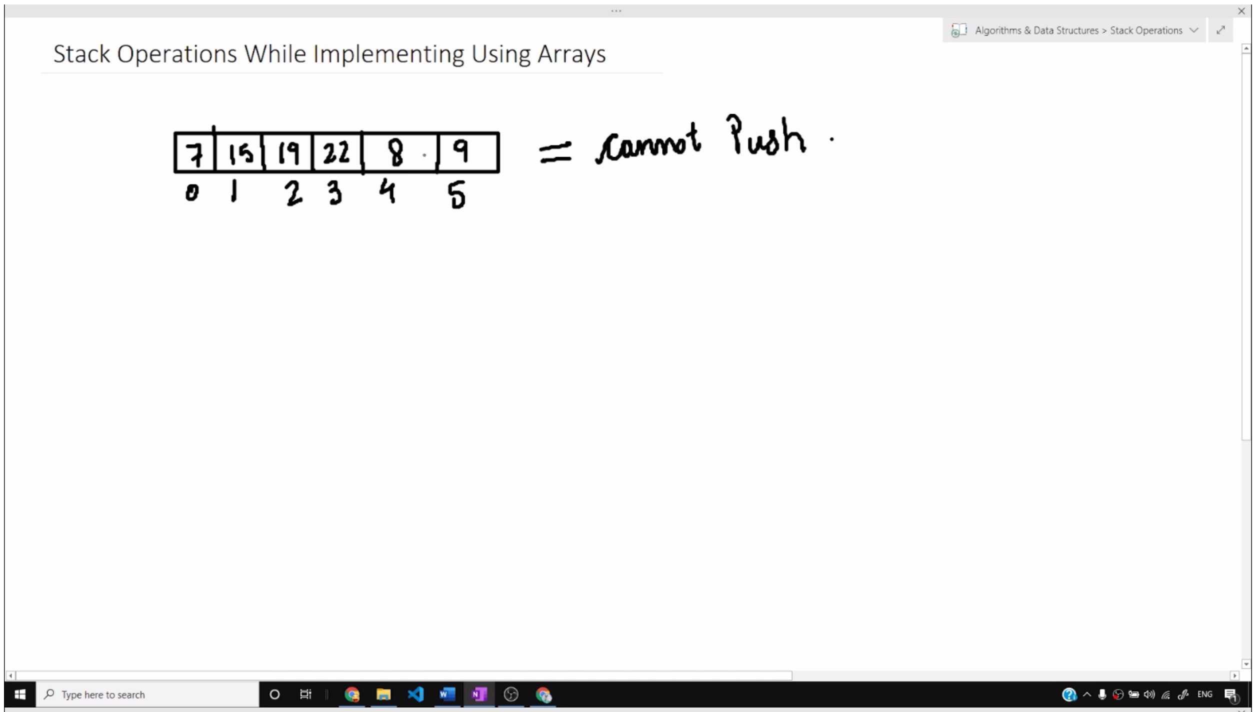
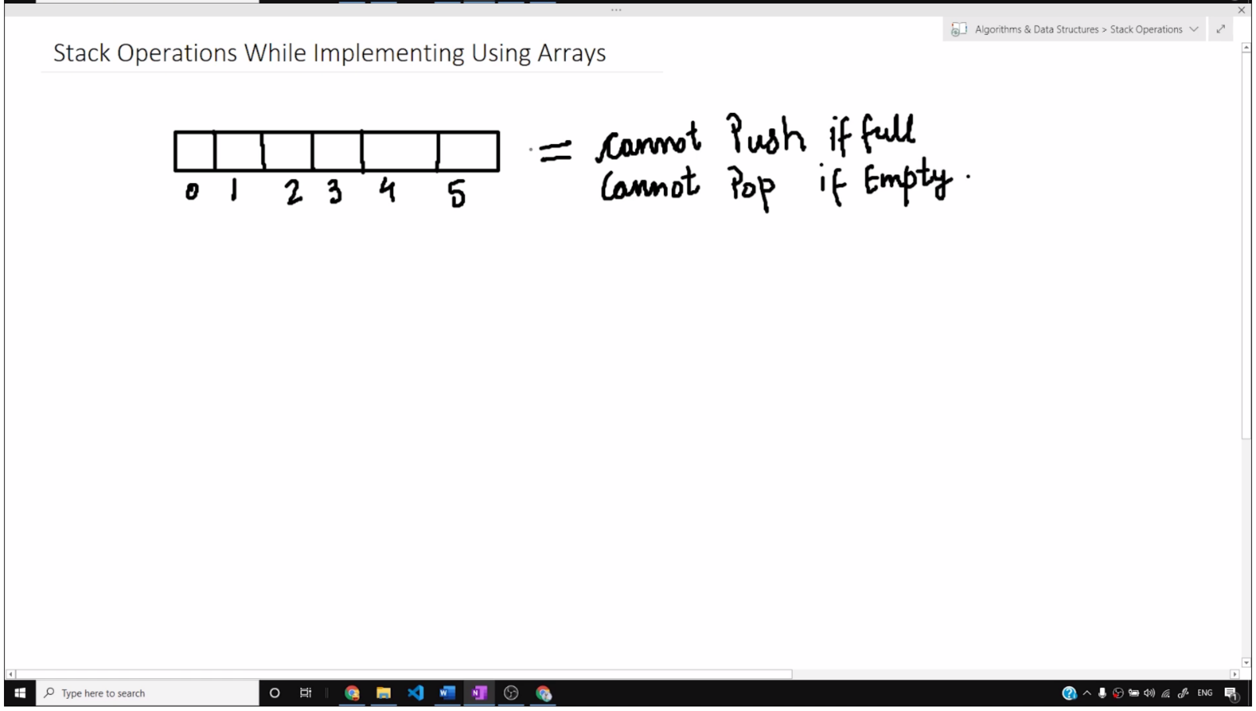


Stack Operations While Implementing Using Arrays

7	15	19	22	8	9
0	1	2	3	4	5

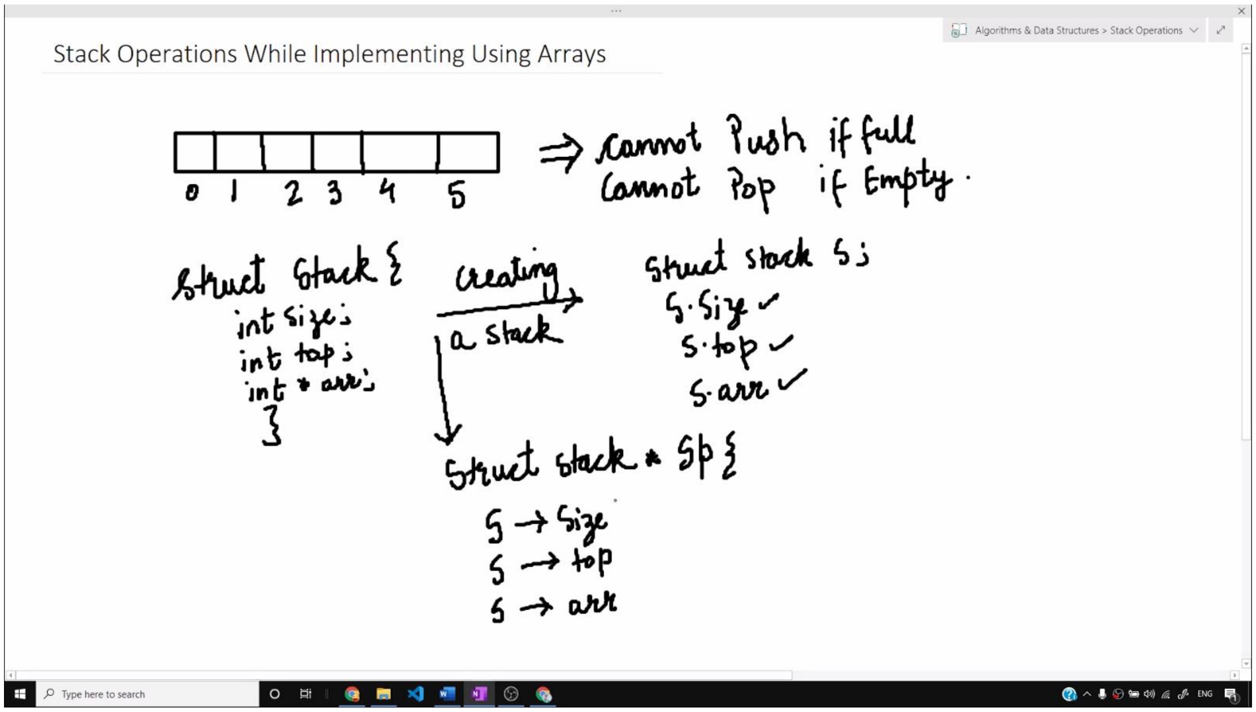
= cannot Push .



Stack Operations While Implementing Using Arrays

0	1	2	3	4	5

= cannot Push if full
Cannot Pop if Empty .



Stack Operations While Implementing Using Arrays

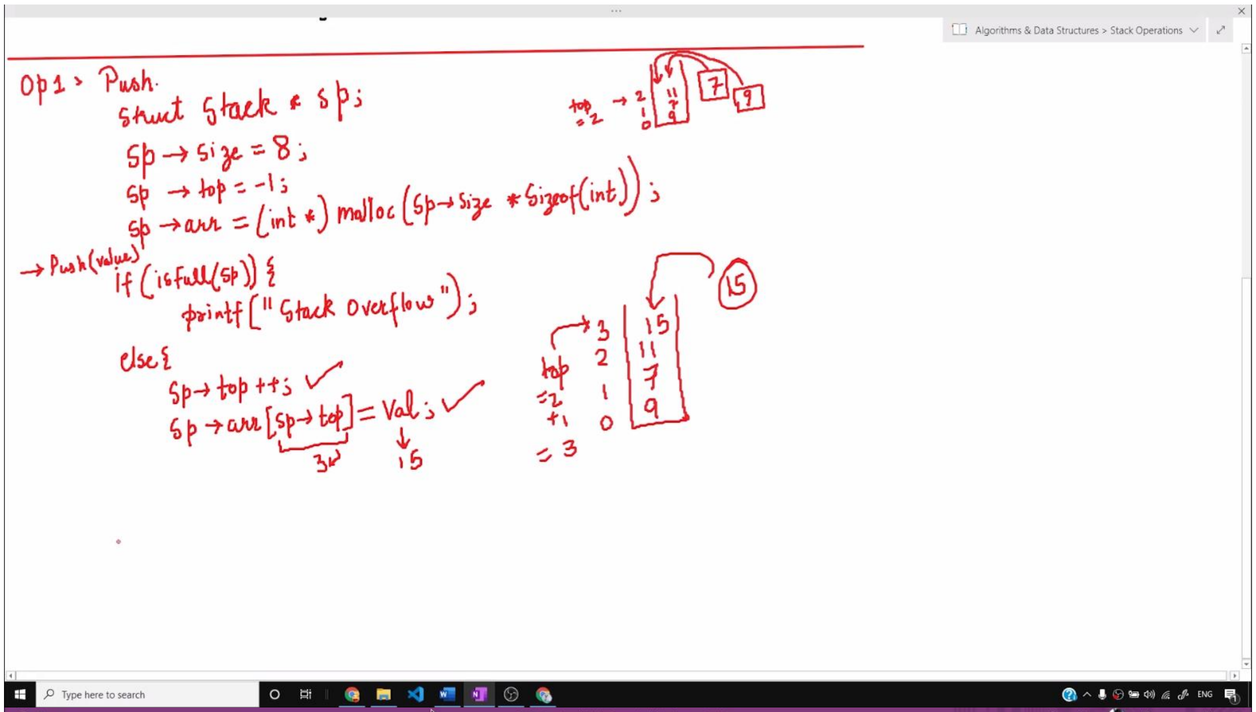
```
struct Stack {
    int size;
    int top;
    int * arr;
}
```

```
struct stack S;
S.size ✓
S.top ✓
S.arr ✓
```

```
struct stack * Sp {
    S → size
    S → top
    S → arr
```

Arrow operator → first deference and then put dot

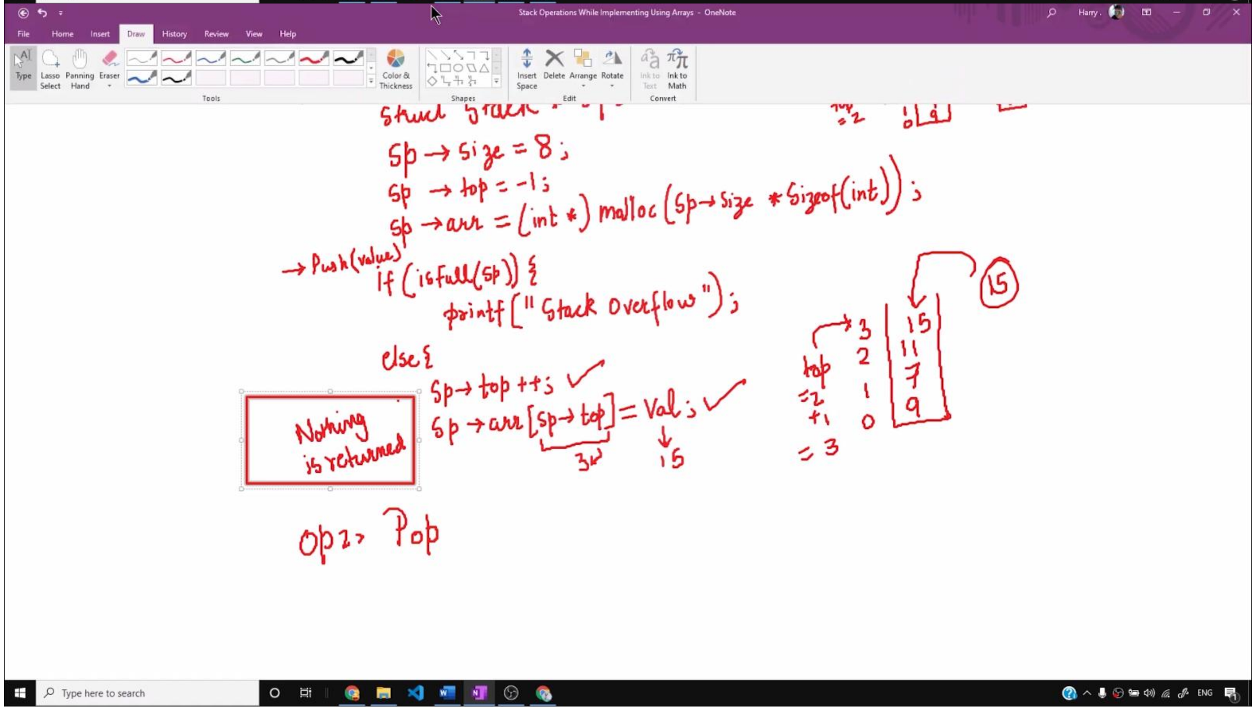
s→size is equivalent to (*s).size



Op1 > Push.

```
struct stack * sp;
  sp → size = 8;
  sp → top = -1;
  sp → arr = (int *) malloc(sp→size * sizeof(int));
→ Push(value)
  if (isFull(sp)) {
      printf("Stack Overflow");
  else {
      sp→top++; ✓
      sp→arr[sp→top] = Val; ✓
```

$\underbrace{sp \to top}_{3}$ $\quad Val \downarrow 15$

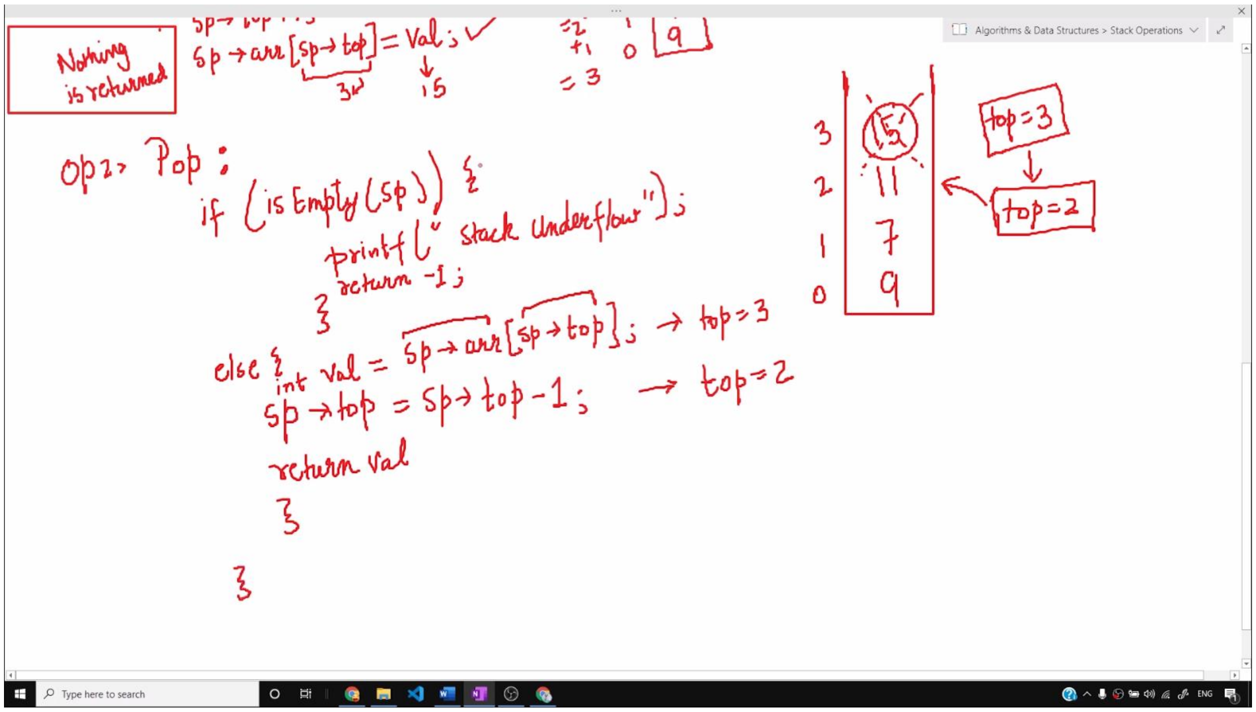


```
struct stack * sp;
  sp → size = 8;
  sp → top = -1;
  sp → arr = (int *) malloc(sp→size * sizeof(int));
→ Push(value)
  if (isFull(sp)) {
      printf("Stack Overflow");
  else {
      sp→top++; ✓
      sp→arr[sp→top] = Val; ✓
```

$\underbrace{sp \to top}_{3}$ $\quad Val \downarrow 15$

$\boxed{\text{Nothing is returned}}$

Op2 > Pop



$\boxed{\text{Nothing is returned}}$

$sp \rightarrow arr[\underbrace{sp \rightarrow top}_{3rd}] = \underset{15}{\underset{\downarrow}{Val}}; ✓$

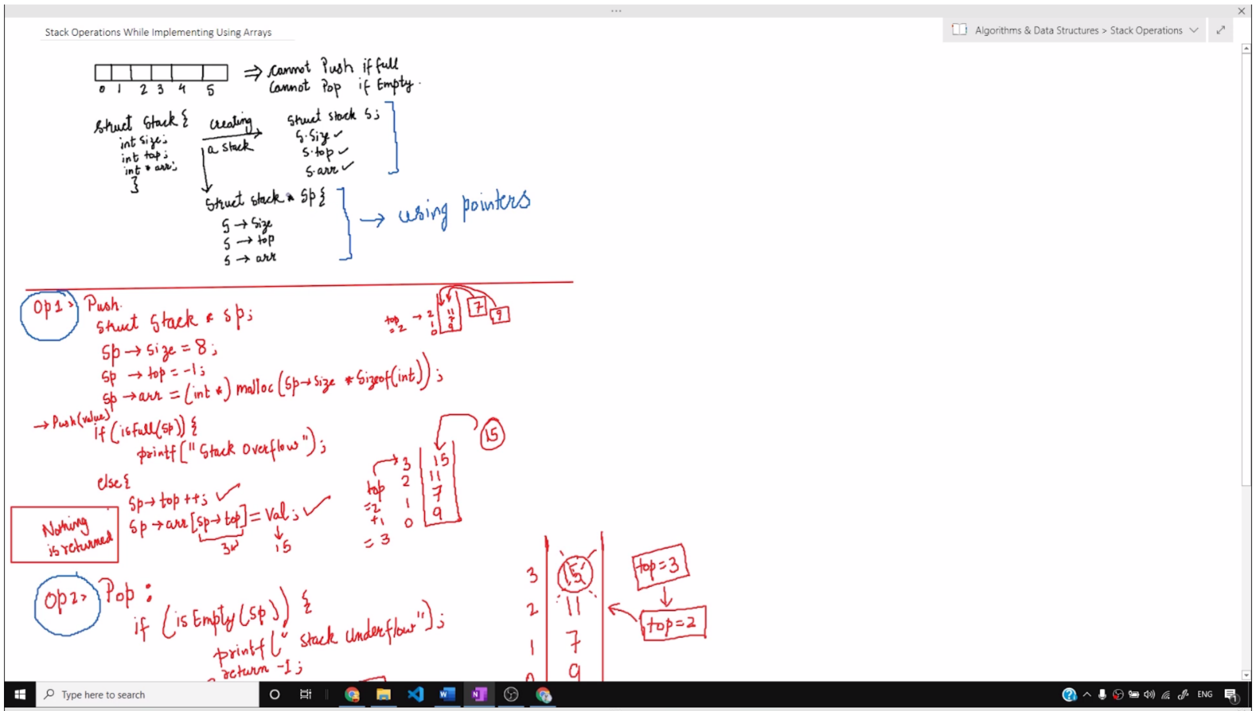
=2
+1
=3

Op2> Pop :

```
if (isEmpty(sp)) {
        printf(" stack underflow");
        return -1;
    }
else {
    int val = sp→arr[sp→top];   → top=3
    sp→top = sp→top-1;          → top=2
    return val
    }
}
```

3	15
2	11
1	7
0	9

$\boxed{top=3} \rightarrow \boxed{top=2}$



Stack Operations While Implementing Using Arrays

⇒ Cannot Push if full
Cannot Pop if Empty.

```
struct Stack {
    int size;
    int top;
    int * arr;
}
```

creating a stack →

```
struct stack S;
    S.size ✓
    S.top ✓
    S.arr ✓
```

```
struct stack * Sp {
    S → size
    S → top
    S → arr
```

→ using pointers

Op1 > Push.

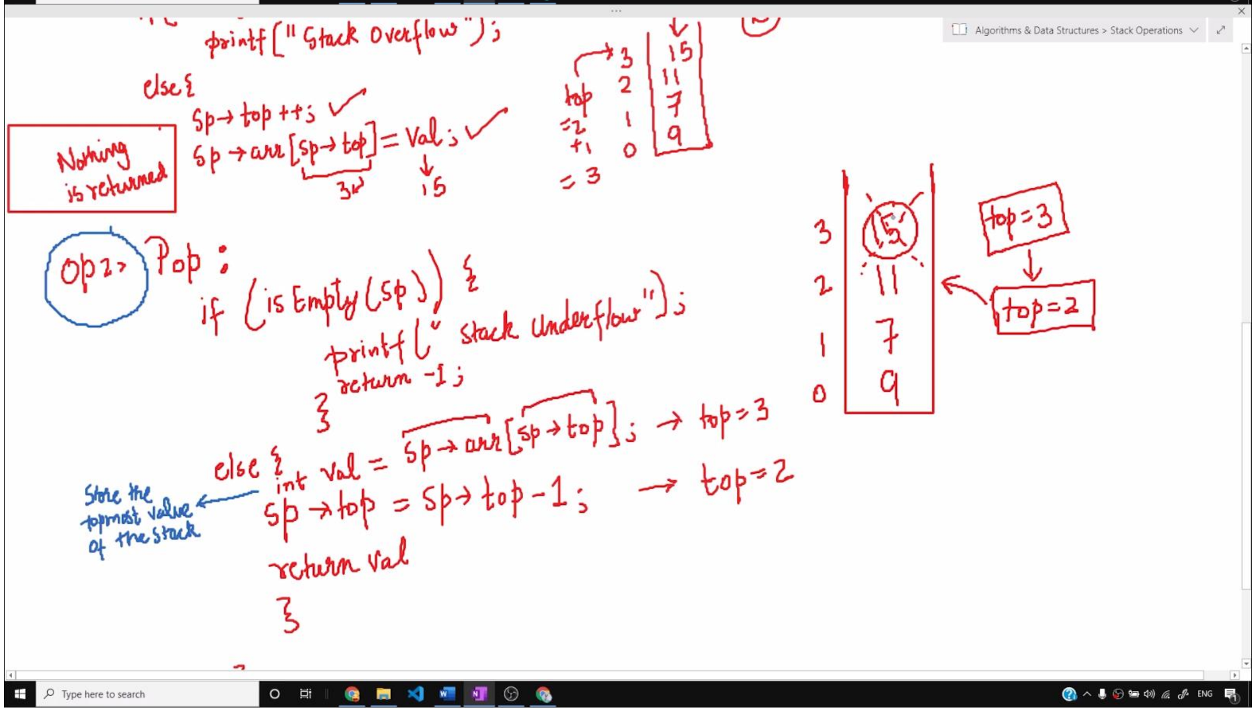
```
struct Stack * Sp;
    Sp → size = 8;
    Sp → top = -1;
    Sp → arr = (int *) malloc (Sp → size * sizeof(int));
→ Push(value)
    if (isFull(Sp)) {
        printf("Stack Overflow");
    else {
        Sp → top ++; ✓
        Sp → arr[Sp → top] = Val; ✓
```

$\underbrace{Sp \rightarrow top}_{3rd}$, Val ↓ 15

Nothing is returned

Op2 > Pop :

```
if (isEmpty(Sp)) {
    printf(" stack underflow");
    return -1;
```



```
        printf("Stack Overflow");
    else {
        Sp → top ++; ✓
        Sp → arr[Sp → top] = Val; ✓
```

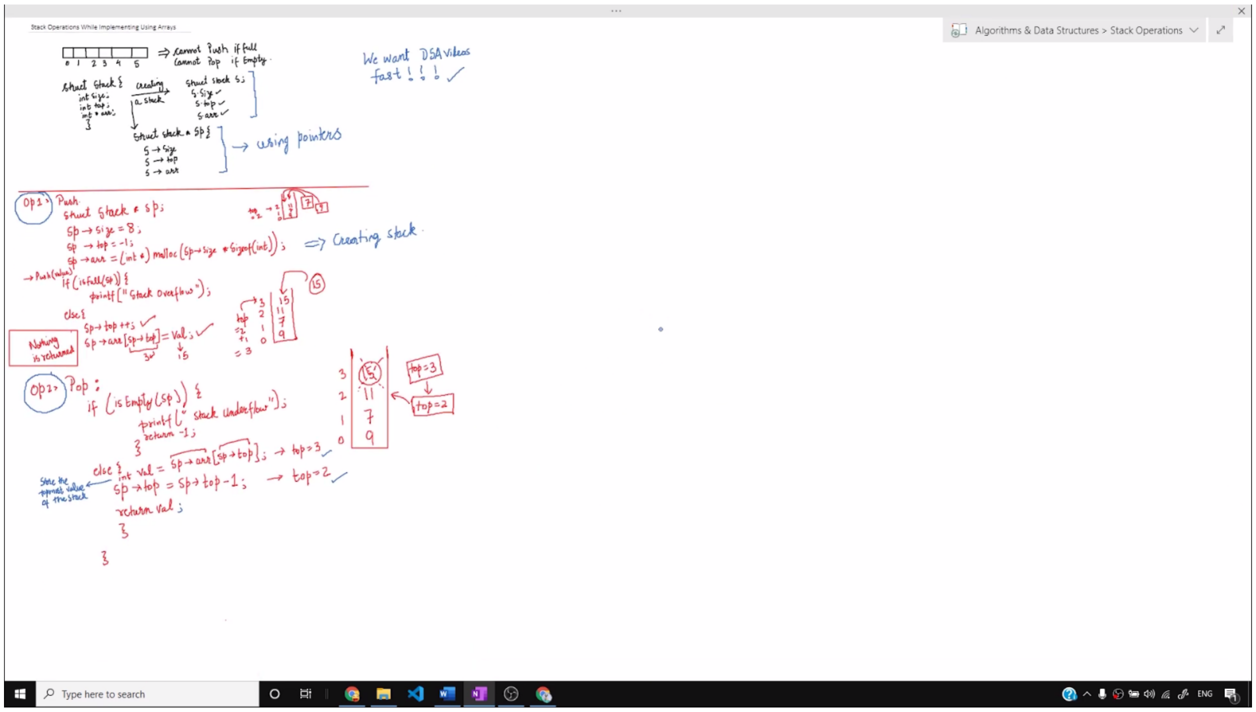
Nothing is returned

Op2 > Pop :

```
if (isEmpty(Sp)) {
    printf(" stack underflow");
    return -1;
}
```

```
else {
    int val = Sp → arr[Sp → top]; → top = 3
    Sp → top = Sp → top - 1; → top = 2
    return val
}
```

Store the topmost value of the stack



Stack Operations While Implementing Using Arrays

```
0 1 2 3 4 5   ⇒ cannot Push if full
                Cannot Pop if Empty.

struct Stack {   --creating a stack-->   struct stack s;
   int size;                               s.size ✓
   int top;                                s.top ✓
   int * arr;                              s.arr ✓
}
        ↓
        struct stack * sp {
           s → size
           s → top          → using pointers
           s → arr
```

We want DSA videos fast ! ! ! ✓

```
Op1 > Push.
   struct Stack * sp;
   sp → size = 8;
   sp → top = -1;
   sp → arr = (int *) malloc (sp → size * sizeof(int));   ⇒ Creating stack.

→ Push(value)
   if (isFull(sp)) {
        printf ("Stack Overflow");

   else {
        sp → top ++; ✓
        sp → arr [sp → top] = Val; ✓
                  3          ↓
                             15
Nothing is returned

   top = 2 + 1 = 3

Op2 > Pop :
   if (isEmpty(sp)) {
        printf (" stack underflow");
        return -1;
   }
   else {
        int val = sp → arr [sp → top];   → top = 3 ✓
        sp → top = sp → top - 1;         → top = 2 ✓
        return val;
   }

}
```

Store the topmost value of the stack

top = 3 → top = 2