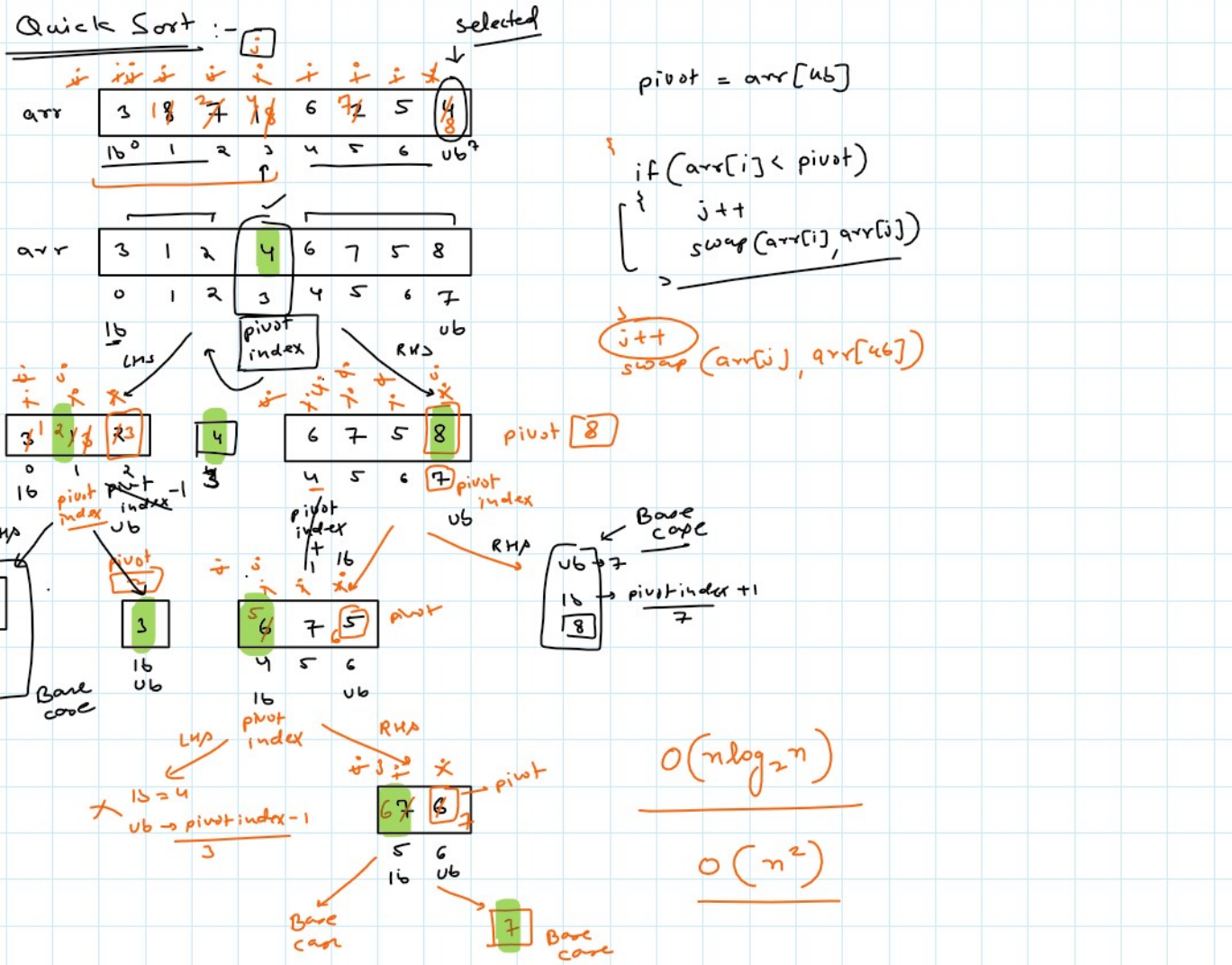
DAY 15

10 May 2024 01:01 PM

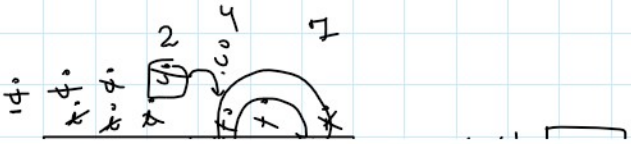
Radix Sort :-

```
#include<iostream>
using namespace std;
void countingSort(int arr[], int n,int
unit)
{
    int count[10]={0};
    int b[n+1];
    //count elements
    for(int i=0;i<n;i++)
    {
        int index=arr[i]/unit%10;
        count[index]++;
    }

    //left sum of count array
    for(int i=1;i<10;i++)
    {
        count[i]+= count[i-1];
    }

    //new index
    for(int i=n-1;i>=0;i--)
    {
        int index=arr[i]/unit%10;
        b[count[index]]=arr[i];
        count[arr[i]/unit%10]--;
    }
    //copy from b to arr
    for(int i=0;i<n;i++)
    {
        arr[i]=b[i+1];
    }
}
void radixSort(int arr[], int n)
{
    int max=0;
    for(int i=0;i<n;i++)
    {
        if(arr[i]>max)
            max=arr[i];
    }
    int unit;
    for(unit=1;max>unit;unit=unit*10)
    {
        countingSort(arr,n,unit);
```

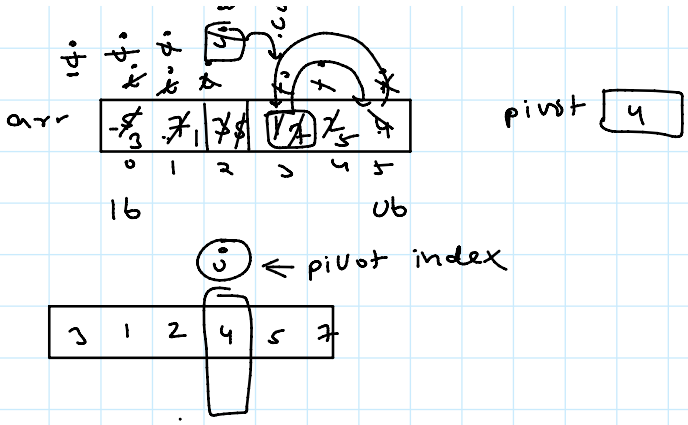
```
        }
    }
    int main()
    {
        int arr[]={110,75,25,189,537,258};
        int n=sizeof(arr)/sizeof(int);
        radixSort(arr,n);
        for(int i:arr)
        {
            cout<<i<<" ";
        }
        return 0;
    }
```



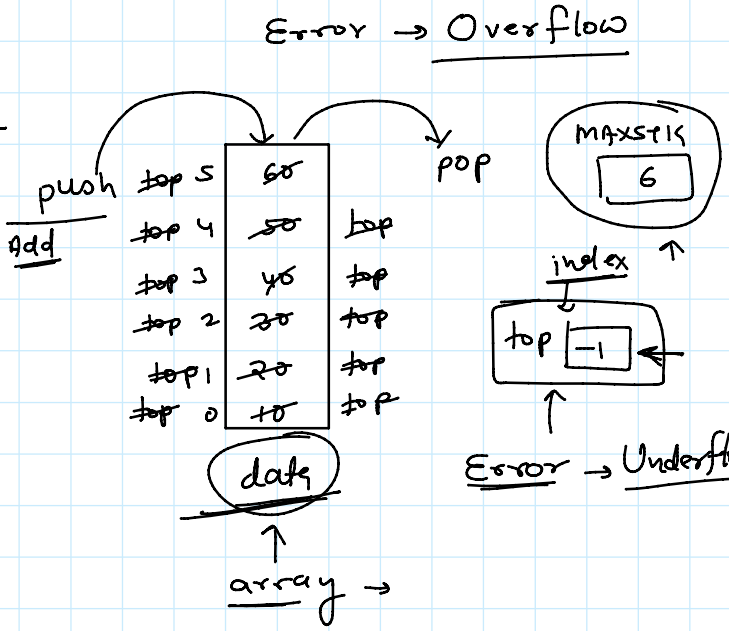

```
#include<iostream>
using namespace std;
int partition(int arr[], int lb, int ub)
{
    int pivot=arr[ub];
    int i,j;
```



```
{
    int pivot=arr[ub];
    int i,j;
    for(i=lb,j=lb-1;i<ub;i++)
    {
        if(arr[i]<pivot)
        {
            j++;
            swap(arr[i],arr[j]);
        }
    }
    j++;
    swap(arr[j],arr[ub]);
    return j;   //pivot_index
}
void QuickSort(int arr[], int lb, int ub)
{
    if(ub<=lb)
        return;
    int pivot_index=partition(arr,lb,ub);
    QuickSort(arr,lb,pivot_index-1);    //LHS
    QuickSort(arr,pivot_index+1,ub);    //RHS
}
int main()
{
    int arr[]={110,75,25,189,537,258};
    int n=sizeof(arr)/sizeof(int);
    QuickSort(arr,0,n-1);
    for(int i:arr)
        cout<<i<<" ";
    return 0;
}
```



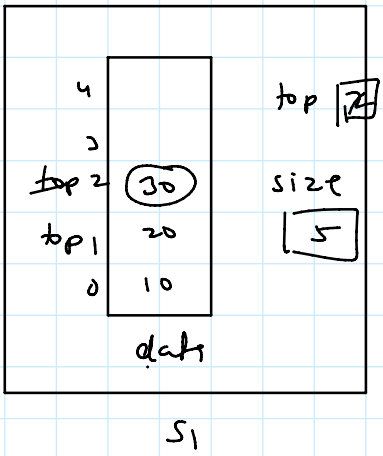
Stack :- Linear data structure :-



Traverse

```
# define size 10
class Stack
{
        int data[size];

        int top;
        public:
            Stack(int n)
            {
                size=n;
```

```
{
    size = n;
    top = -1;
}
void push(int no)
{
    if(top == size-1)
    {
        cout << "Overflow";
        return;
    }
    top++;
    data[top] = no;
}
void pop()
{
    if(top == -1)
    {
        cout << "Under flow";
        return;
    }
    top--;
}
```




```
};
```

```
#include<iostream>
using namespace std;
#define size 5
class Stack
{
    int data[size];
    int top;
    public:
        Stack()
        {
            top=-1;
        }
        void push(int no)
        {
            if(top==size-1)
            {
                cerr<<"Overflow\n";
                return;
            }
            top++;
```

```
            data[top]=no;
        }
        void pop()
        {
            if(top == -1)
            {
                cerr<<"Underflow\n";
                return;
            }
            top--;
        }
        int top_element()
        {
            if(top==-1)
                return 0;
            return data[top];
        }
};
int main()
{
    Stack s1;
    s1.push(10);
    s1.push(20);
    s1.push(30);
    s1.push(40);
    s1.push(50);
    s1.push(60);
    cout<<s1.top_element()<<endl;
    s1.pop();
    cout<<s1.top_element()<<endl;
    s1.push(60);
    cout<<s1.top_element()<<endl;
    while(s1.top_element()!=0)
    {
        s1.pop();
    }
    s1.pop();
}
```

Application of stack :-

Polish notation

infix to post fix ← (Rev polish)

infix to prefix ← (polish)

	precedency	Associativity
①	^	R to L
②	* /	L to R
③	+ −	L to R

infix → a + b

a b +

infix → a+b

post → a b +

prefix → + a b

② * /

③ + −

L to R

infix → a + b * c / d − e to postfix

a + bc* / d − e

a + bc*d / − e

abc*d/+ − e

abc*d/+e− ← postfix

infix → a + b * c / d − e to Prefix

a + *bc / d − e

a + /*bcd − e

+a/*bcd − e

−+a/*bcde ← prefix

infix to postfix using stack

infix (L to R) a+b−c/d+e*f

postfix → ab+cd/−ef*+

infix a+b-c/d+e*f

L to R

infix	stack	postfix
a		a
+	+	a
b	+	ab
-	-	ab+
c	-	ab+c
/	-, /	ab+c
d	-, /	ab+cd
+	+	ab+cd/-
e	+	ab+cd/-e
*	+, *	ab+cd/-e
f	+, *	ab+cd/-ef*+

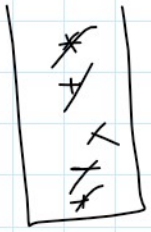
postfix → ab+cd/-ef*+

operand → postfix

operator → stack

if

Top < cur



stack

postfix

infix to prefix

infix → a+b-c/d+e*f

R to L

prefix → +-+ab/cd*ef

infix	stack	prefix
f		f
*	*	f
e	*	fe
+	+	fe*
d	+	fe*d
/	+, /	fe*d
c	+, /	fe*dc
-	+, -	fe*dc/
b	+, -	fe*dc/b
+	+, -, +	fe*dc/b
a	+, -, +	fe*dc/ba+-+

operand → prefix

operator → stack

if

<=

Reverse

+-+ab/cd*ef

prefix

infix → (a+b) * c

postfix → a b + c *

infix → a + b * c

postfix → a b c * +

infix a+(b-c)/d*e → postfix a b c - d / e * +

L to R

infix	stack	postfix
a		a
+	+	a
'c'	+, 'c'	a
b	+, 'c'	ab
-	+, 'c', -	ab
c	+, 'c', -	abc
')'	+	abc -
/	+, /	abc -
d	+, /	abc - d
*	+, *	abc - d /
e	+, *	abc - d / e * +

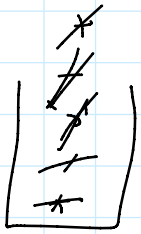
postfix

infix → a+(b-c)/d*e . to prefix using stack →

+ a * / - b c d e

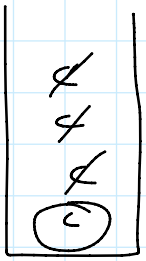
<=

infix	stack	pre
e		e
*	*	e
d	*	ed
/	* /	ed
')'	* / ')'	ed
c	* / ')'	edc
-	* / ')' -	edc
b	* / ')' -	edcb
'('	* /	edcb -
+	+	edcb - / *
a	+	edcb - / * a +

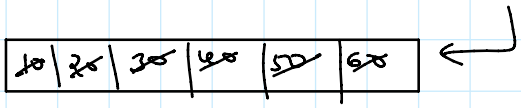


parentheses matching :—

((a+b) - ((d * e) + f)



Queue :→ Linear DS :→ FIFO

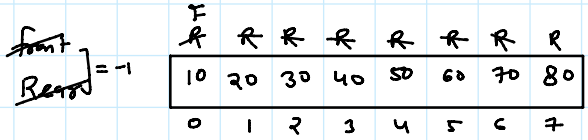


operations
→ Add Queue
→ del Queue

① Linear queue
② Circular queue
③ priority queue

① linear queue :—

MAXQ 8

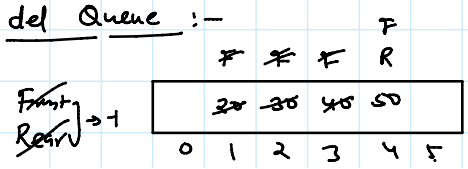


Add Queue

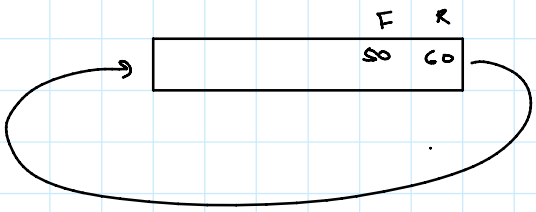
```
if (Rear == MAXQ-1)
    overflow
    Return
if (Front == -1)
    Front = Rear = 0;
else
    Rear ++;
Queue[Rear] = no;
```

del Queue :—

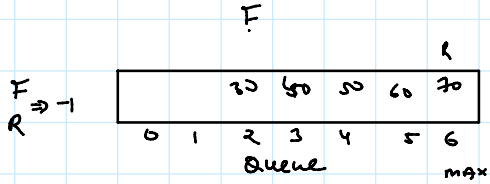
MAXQ 6



```
if (Front == -1)
{       Underflow
        Return
if( Front == Rear)
        front = Rear = -1
else
        front ++
```



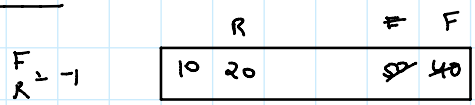
② Circular Queue :—



MAXQ 7

```
if (Rear == front -1 || Rear == MAXQ-1 && front ==0)
        Overflow, Return.
if (Front == -1)
        front = Rear = 0
else if( Rear == MAXQ-1)
        Rear = 0;
else
        Rear ++
```

del Queue :-



MAXQ 6

```
if ( Front == -1 )
        under flow , Return.
if ( front == Rear)
        front = Rear = -1 ;
else if (front == MAXQ-1)
        front = 0;
```

else

front ++