ Now, we have to need a min element at every value.

⇒ So, for this, we can use the approach of storing two numbers in one no.

⇒ This will by using '%' operator.

⇒ So, we will store both no. & min no at that place in the stack.

⇒ So, according to our question, we will get value in the range between 1 to 100 :

⇒ So,

number = ori. no. × 101 + min. element

For ex:

Ori. no = 2 , min. element = 2

number = 2 × 101 + 2

= 202 + 2 = 204

Code

```
stack <int> st1;
```

```
void push(int x){
    if(st1.empty()){
        st1.push(x*101+x);
    else
        st1.push(x*101+min(x, st1.top()%101));
}
```

Date _____ Page _____

```
int pop(){
    if( st.empty()){
        return -1;
    }
    else{
        int element = st1.top()/101;
        st1.pop();
        return element;
    }
}

int getmin(){
    if (st1.empty())
        return -1;
    else
        return st1.top()%101;
}
```

Note:

1. Only applicable for limited range.
2. / → Original no.
   % → Minimum no.

* Max. of min. for every window size:

10	20	15	50	10	70	30

k = 3

Date _____ Page _____

⇉ First, we have to create windows —

→ min → 10 15 10 10

{10, 20, 15}, {20, 15, 50}, {15, 50, 10}, {50, 10, 70},

10

{10, 70, 30}

⇉ Min from every window → 10, 15, 10, 10, 10

⇉ Max of min of every window → **15** → (ans for (when window size is 3))

⇉ We have to find this for all window sizes → 1, 2, 3, ...

⇉ k=1 → {10}, {20}, {15}, {50}, {10}, {70}, {30}

ans → 70

⇉ Same for k = 2, 3, 4, 5, 6, 7 ....

⇉ In the end, we will get an array —

	0	1	2	3	4	5	6
ans →	70	30	15	10	10	10	10

⇉ First, we will create an array 'ans' to store ans.

⇉ After that, we will run a loop of size of array times.

Date _____ Page _____

⇉ For how many window in that window size is equal to 'size of array - i'.

⇉ window size = i+1.

⇉ Indexing for elements —
from 'j' to 'j+i+1'.
i → Outer loop, j → Inner loop.

⇉ We will take a variable num to store min of that window.

⇉ After that, we will find max from all of the windows.

⇉ In the end, return ~~num~~ ans array.

Code

```
⇉ for(i=0; i<n ; i++){
      for(j=0; j<n-i; j++){
          int num = INT_MAX;
          for(k=j; k<j+i+1; k++){
              num = min(num, arr[k]);
          }
          ans[i] = max(ans[i], num);
      }
  }
  return ~~num~~ ans;
```

T.C. → $O(n^3)$

Date _____ Page _____

⇉ Approach to solve in $O(n^2)$.

⇉ Instead of running or solving in horizontal direction, we will solve it in vertical direction.

⇉ Approach to solve in $O(n)$.

⇉ In the previous approaches, we are finding window and then min element.

⇉ Now, we will reverse this process.

⇉ We will select any element from array & check for which window size, it will be min.

⇉ We will do this for all elements & if we get any greater element then we will change that ans with greater element.

⇉ For finding window size, we will do —
NSR − NSL − 1

⇉ After that we will update the max value for every every window size, to that element.

⇉ This approach will also take $O(n^2)$ T.C.

⇉ So, to make it more optimized, we will fill only the last window size for that element.

⇉ In the end, we will do a traversal from end-1 element to start & check with next element for max.

Date Page

Code

```
vector<int> ans (n,0);
stack < int> st;
for(i=0; i<n; i++){
    while(!st.empty() && arr[st.top()] >= arr[i])
        int index = st.top();
        st.pop();
        int range = i - st.top()
        if (st.empty()){
            int range = i;
            ans[range-1] = max(ans[range-1],
                arr[index]); ans[ra
        }
        else{
            int range = i-st.top-1;
            ans[range-1] = max(ans[range-1],
                arr[index]);
        }
    }
    st.push(i);
}
while( !st.empty()){
    int index = st.top();
    st.pop();
    if( st.empty()){
        range = n;
        ans[range-1] = max(ans[range-1],
                arr[index]);
    }
```

Date Page

```
        else{
            range = n - st.top() - 1;
            ans[range-1] = max(ans[range-1],
                               arr[index]);
        }
    }
    for(i = n-2; i>=0; i--){
        ans[i] = max(ans[i], ans[i+1]);
    }
    return ans;
```

T.C. → O(n)
S.C. → O(n)