Data Structures – CST 201
Module - 2

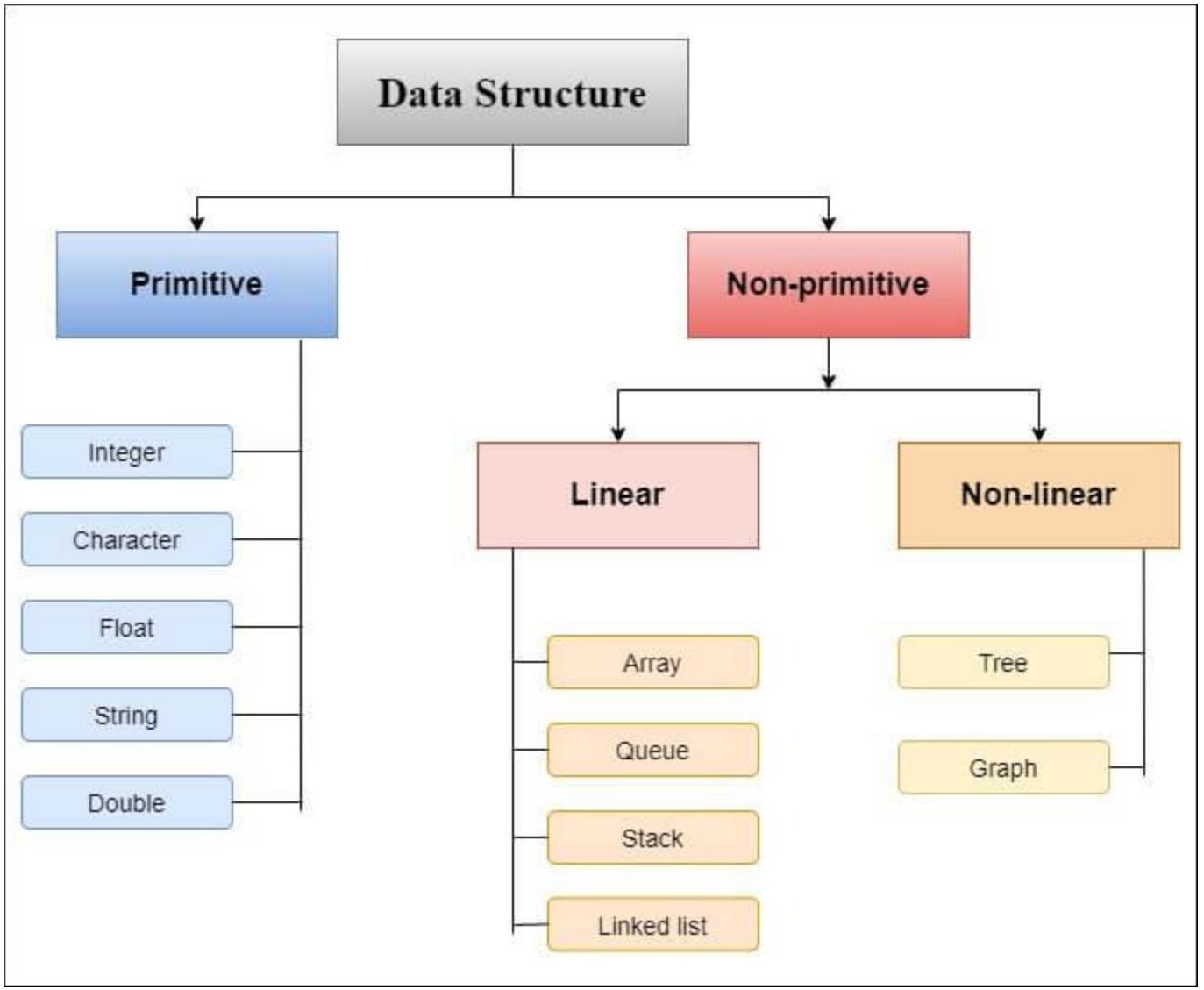
Syllabus

- Polynomial representation using Arrays
- Sparse matrix
- Stacks
  - Evaluation of Expressions
- Queues
  - Circular Queues
  - Priority Queues
  - Double Ended Queues,
- Linear Search
- Binary Search

DATA STRUCTURES

- Data may be organized in many different ways
- The logical or mathematical model of a particular organization of data is called data structure
- A data structure is a particular way of organizing data in a computer so that it can be used efficiently
- It is also called building block of a program.

TYPE DATA STRUCTURES



TYPE DATA STRUCTURES

- **Primitive data structure(Simple Data structure)**
  - Simple data structure can be constructed with the help of primitive data types
  - It can hold a single value
- **Non-primitive data structure(Compound data structure)**
  - It can be constructed with the help of any one of the primitive data structure and it is having a specific functionality. It is divided into two.
  - **Linear data structure**
    - Elements are arranged in a sequential manner
  - **Non-linear data structure**
    - Elements are arranged in a random manner

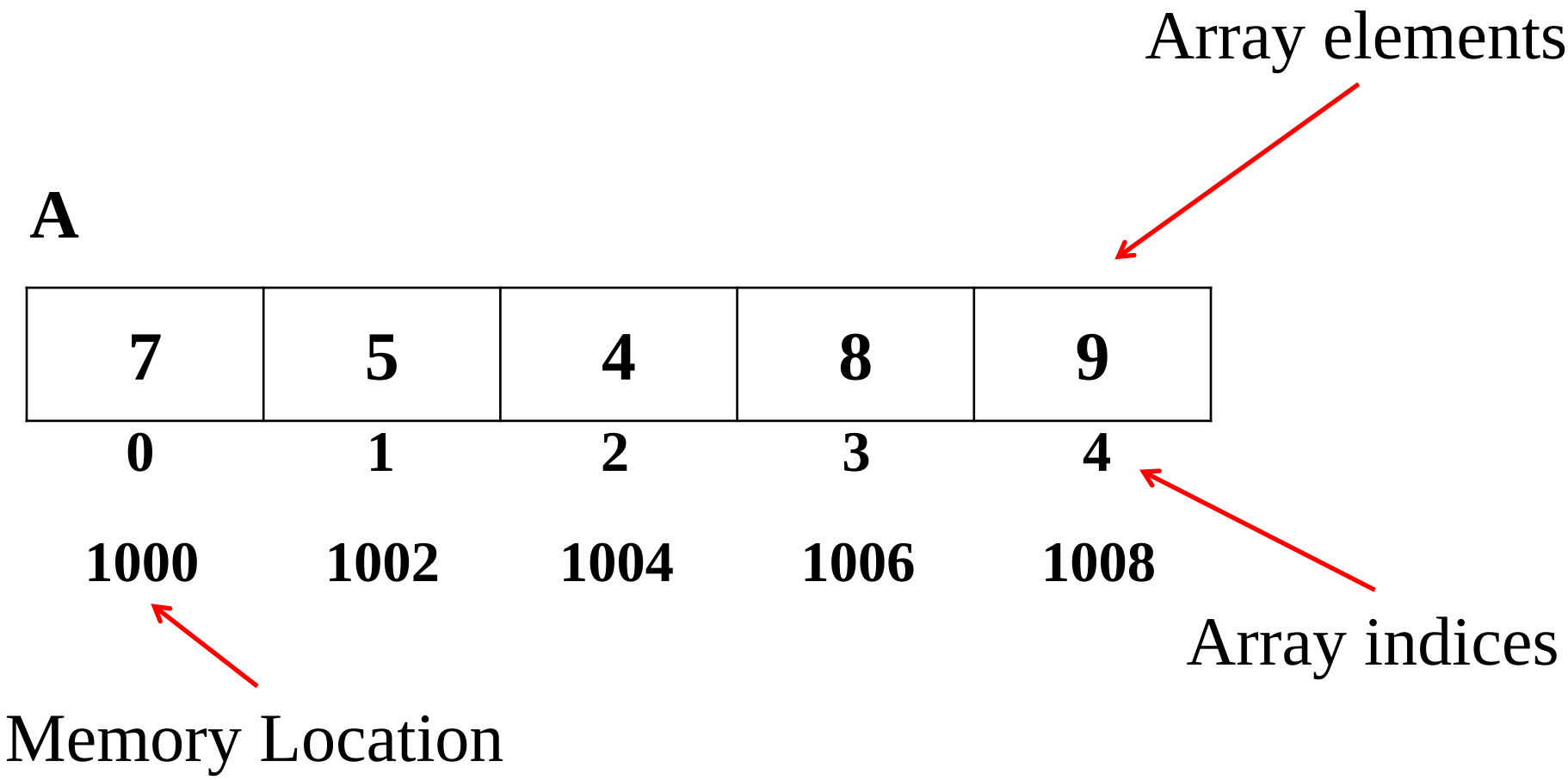
OPERATIONS ON DATA STRUCTURES

- Add an element
- Delete an element
- Traverse / Display
- Sort the list of elements
- Search for a data element

ARRAY

- It is a Linear data structure
- It is a list of finite no. of homogeneous data elements
  - It is finite, because it contains fixed no. of elements
  - It is homogeneous, because the elements of the array are of same data types (int, float, double, char, etc)
- Array is a collection of similar items stored at contiguous memory locations
- Elements can be accessed randomly using indices of an array

ARRAY



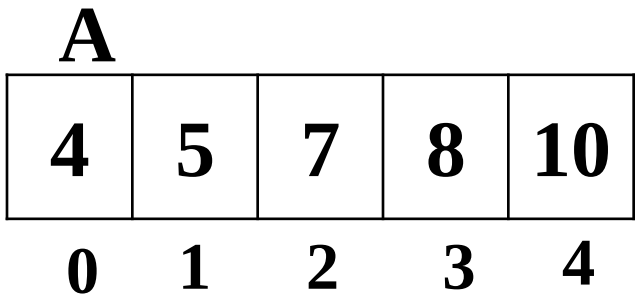
NB: Array index will always starts with 0

ARRAY - TERMINOLOGIES

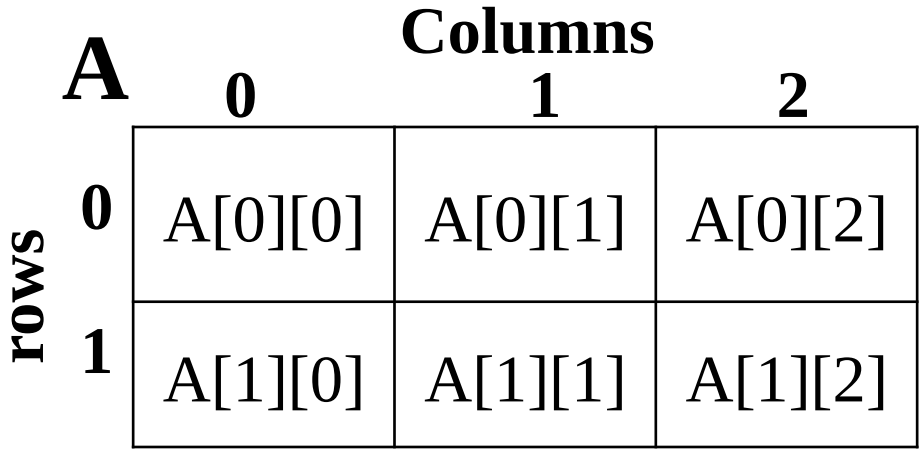
- **Type**: Kind of data type it is meant for.
- **Base**: Address of the memory location where the first element of the array is located
- **Index**: Subscript like A_i or A[i]
- **Range of indices**: Boundaries of an array
  - Upper bound(U) and lower bound(L)
  - In C, the range of indices is from 0
- Index (A[i]) = L + i – 1
- **Size(Length/Dimension)**: The no. of elements in an array.
  - Size(A) = U – L + 1

ARRAY

- Different Types
  - Single Dimensional Array

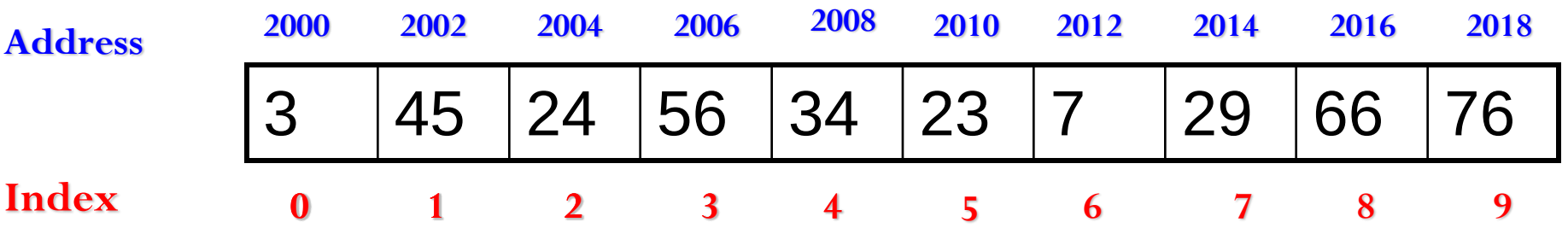


  - Multidimensional Array

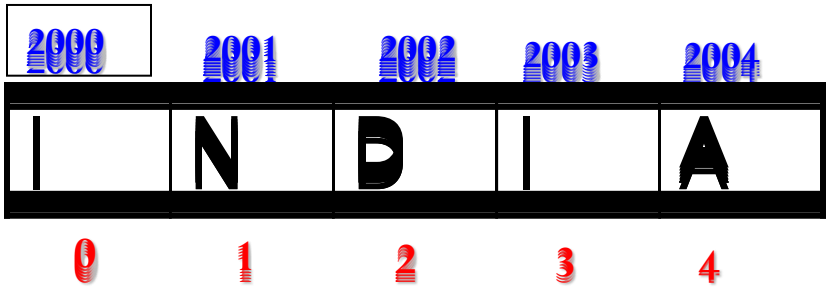


Representation of Array in Memory

int a[10]={3,45,24,56,34,23,7,29,66,76}



char mystring="INDIA"



The basic Operations Performed in an Array

- Traversal
  - Processing each element in the list
- Insertion
  - Adding a new element to the list
- Deletion
  - Removing an element from the list
- Searching
  - Find the location of the element with the given value
- Sorting
  - Arranging elements in some type of order
- Merging
  - Combining two list into a single list

Array Traversal

- In traversing operation of an array, each element of an array is accessed exactly for once for processing.
- This is also called visiting of an array.
- If we need to print all the elements or to count no. of elements or to find largest or smallest element, the traversal should be done.

TRAVERSING

ALGORITHM

1. Start
2. Set K=LB
3. Repeat step 3 and 4 until K<= UB
4. Apply Process to Array
5. Increase Counter K=K+1
6. Stop

Program

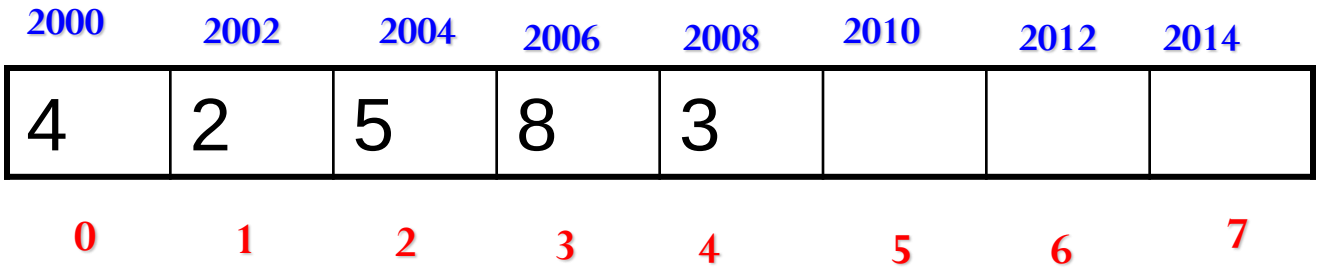
```
#include <stdio.h>
 void main()
{
int LA[] = {2,4,6,8,9};
 int i, n = 5;
printf("The array elements are:\n");
for(i = 0; i < n; i++)
{
 printf("LA[%d] = %d \n", i, LA[i]);
 }
}
```

INSERTION

- Insert operation is to insert one or more data elements into an array.
- Based on the requirement, new element can be added at the beginning, end or any given index of array.
- Insertion at the end of the array can be easily done
- Suppose we need to insert an element at the middle of the array
  - Half of the elements must be moved forward to the new location to accommodate the new element and keep the order of other elements

int a[8]={4,2,5,8,3}

UB=4,LB=0



ALGORITHM

1. Start
2. Let a[] be an array
3. Let J=UB

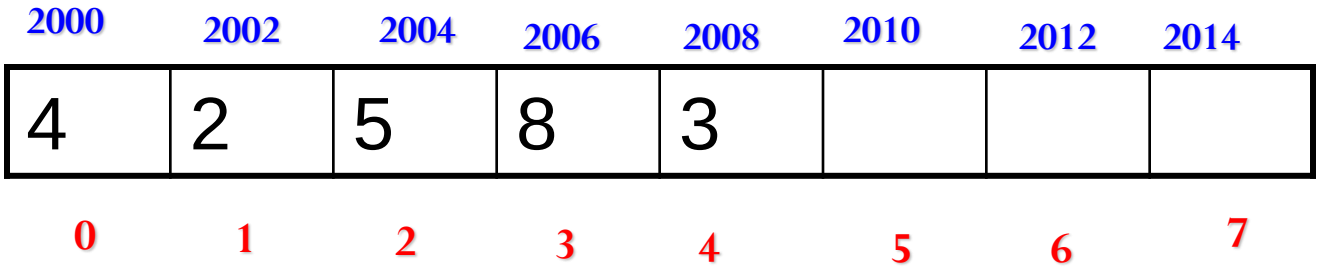
PROGRAM

```
#include <stdio.h>
void main()
{
int a[8] = {4,2,5,8,3};
int J=4;
```

Suppose we need to insert an element 9 to the position 2

Suppose we need to insert an element 9 to the position 2

int a[8]={4,2,5,8,3}



ALGORITHM

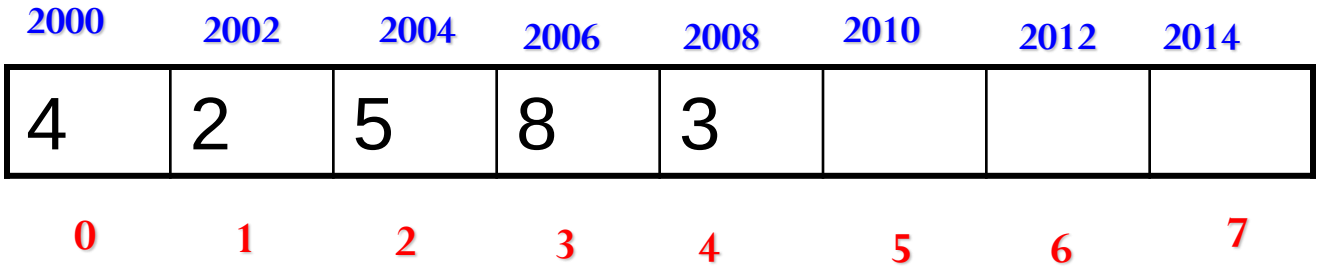
1. Start
2. Let a[] be an array
3. J
4. Let J=UB
5. Let ITEM be the new element and position is K

PROGRAM

```
#include <stdio.h>
void main()
{
int a[8] = {4,2,5,8,3};
int J=4;
Int item=9,K=2;
```

Suppose we need to insert an element 9 to the position 2

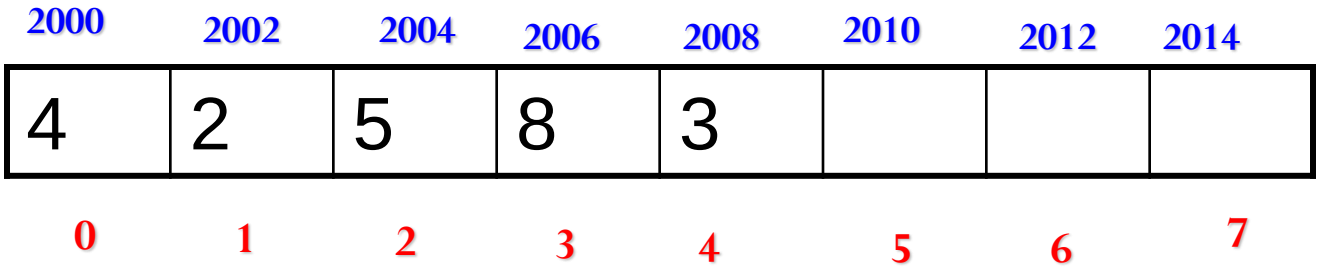
int a[8]={4,2,5,8,3}



Suppose we need to insert an element 9 to the position 2

int a[8]={4,2,5,8,3}

Step 1: Need to create space for position 2



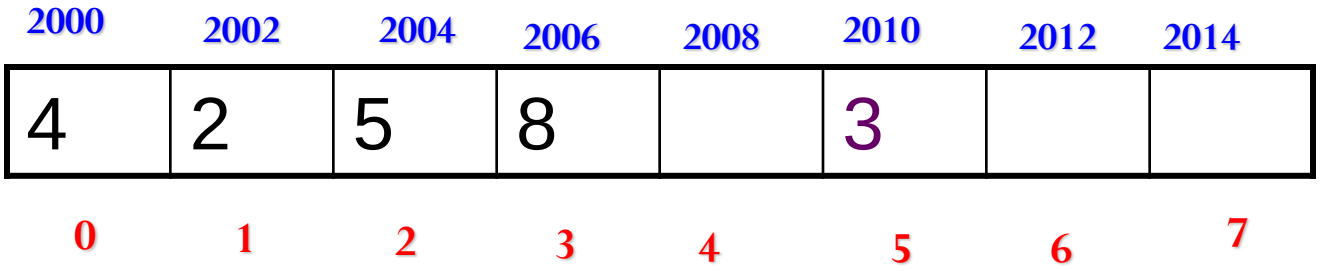
Step 2: First shift last element 3 to position 5

That is a[5]=a[4]

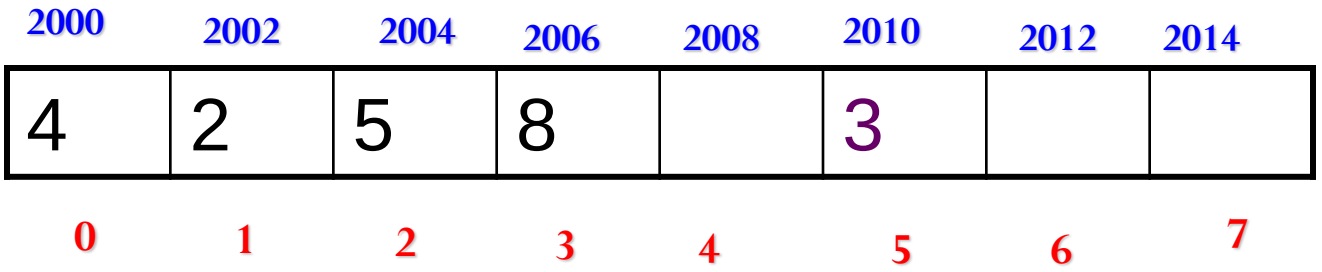
Step 2: First shift last element 3 to position 5

That is a[5]=a[4]

int a[8]={4,2,5,8,3}



int a[8]={4,2,5,8,3}



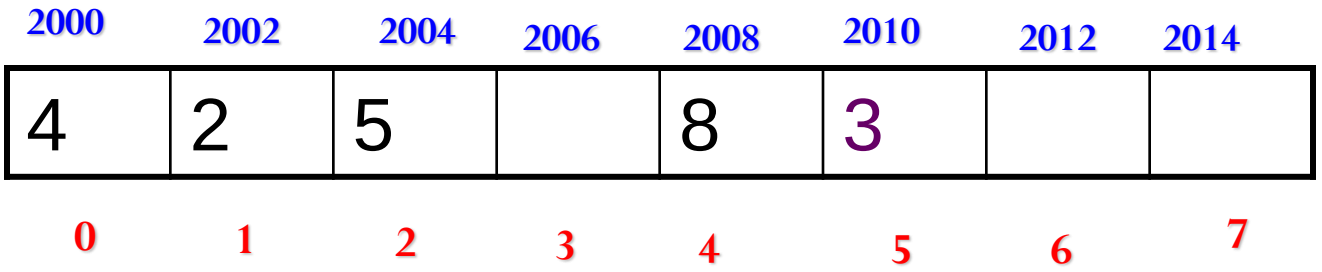
Step 3: Now we have a space to shift 8 to position 4

That is a[4]=a[3]

Step 3: Now we have a space to shift 8 to position 4

That is a[4]=a[3]

int a[8]={4,2,5,8,3}



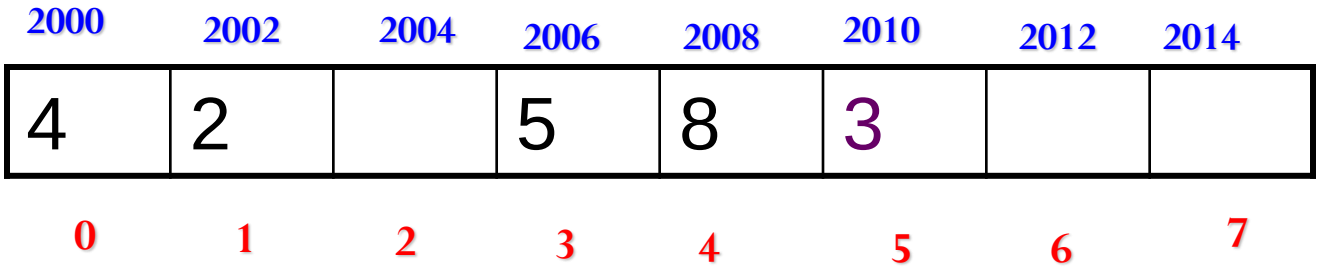
Step 4: Now we have a space to shift 5 to position 3

That is a[3]=a[2]

Step 4: Now we have a space to shift 5 to position 3

That is a[3]=a[2]

int a[8]={4,2,5,8,3}



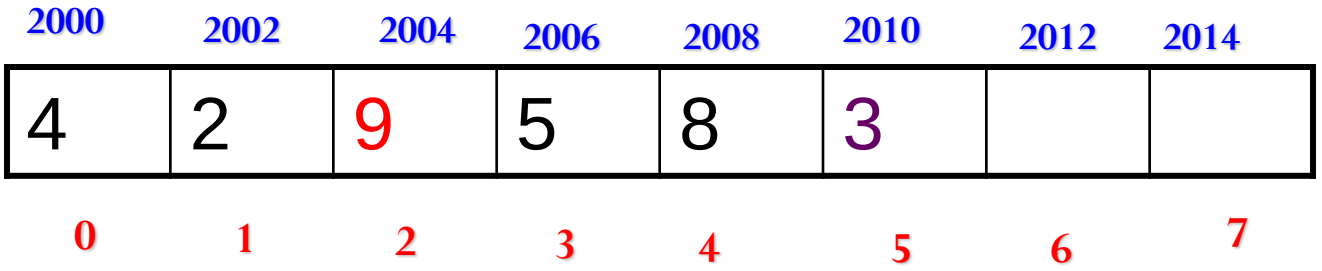
Step 4: Now position 2 is free

So that we can insert item to position 2

Step 5: Now position 2 is free

So that we can insert item to position 2

int a[8]={4,2,5,8,3}



4	2	9	5	8	3		

ALGORITHM

1. Start
2. Let a[] be an array
3. Let J=UB
4. Let ITEM be the new element and position is K
5. Repeat step 6 & 7 while J>=K
6. Set a[J+1]=a[J]
7. Set J=J-1
8. Set a[K]=ITEM
9. Set UB=UB+1
10. Stop

PROGRAM

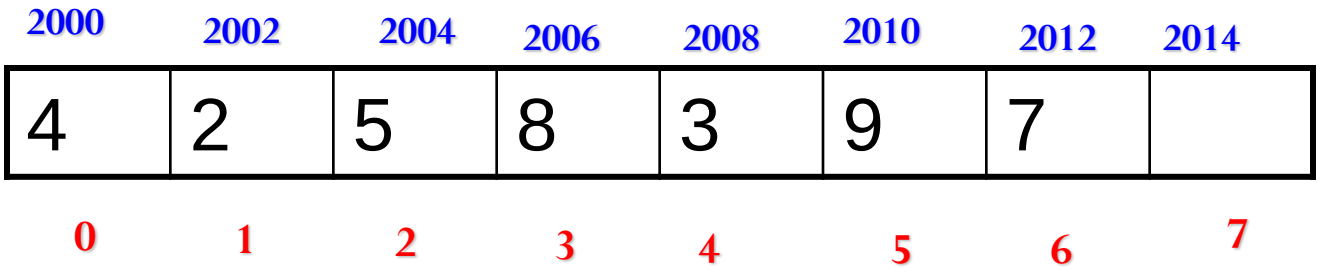
```
#include <stdio.h>
void main()
{
int a[8] = {4,2,5,8,3},n=4;
int j=n, item=9,k=2,i;
printf("Orginal array is");
for (i=0;i<=n;i++)
      printf("%d  ",a[i]);
while(j>=k)
      a[j+1]=a[j];
      j--;
a[k]=item;
n=n+1;
printf("New array is");
for(i=0;i<=n;i++)
      printf("%d",a[i]);
getch();
}
```

DELETION

- Deletion refers to removing an existing element from the array and re-organizing all elements of an array.
- Deleting an element at the end of the array presents no difficulties
- But deleting an element somewhere in the middle of the array would require that each subsequent element be moved one location backward to fill up the location

int a[8]={4,2,5,8,3,9,7}

UB=6,LB=0



ALGORITHM

1. Start
2. Let a[] be an array
3. Let J=UB

PROGRAM

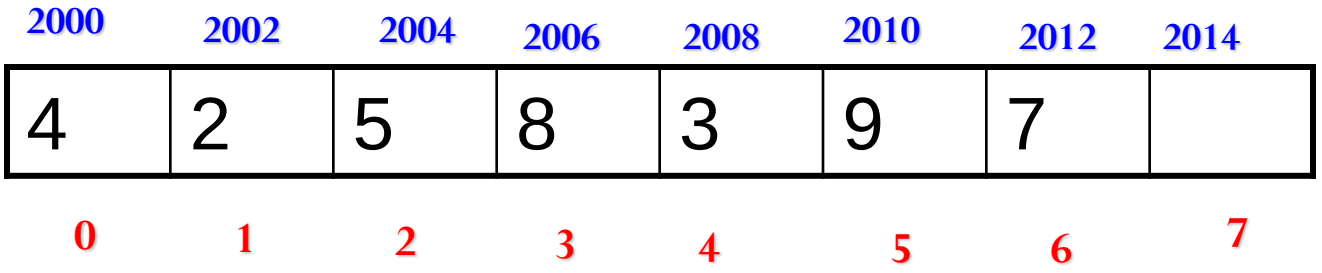
```
#include <stdio.h>
void main()
{
int a[8] = {4,2,5,8,3,9,7};
int J=6;
```

Suppose we need to delete an element 5 from the position 2

Suppose we need to delete an element 5 from the position 2

int a[8]={4,2,5,8,3,9,7}

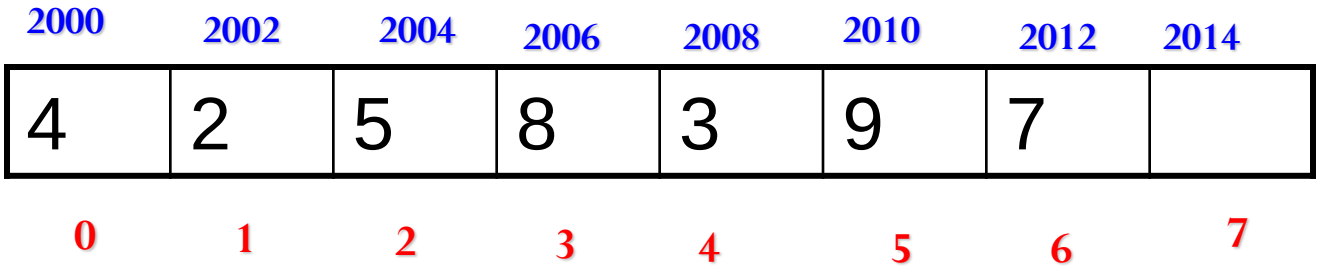
UB=6,LB=0



Actually we are not deleting the item in position 2 , we are just overwriting the elements in position 2 with position 3 and position 3 with position 4 and position 4 with position 5 and position 5 with position 6 and change UB to UB-1

int a[8]={4,2,5,8,3,9,7}

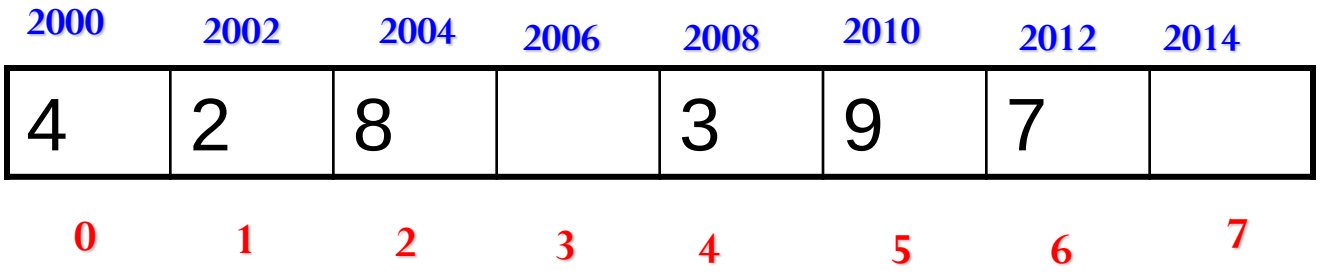
UB=6,LB=0



a[2]=a[3]

int a[8]={4,2,5,8,3,9,7}

UB=6,LB=0

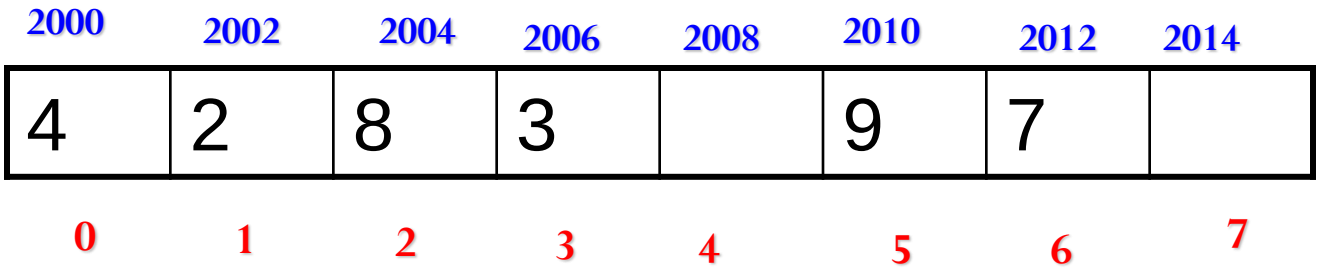


a[2]=a[3]

a[3]=a[4]

int a[8]={4,2,5,8,3,9,7}

UB=6,LB=0



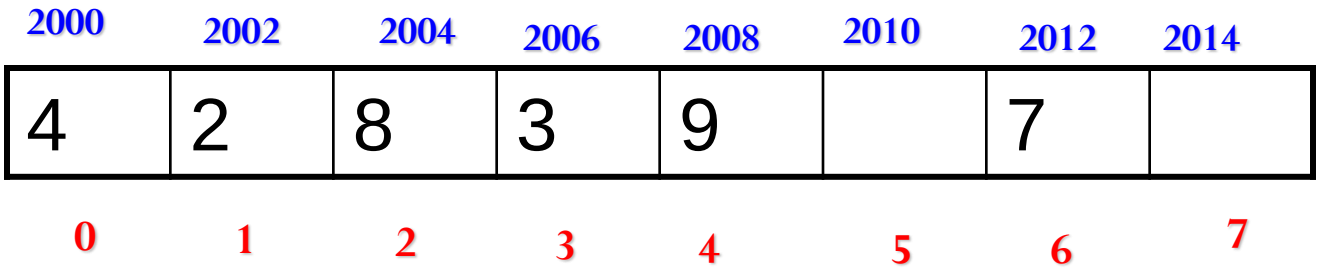
a[2]=a[3]

a[3]=a[4]

a[4]=a[5]

int a[8]={4,2,5,8,3,9,7}

UB=6,LB=0



a[2]=a[3]

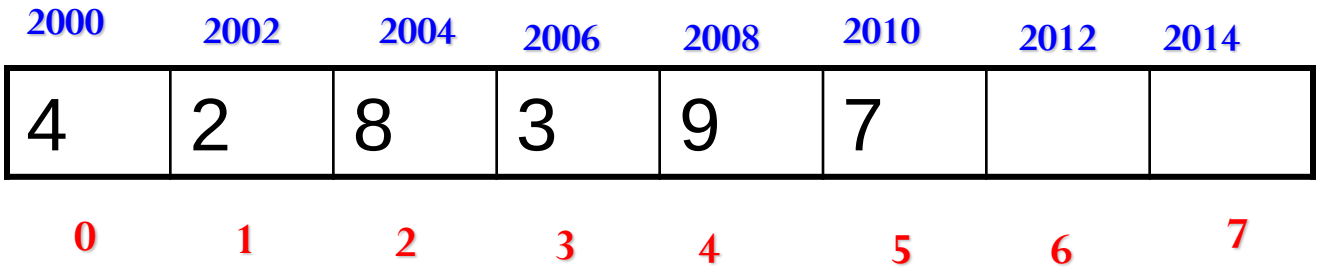
a[3]=a[4]

a[4]=a[5]

a[5]=a[6]

int a[8]={4,2,5,8,3,9,7}

UB=6,LB=0



a[2]=a[3]

a[3]=a[4]

a[4]=a[5]

a[5]=a[6]

UB=UB-1

ALGORITHM

1. Start
2. Let a[] be an array
3. Let k be the position of the element to be deleted
4. Set item=a[k]
5. Repeat step 6 & 7 for J=k to UB-1
6. Set a[J]=a[J+1]
7. Set J=J+1
8. Set UB=UB-1
9. Stop

PROGRAM

```
#include <stdio.h>
void main()
{
int a[8] = {4,2,5,8,3,9,7},n=6;
int j, item,k=2,i;
Item=a[k];
Printf("Orginal array is");
For (i=0;i<=n;i++)
      printf("%d  ",a[i]);
For(j=k;j<n;j++)
      a[j]=a[j+1];
n=n-1;
Printf("New array is");
For(i=0;i<=n;i++)
      printf("%d",a[i]);
getch();
}
```

Linear Search

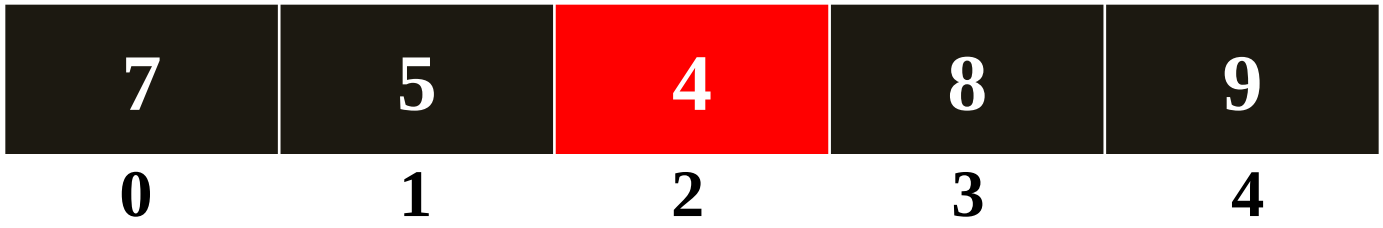
Linear Search

Search Data=8

Linear Search

Search Data=8

Linear Search



Search Data=8

Linear Search

Search Data=8

Linear Search

Search Data=8
Stop searching
Print “Search data found”

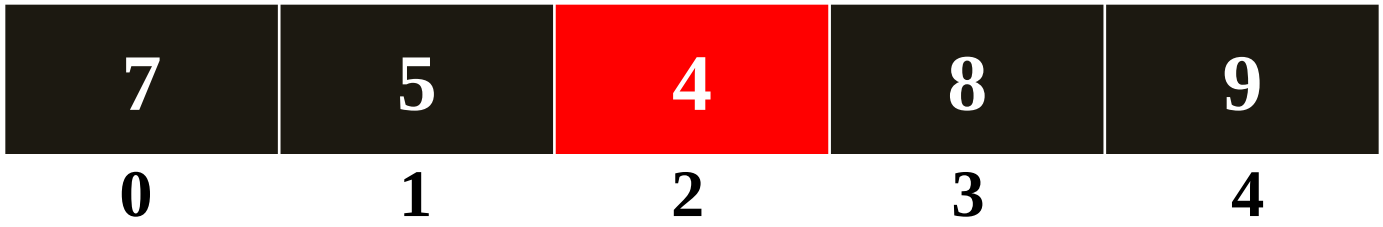
Linear Search

Search Data=1

Linear Search

Search Data=1

Linear Search



Search Data=1

Linear Search

Search Data=1

Linear Search

Search Data=1

Linear Search

Search Data=1

Print “Search data not found”

Algorithm SEARCH_ARRAY(KEY)

Input: KEY is the element to be searched.

Output: Index of KEY in A or a message on failure.

Data structures: An array A[*L* ... *U*]. //*L* and *U* are the lower and upper bound of array index

Steps:

1. *i* = *L*, found = 0, location = 0 //found = 0 indicates search is not finished and unsuccessful
2. While (*i* ≤ *U*) and (found = 0) do // Continue if all or any one condition do(es) not satisfy
    1. If COMPARE(A[*i*], KEY) = TRUE then //If key is found
        1. found = 1 //Search is finished and successful
        2. location = *i*
    2. Else
        1. *i* = *i* + 1 //Move to the next
    3. EndIf
3. EndWhile
4. If found = 0 then
    1. Print "Search is unsuccessful : KEY is not in the array"
5. Else
    1. Print "Search is successful : KEY is in the array at location", location
6. EndIf
7. Return(location)
8. Stop

Linear Search- Algorithm

Algorithm LinearSearch()

```
{       Read n, the size of array
        Read the elements of A[]
        Read the search data search_data
        flag=0
        for i=0 to n-1 do
        {       if A[i]== search_data then
                {       flag=1
                        break
                }
        }
        if flag==0 then
                Print “Search data not found”
        else
                Print “Search data found at the index i”
}
```

Linear Search-Program

```
void main()
{       int A[3],n,i,search_data,flag=0;
        printf("Enter the number of elements: ");
        scanf("%d",&n);
        printf("Enter %d numbers ",n);
        for(i=0;i<n;i++)
                scanf("%d",&A[i]);
        printf("Enter Search Data: ");
        scanf("%d",&search_data);
        for(i=0;i<n;i++)
        {       if(A[i]==search_data)
                {       flag=1;
                        break;
                }
        }
        if(flag==0)
                printf("Search data not found");
        else
                printf("Search data found at the index %d“,i);
}
```

POLYNOMIAL REPRESENTTION USING ARRAY

- Example Polynomial is

  $A(x) = 3x^2 + 2x + 4$ and

  $B(x) = x^4 + 10x^3 + 3x^2 + 1$

- For a mathematician a polynomial is a sum of terms where each term has the form

$$ax^e$$

  - x is the variable,
  - a is the coefficient and
  - e is the exponent.

- However this is not an appropriate definition for our purposes.
- When defining a data object one must decide what functions will be available, what their input is, what their output is and exactly what it is that they do.

POLYNOMIAL REPRESENTATION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$

POLYNOMIAL REPRESENTATION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$

```
struct poly
{
   int coeff;
   int expo;
}p;
```

POLYNOMIAL REPRESENTATION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$

```
struct poly
{
   int coeff;
   int expo;
}p;
```

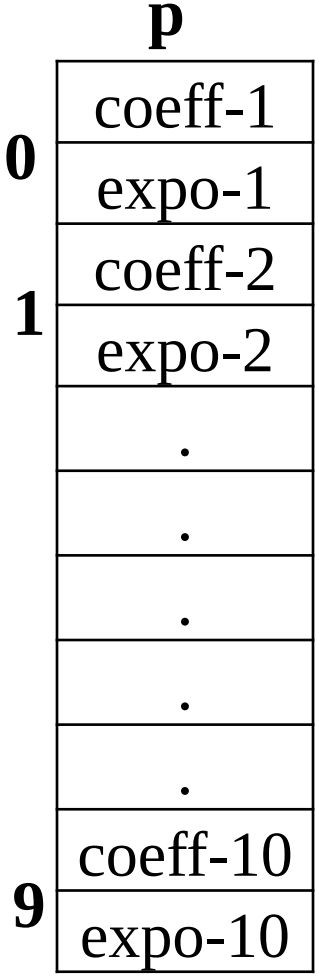
p

coeff
expo

POLYNOMIAL REPRESENTATION USING ARRAY

- **$7x^4 + 3x^2 + 2x + 4$**

```
struct poly
{
   int coeff;
   int expo;
}p[10];
```



POLYNOMIAL REPRESENTATION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$

```
struct poly
{
   int coeff;
   int expo;
}p[10];
```

	p
	7
0	4
	3
1	2
	2
2	1
	4
3	0
	.
	.
9	.

POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

POLYNOMIAL ADDITION USING ARRAY

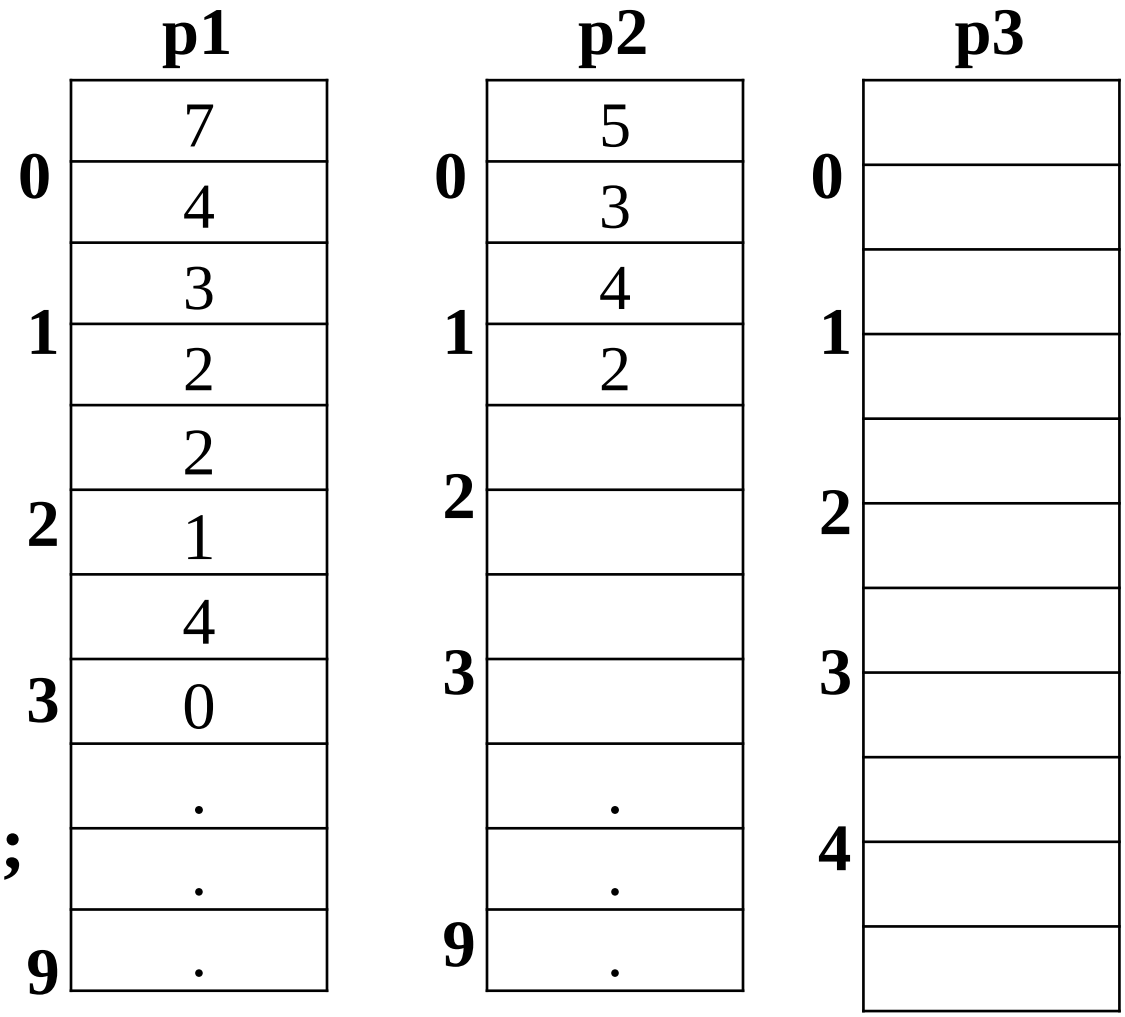
- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```

POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

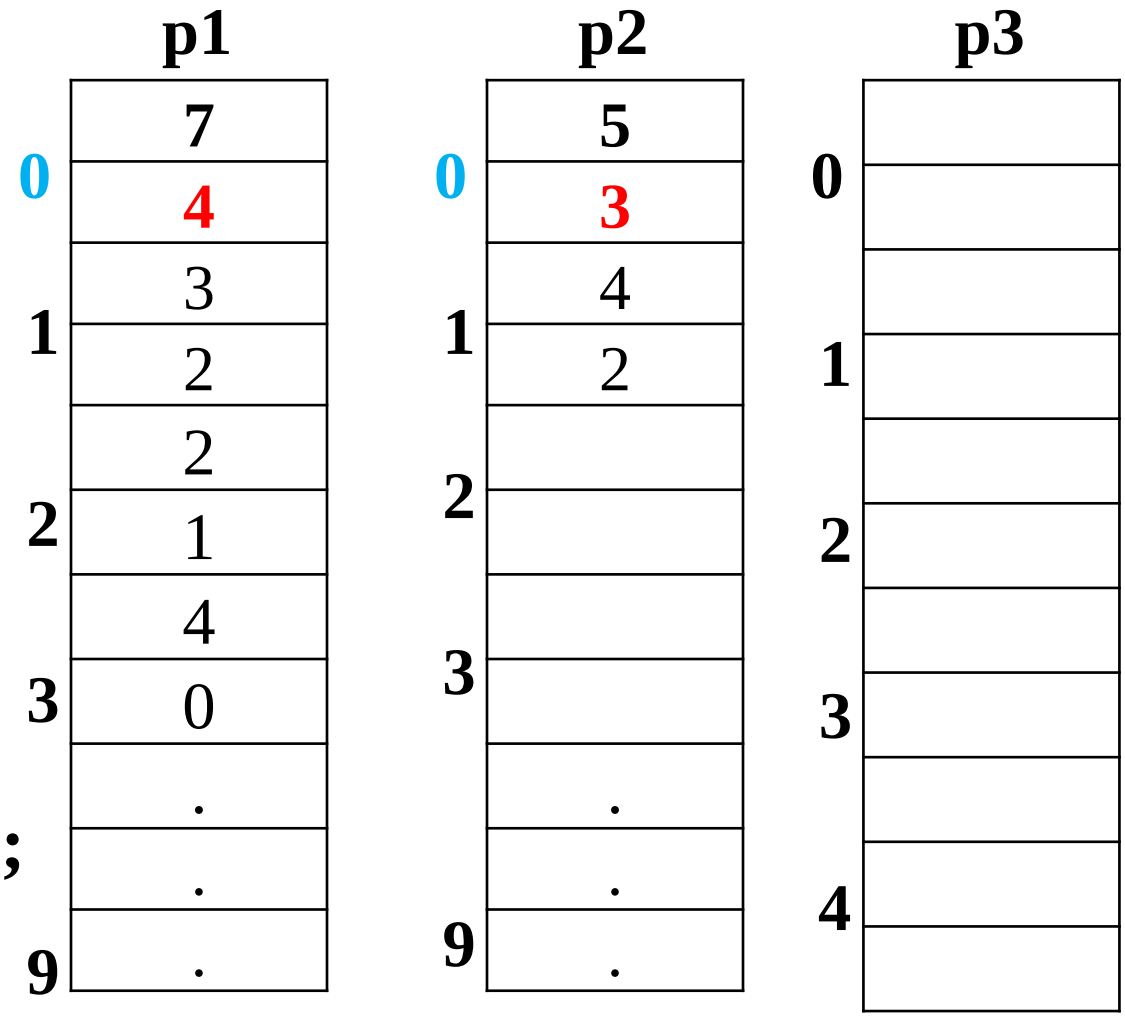
```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```



POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

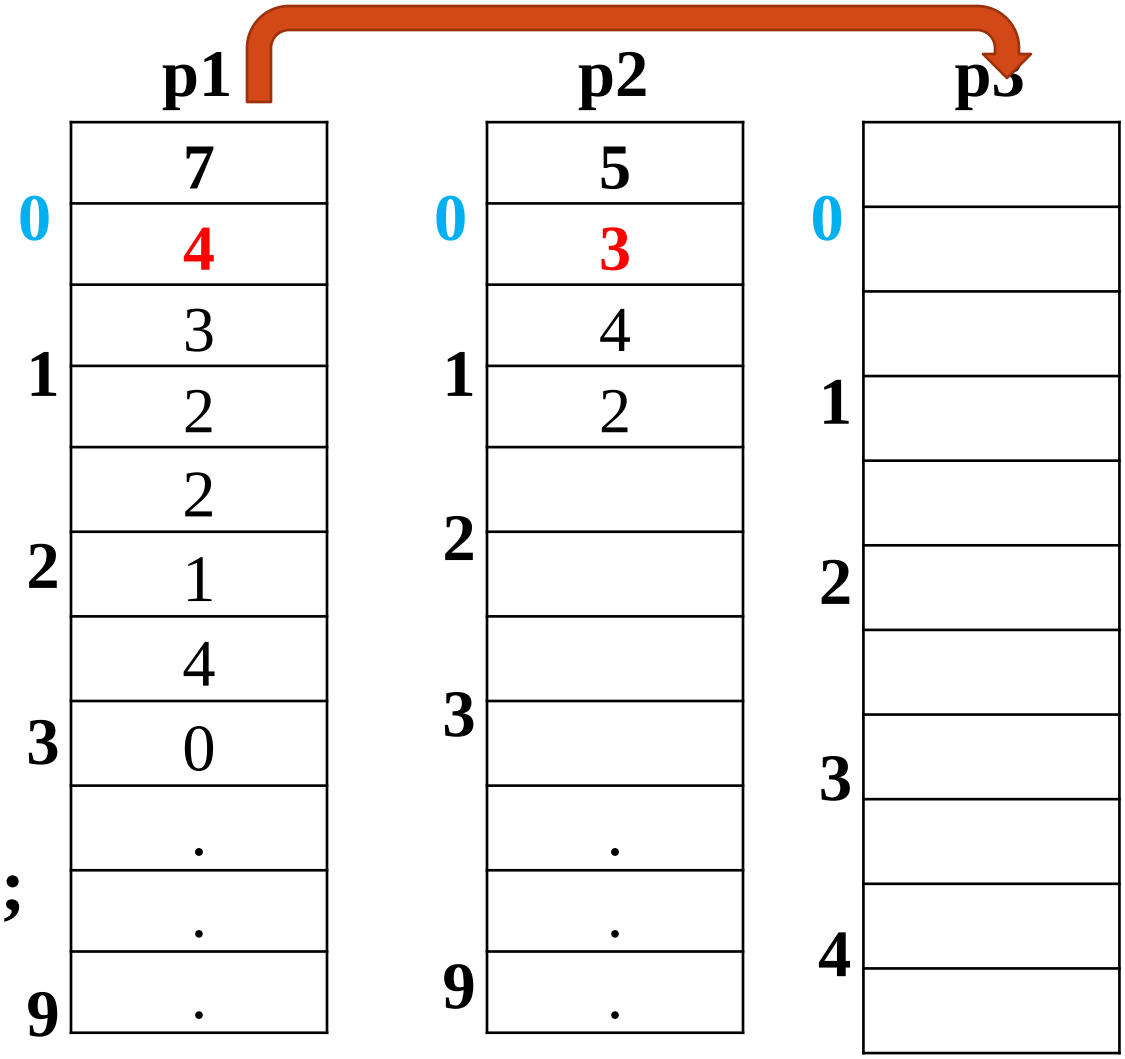
```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```



POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

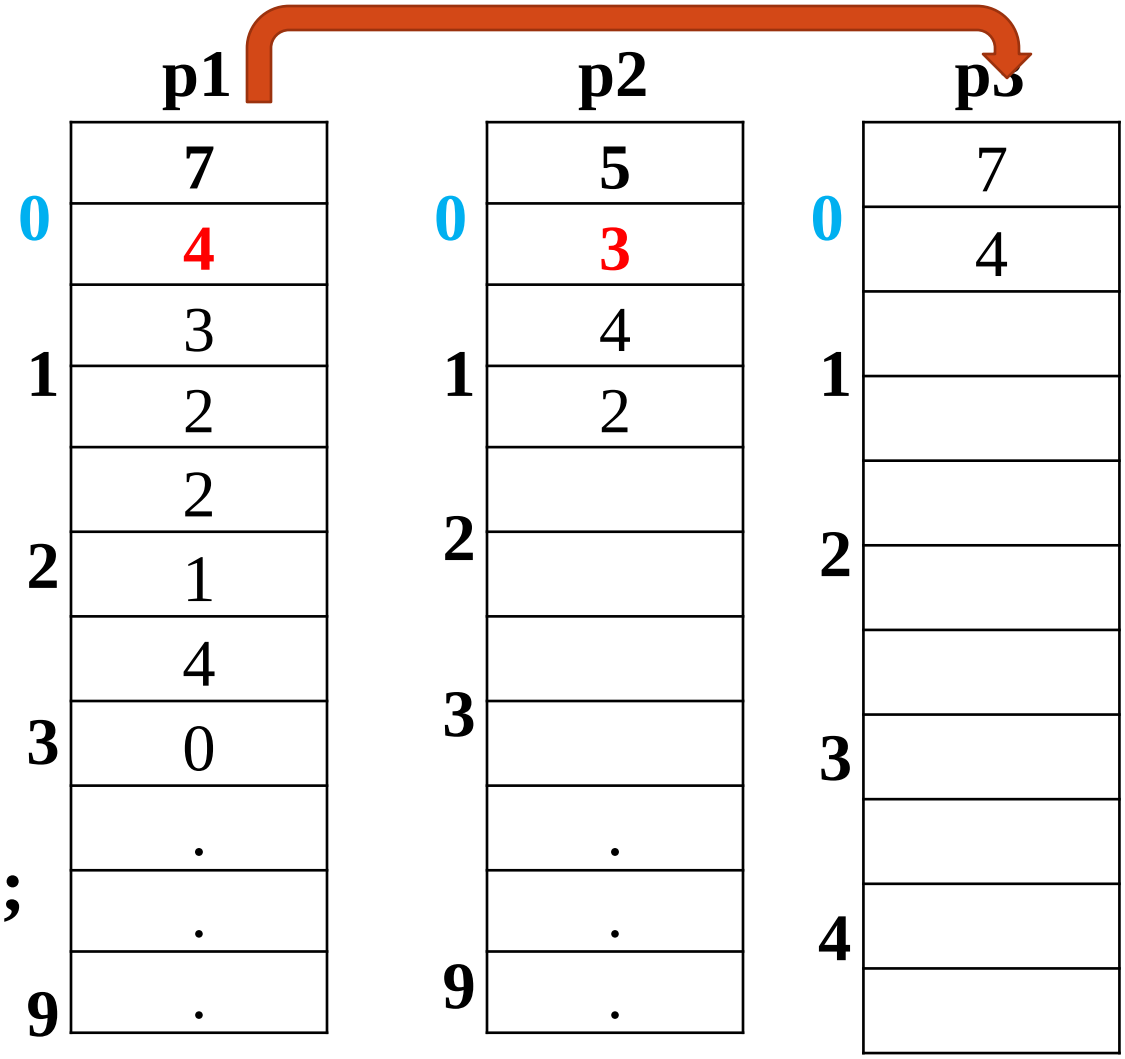
```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```



POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```



POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```

p1

0	7
	4
	3
1	2
	2
2	1
	4
3	0
	.
	.
9	.

p2

0	5
	3
	4
1	2
2	
3	
	.
	.
9	.

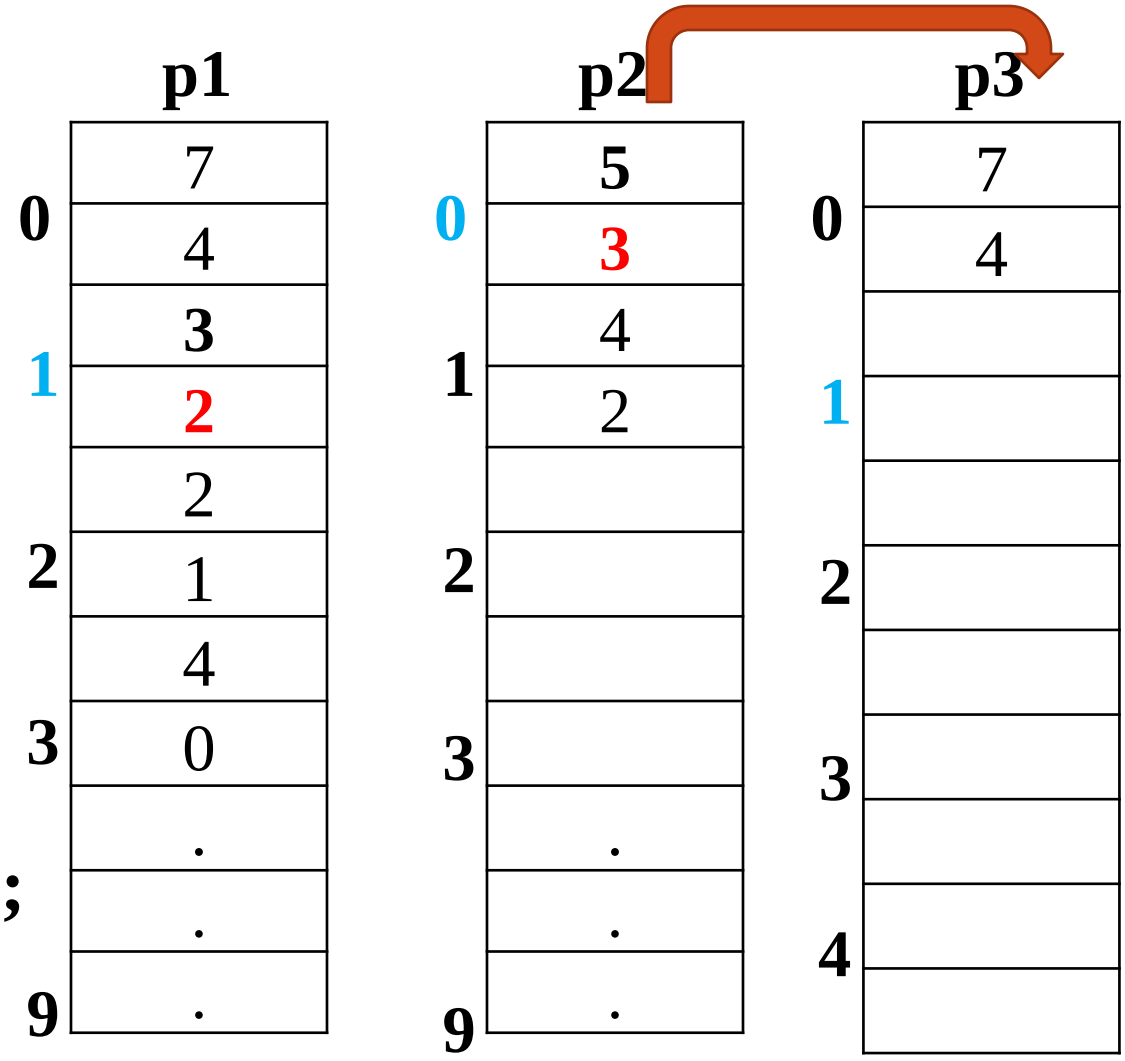
p3

0	7
	4
1	
2	
3	
4	

POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

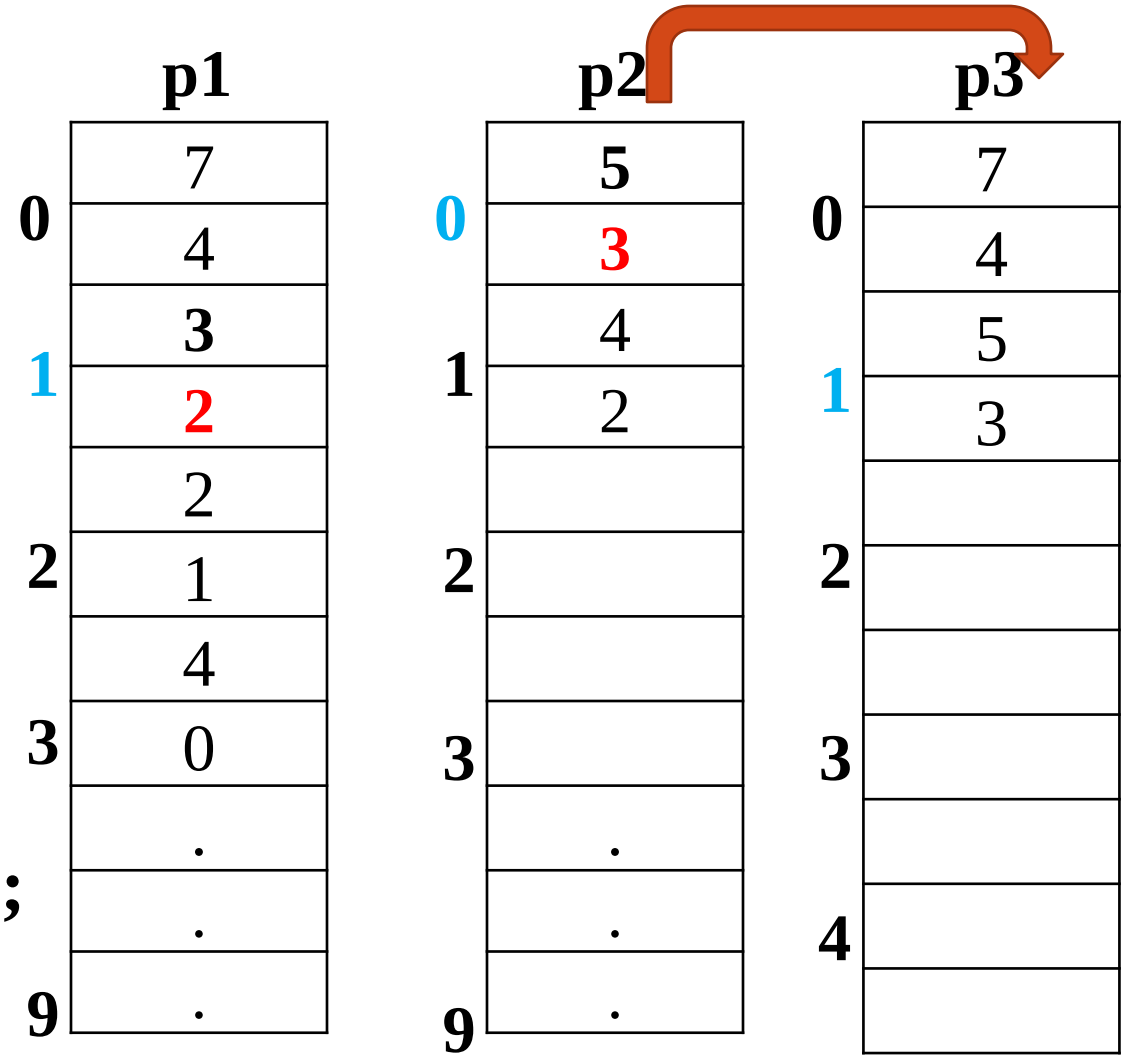
```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```



POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```



POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```

p1

0	7
	4
1	3
	2
2	2
	1
3	4
	0
	.
	.
9	.

p2

0	5
	3
1	4
	2
2	
3	
	.
	.
9	.

p3

0	7
	4
	5
1	3
2	
3	
4	

POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```

	p1	p2	p3
0	7	5	7
	4	3	4
1	3	4	5
	2	2	3
	2		
2	1		2
	4		
3	0		3
	.	.	
	.	.	4
9	.	.	

POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

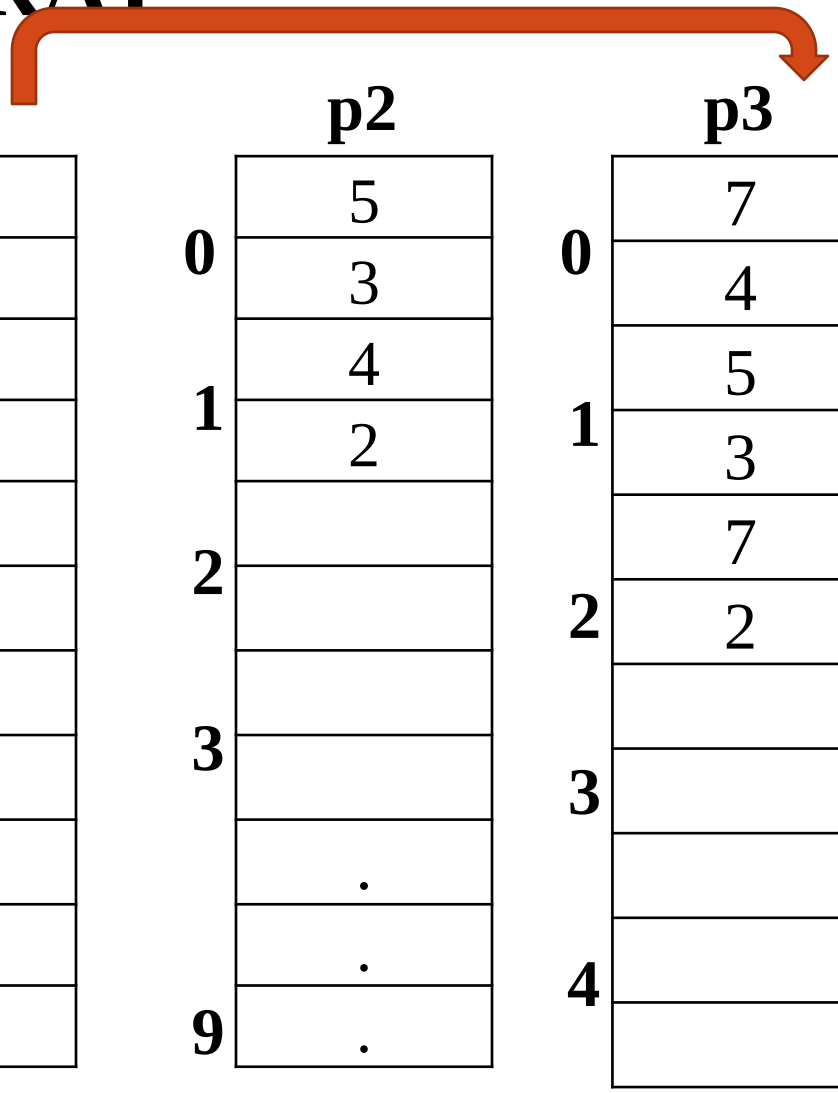
```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```

	p1	p2	p3
0	7	5	7
	4	3	4
1	3	4	5
	2	2	3
	2		7
2	1		2
	4		
3	0		
	.	.	
	.	.	4
9	.	.	

POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

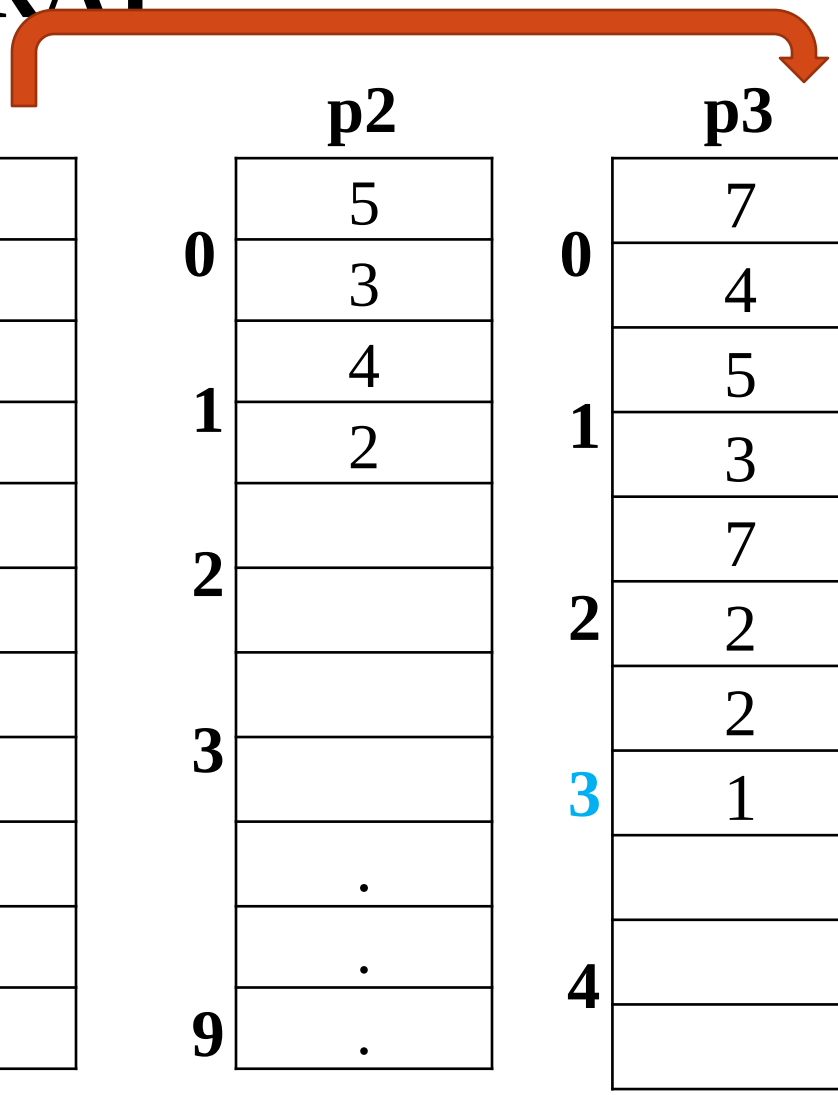
```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```



POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

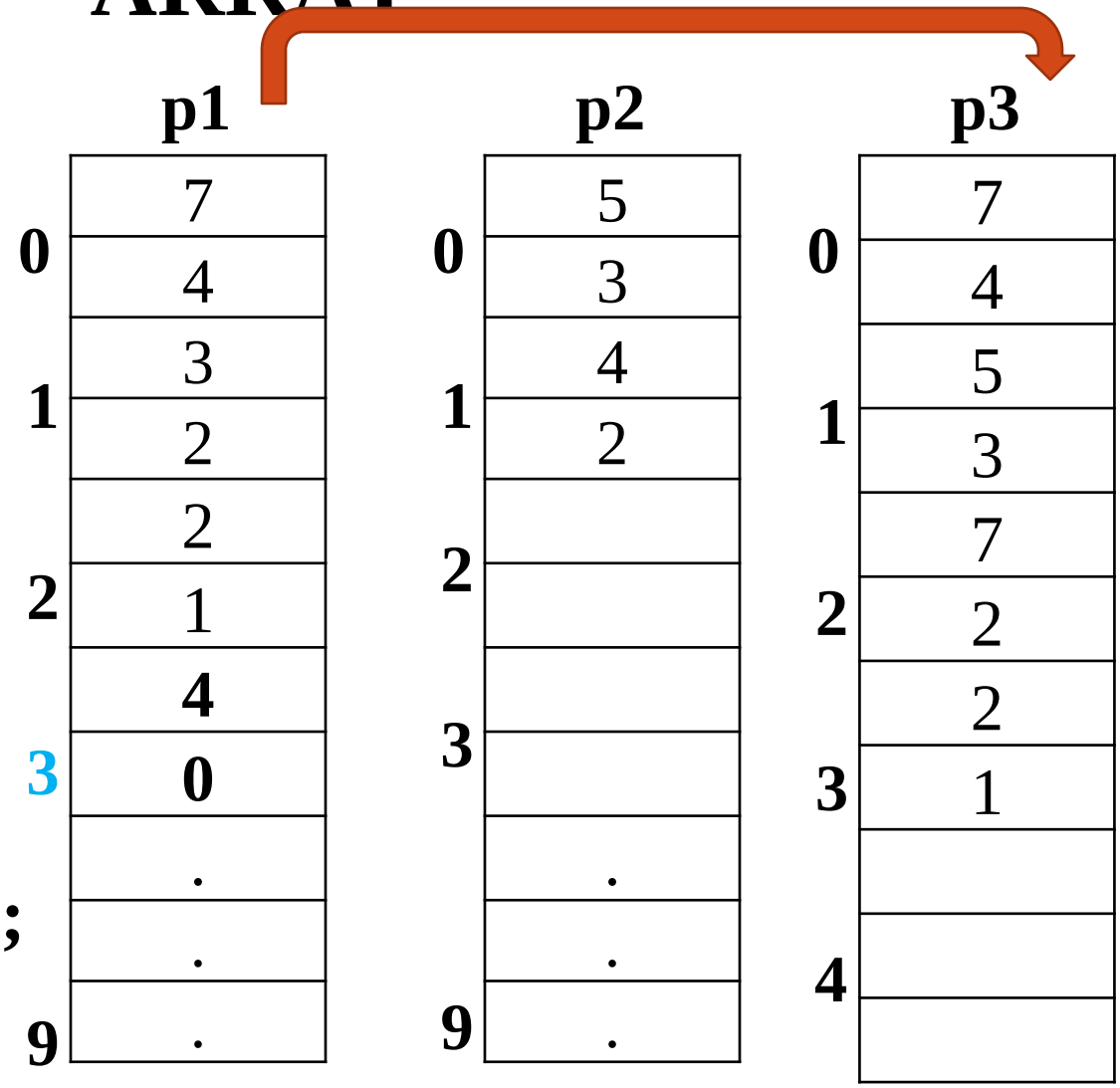
```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```



POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

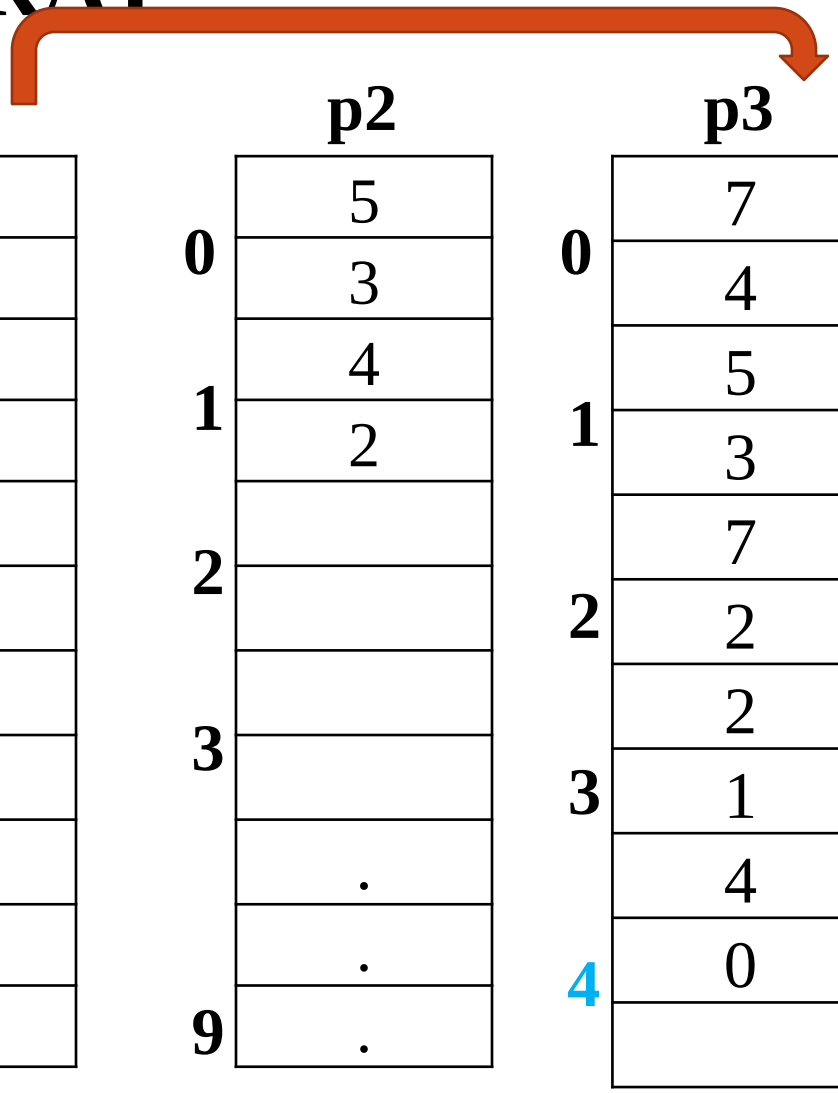
```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```



POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

```
struct poly
{
   int coeff;
   int expo;
}p1[10],p2[10],p3[10];
```



POLYNOMIAL ADDITION USING ARRAY

- $7x^4 + 3x^2 + 2x + 4$
- $5x^3 + 4x^2$

- **Sum =** $7x^4 + 5x^3 + 7x^2 + 2x^1 + 4x^0$

	p3
	7
0	4
	5
1	3
	7
2	2
	2
3	1
	4
4	0

Polynomial Addition- Algorithm

Algorithm main()

{

Declare a structure to keep the coefficient and exponent of polynomial

Declare structure arrays p1[10],p2[10],p3[10]

call t1=readPoly(p1)

call displayPoly(p1,t1)

call t2=readPoly(p2)

call displayPoly(p2,t2)

call t3=addPoly(p1,p2,t1,t2)

call displayPoly(p3,t3)

}

Polynomial Addition- Algorithm

Algorithm readPoly(p)

```
{
    Read t, the number of terms of the polynomial
    for i=0 to t-1 do
        read p[i].coeff and p[i].expo
    return t
}
```

Algorithm displayPoly(p,t)

```
{
    for i=0 to t-1 do
        display p[i].coeff , ”(x^” ,p[i].expo ”)+”
}
```

Algorithm addPoly(p1,p2,t1,t2,p3)

```
{     i=0,j=0,k=0
      while i<t1 && j<t2 do
      {    if p1[i].expo==p2[j].expo then
           {          p3[k].coeff=p1[i].coeff + p2[j].coeff
                      p3[k].expo=p1[i].expo
                      i=i+1
                      j=j+1
                      k=k+1
           }
           else if p1[i].expo>p2[j].expo then
           {          p3[k].coeff=p1[i].coeff
                      p3[k].expo=p1[i].expo
                      i=i+1
                      k=k+1
           }
```

```
        else
        {       p3[k].coeff=p2[j].coeff
                p3[k].expo=p2[j].expo
                j=j+1           k=k+1
        }
    }
    while i<t1 do
    {   p3[k].coeff=p1[i].coeff
        p3[k].expo=p1[i].expo
        i=i+1           k=k+1
    }
    while j<t2 do
    {   p3[k].coeff=p2[j].coeff
        p3[k].expo=p2[j].expo
        j=j+1           k=k+1
    }
    return(k)
}
```

Polynomial Addition- Program

```
struct poly
{
        int coeff;
        int expo;
}p1[10],p2[10],p3[10];

int readPoly(struct poly []);
int addPoly(struct poly [],struct poly [],int ,int ,struct poly []);
void displayPoly(struct poly [],int terms);
```

```
void main()
{     int t1,t2,t3;

      t1=readPoly(p1);
      printf("\nFirst polynomial : ");
      displayPoly(p1,t1);

      t2=readPoly(p2);
      printf("\nSecond polynomial : ");
      displayPoly(p2,t2);

      /* add two polynomials and display resultant polynomial */
      t3=addPoly(p1,p2,t1,t2,p3);
      printf("\n\nResultant polynomial after addition : ");
      displayPoly(p3,t3);
}
```

```
int readPoly(struct poly p[10])
{     int t1,i;

      printf("\nEnter the total number of terms in the polynomial:");
      scanf("%d",&t1);

      printf("\nEnter the COEFFICIENT and EXPONENT in
      DESCENDING ORDER\n");
      for(i=0;i<t1;i++)
      {    printf("   Enter the Coefficient(%d): ",i+1);
           scanf("%d",&p[i].coeff);
           printf("      Enter the exponent(%d): ",i+1);
           scanf("%d",&p[i].expo);
      }
      return(t1);
 }
```

```
void displayPoly(struct poly p[10],int term)
{
	int k;
	for(k=0;k<term-1;k++)
		printf("%d(x^%d)+",p[k].coeff,p[k].expo);

	printf("%d(x^%d)",p[term-1].coeff,p[term-1].expo);
}
```

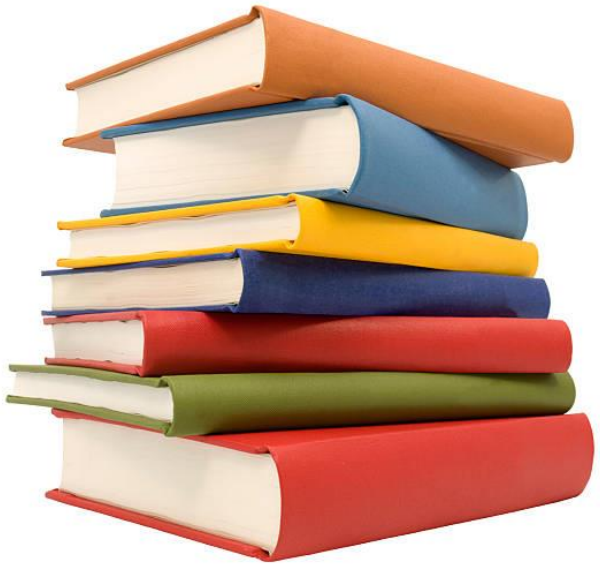
```
int addPoly(struct poly p1[10],struct poly p2[10],int t1,int t2,struct poly p3[10])
{       int i=0,j=0,k=0;
        while(i<t1 && j<t2)
        {    if(p1[i].expo==p2[j].expo)
             {          p3[k].coeff=p1[i].coeff + p2[j].coeff;
                        p3[k].expo=p1[i].expo;
                        i++;       j++;       k++;
             }
             else if(p1[i].expo>p2[j].expo)
             {          p3[k].coeff=p1[i].coeff;
                        p3[k].expo=p1[i].expo;
                        i++;       k++;
             }
             else
             {          p3[k].coeff=p2[j].coeff;
                        p3[k].expo=p2[j].expo;
                        j++;       k++;
             }
        }
```

```
    /* for rest over terms of polynomial 1 */
    while(i<t1)
    {
        p3[k].coeff=p1[i].coeff;
        p3[k].expo=p1[i].expo;
        i++;        k++;
    }

    /* for rest over terms of polynomial 2 */
    while(j<t2)
    {
        p3[k].coeff=p2[j].coeff;
        p3[k].expo=p2[j].expo;
        j++;        k++;
    }

    return(k);          /* k is number of terms in resultant polynomial*/
}
```

STACK



- Stack is a linear data structure
- In case of array , insertion and deletion take place at any position
- Stack is an ordered collection of homogenous data elements where the **insertion and deletion operation takes place at one end only**
- It is a **Last in First Out (LIFO)** memory

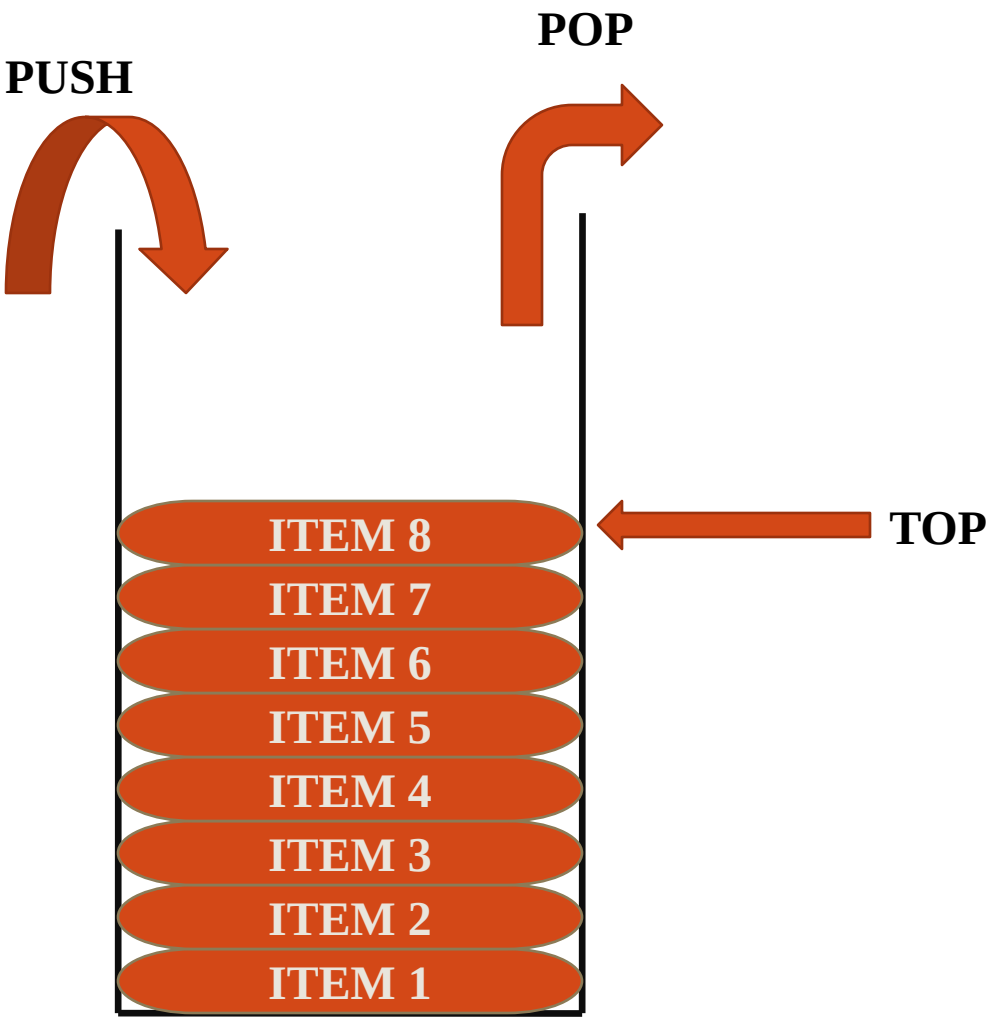
OPERATIONS ON STACK

- **PUSH**- Insert an element into the stack
- **POP**- Delete an element from the stack
- **STATUS-** To Know the present state of stack
- **DISPLAY** – To display the elements of the stack

Some Terminologies

- **TOP**- Position of the stack where PUSH and POP operations are performed
- **ITEM** – An element in a stack
- **SIZE** – Maximum number of elements that a stack can accommodate

STACK


POP
PUSH
ITEM 8
ITEM 7
ITEM 6
ITEM 5
ITEM 4
ITEM 3
ITEM 2
ITEM 1
TOP

STACK

- **Real time examples:**
  - Trains in a railway yard
  - Shipment of cargo

- **Applications of Stack:**
  - Evaluation of Arithmetic Expressions
  - Implementation of Recursion

REPRESENTATION OF STACK IN MEMORY

Two representations:

1. Array Representation
2. Linked List Representation

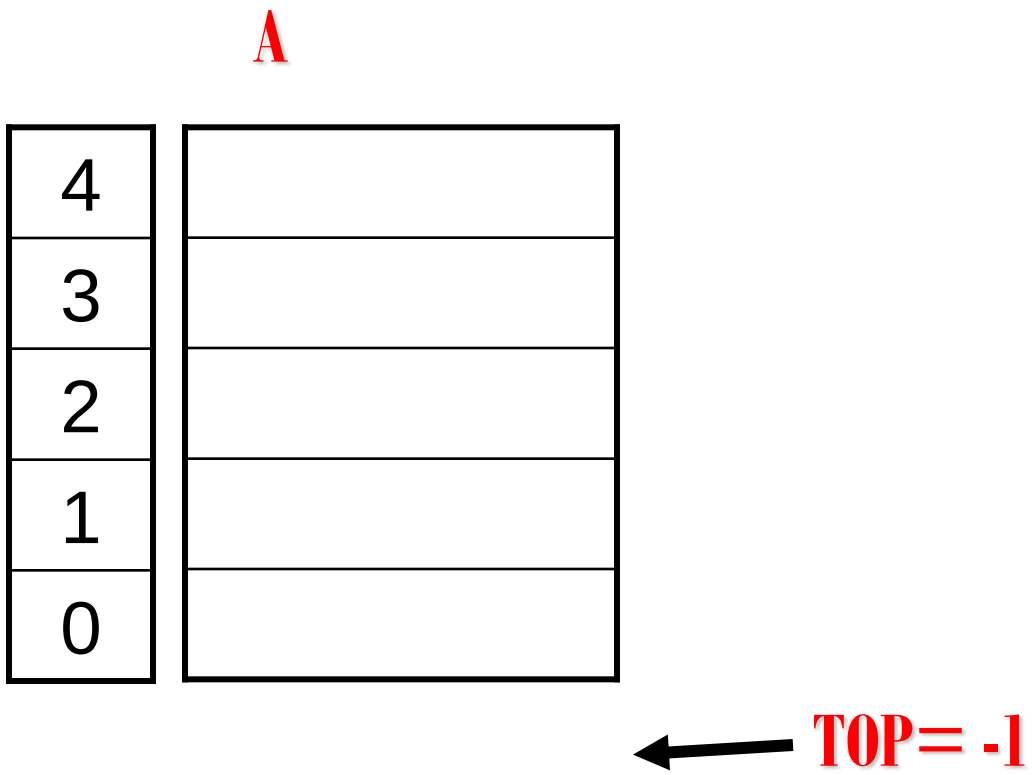
STACK DATA STRUCTURE ALGORITHMS

STACK REPRESENTATION USING ARRAY

int A[5];

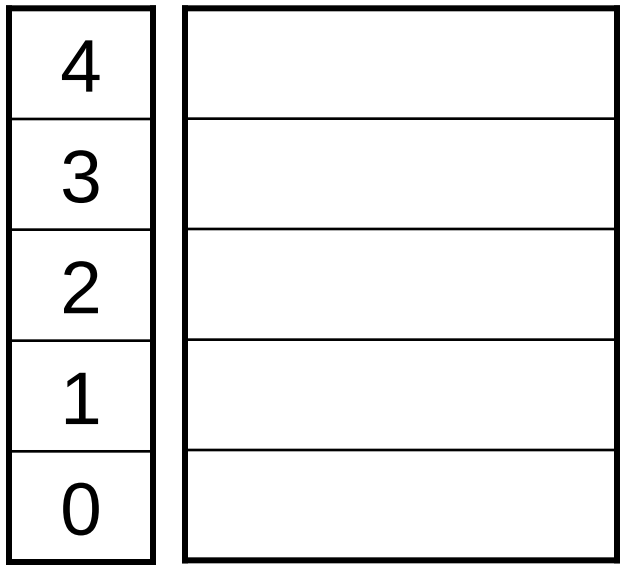
If TOP==-1

Stack is EMPTY



PUSH ITEM1

A

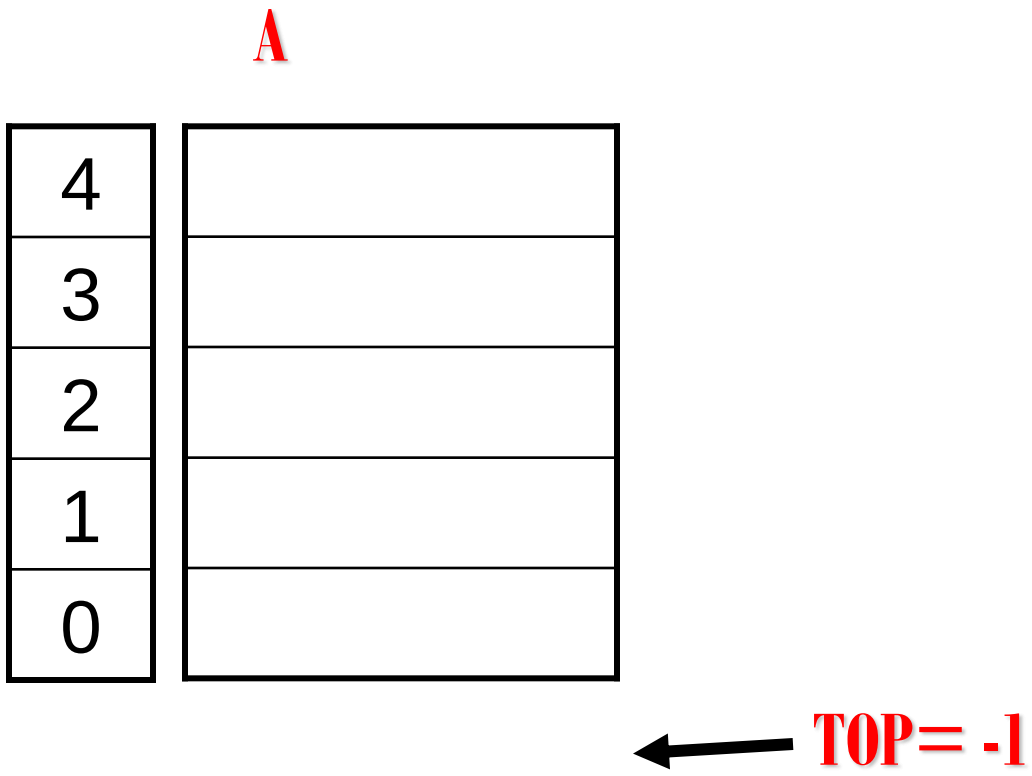


TOP= -1

TOP=TOP+1

A[TOP]=ITEM1

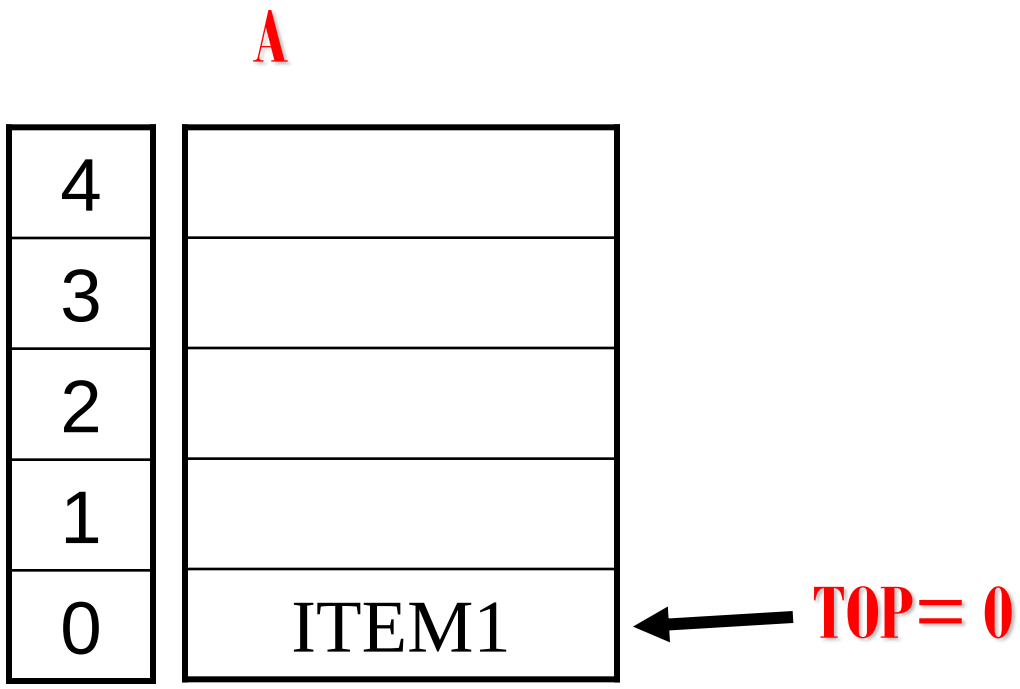
PUSH ITEM1



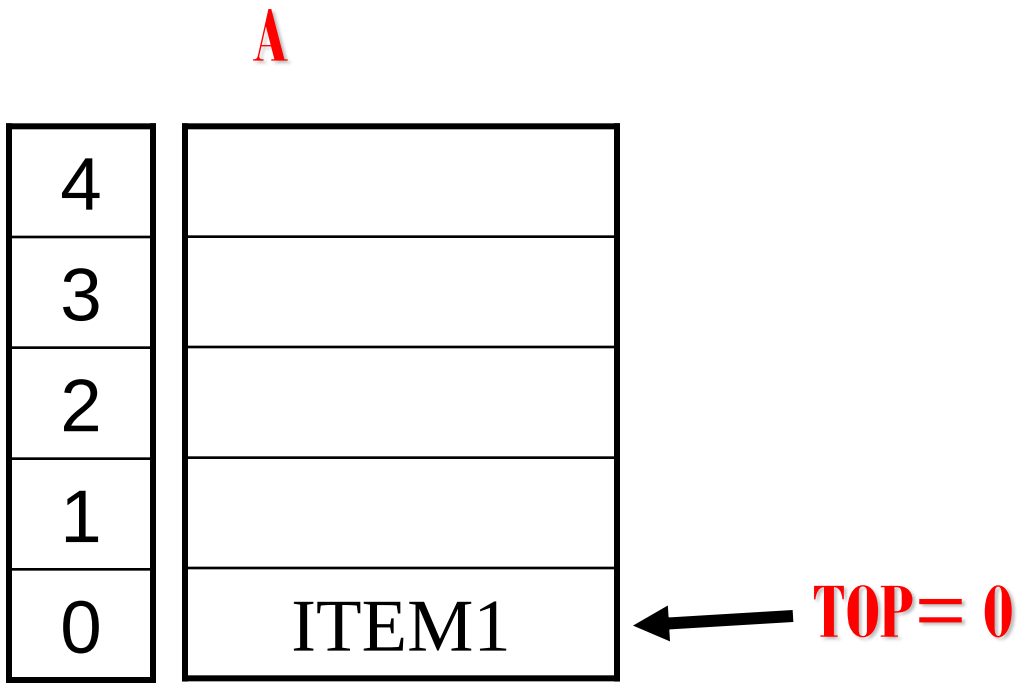
TOP=TOP+1

A[TOP]=ITEM1

PUSH ITEM1



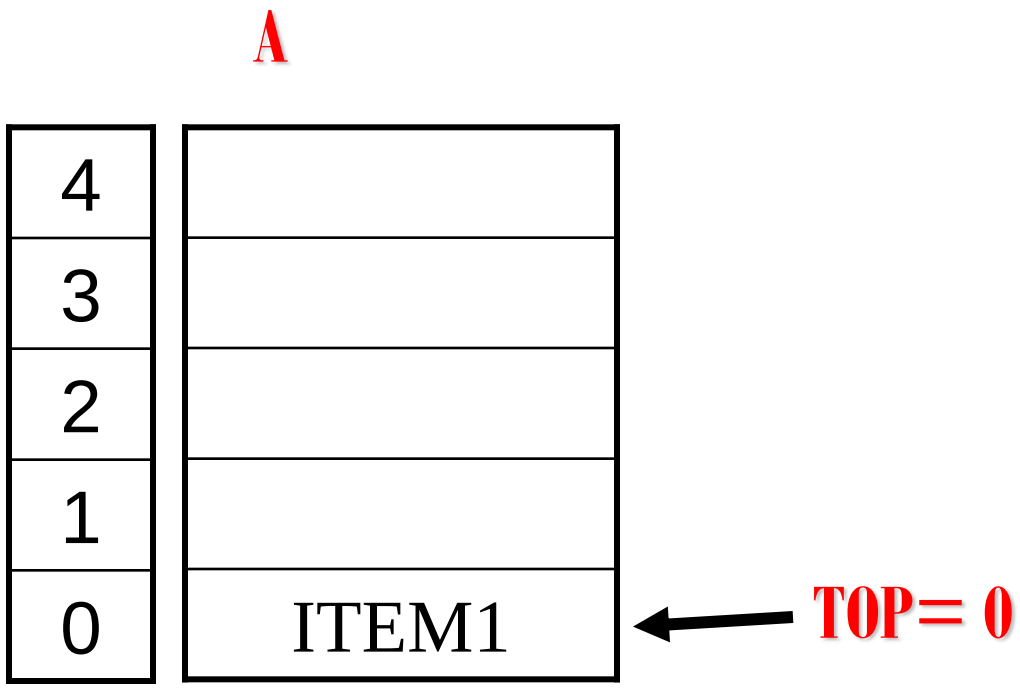
PUSH ITEM2



TOP=TOP+1

A[TOP]=ITEM2

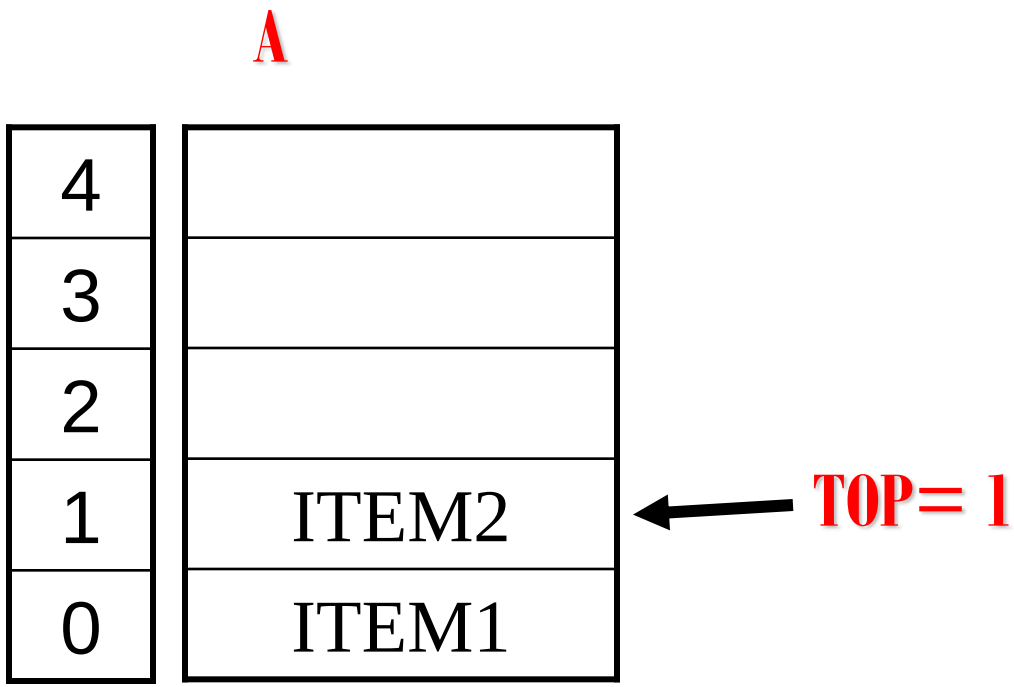
PUSH ITEM2



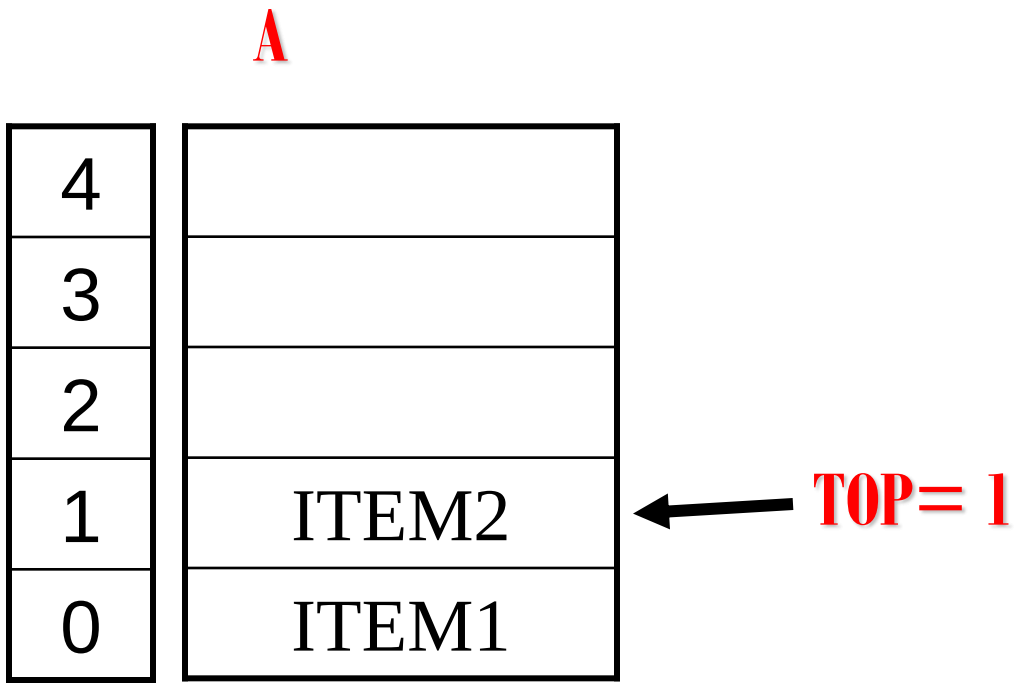
TOP=TOP+1

A[TOP]=ITEM2

PUSH ITEM2



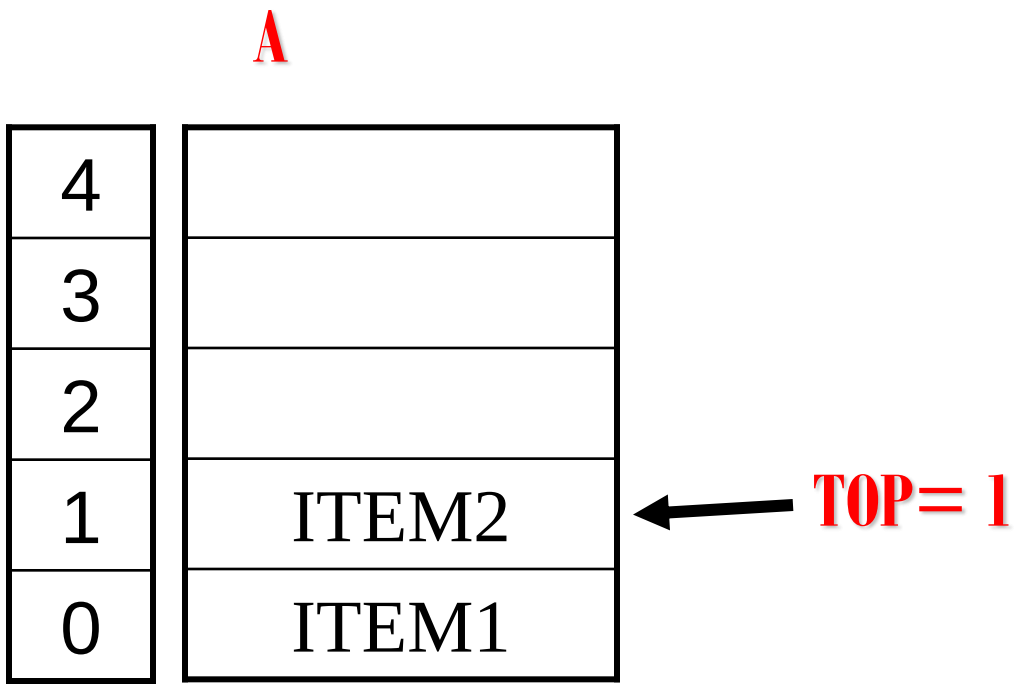
PUSH ITEM3



TOP=TOP+1

A[TOP]=ITEM3

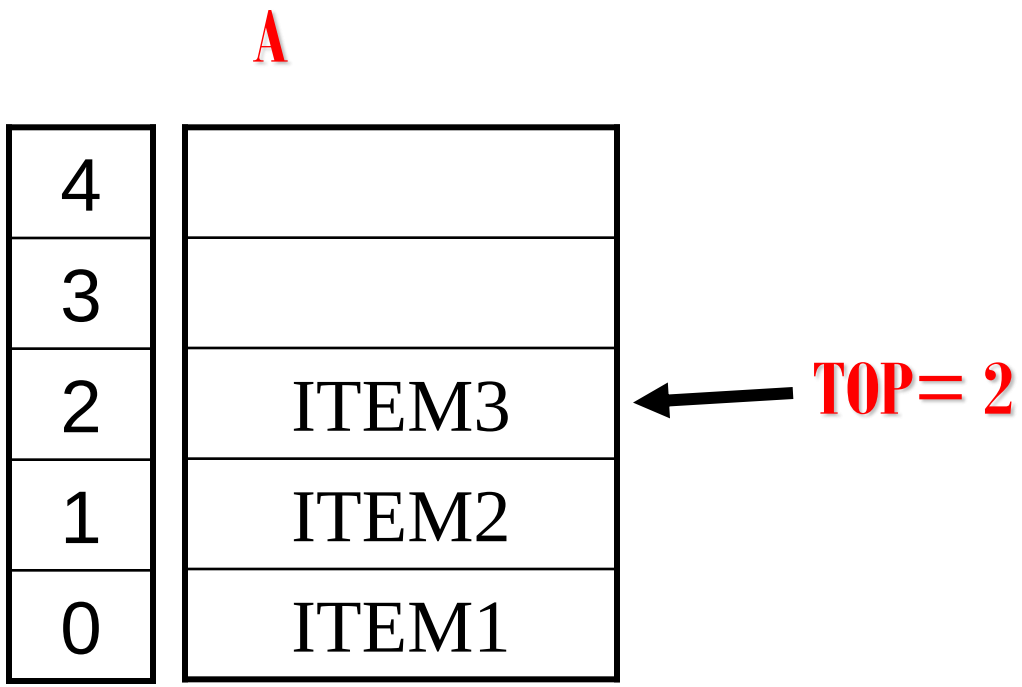
PUSH ITEM3



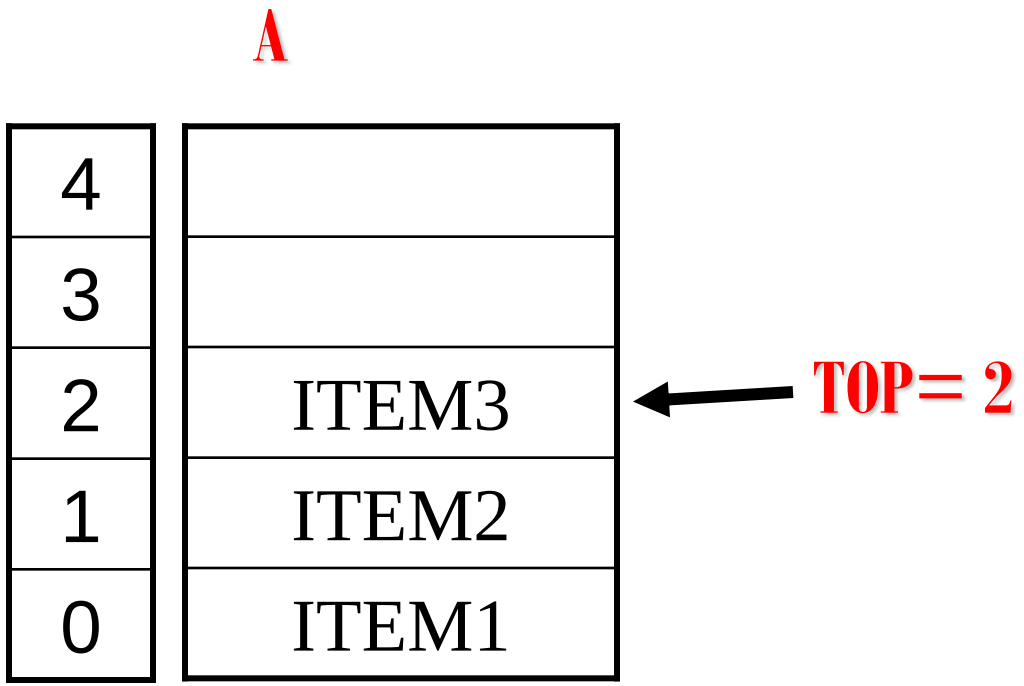
TOP=TOP+1

A[TOP]=ITEM3

PUSH ITEM3



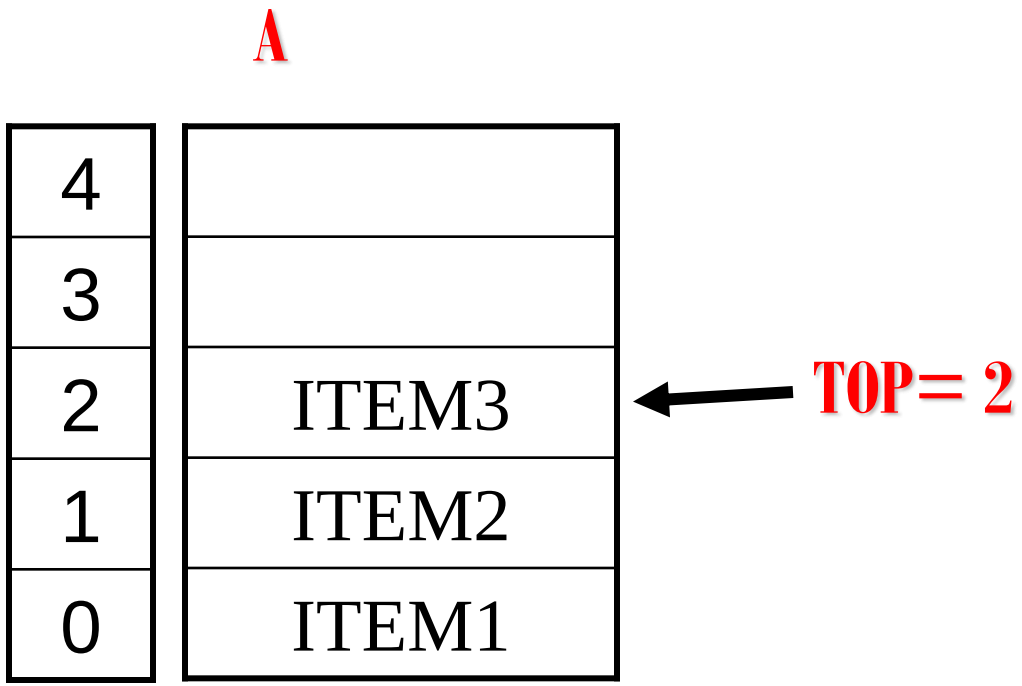
PUSH ITEM4



TOP=TOP+1

A[TOP]=ITEM4

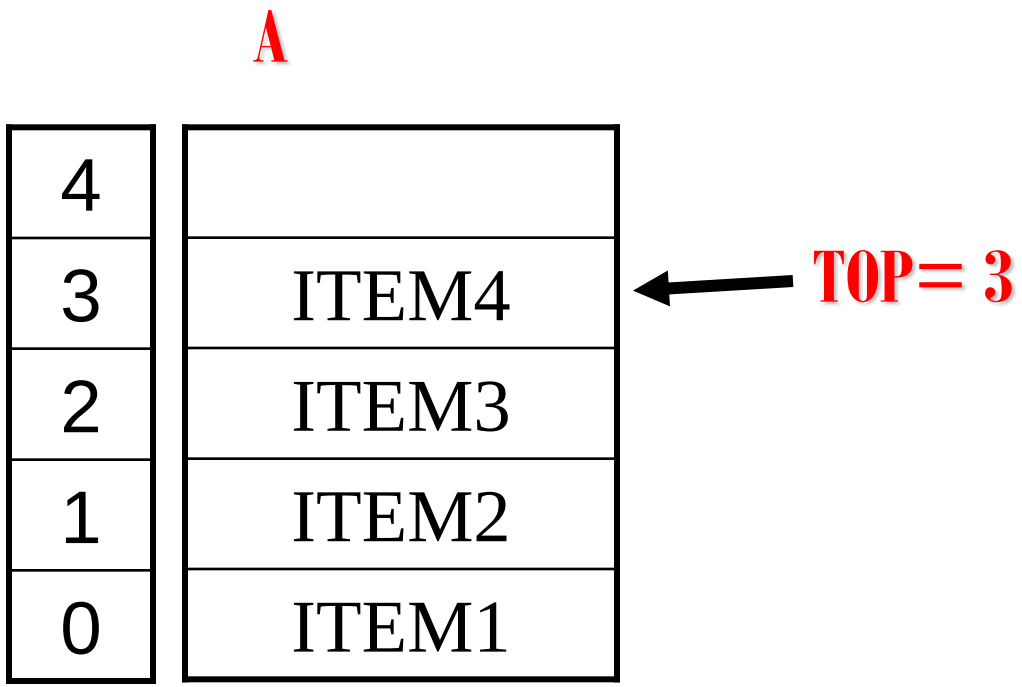
PUSH ITEM4



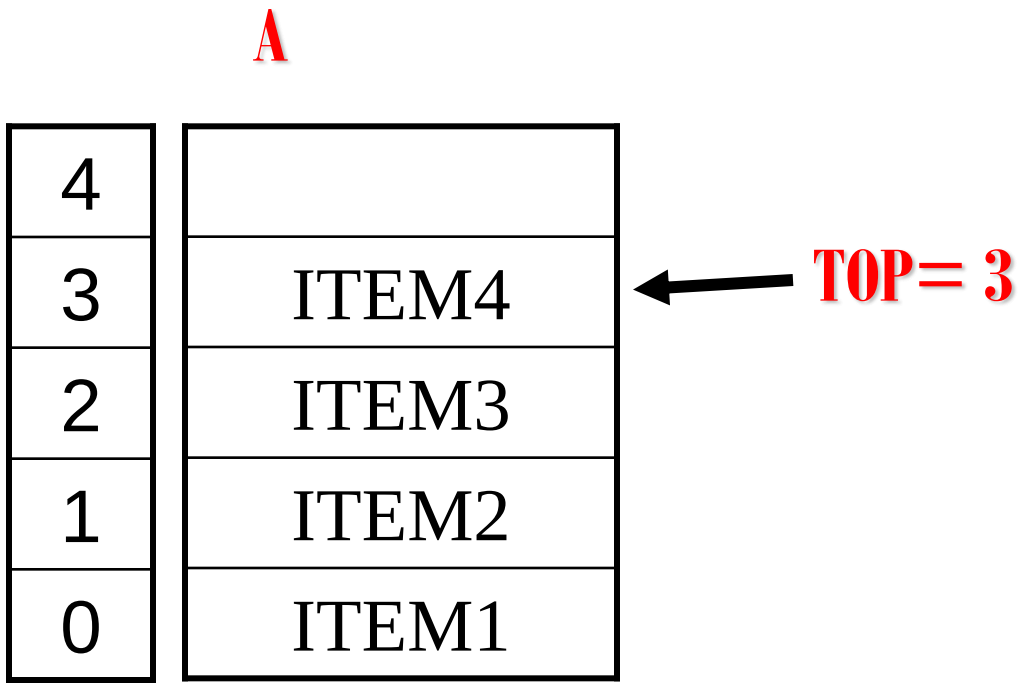
TOP=TOP+1

A[TOP]=ITEM4

PUSH ITEM4



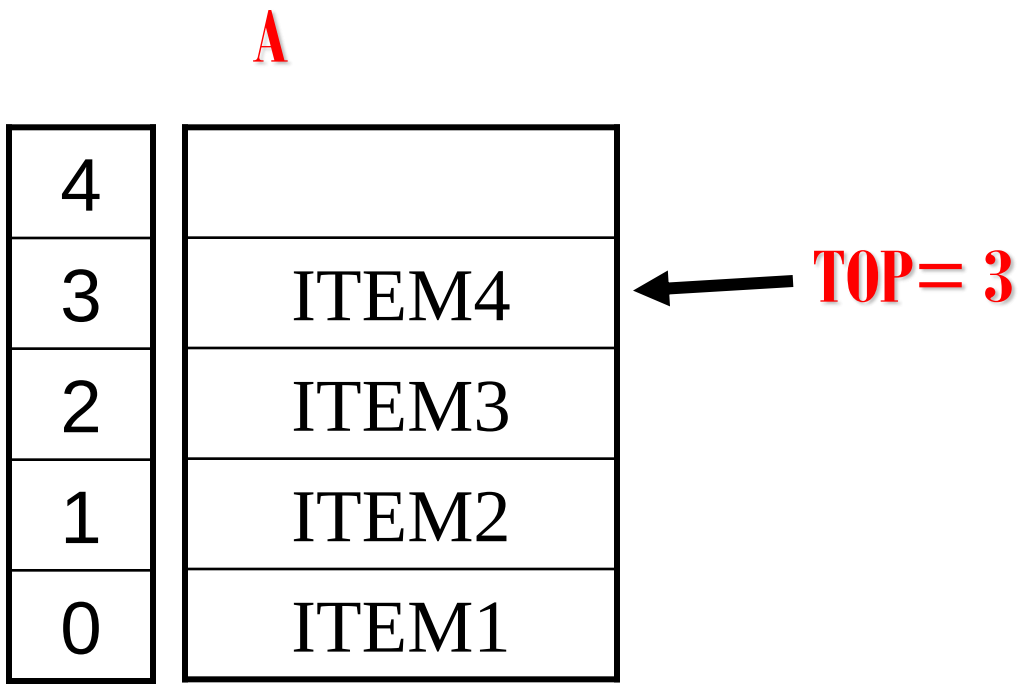
PUSH ITEM5



TOP=TOP+1

A[TOP]=ITEM5

PUSH ITEM5

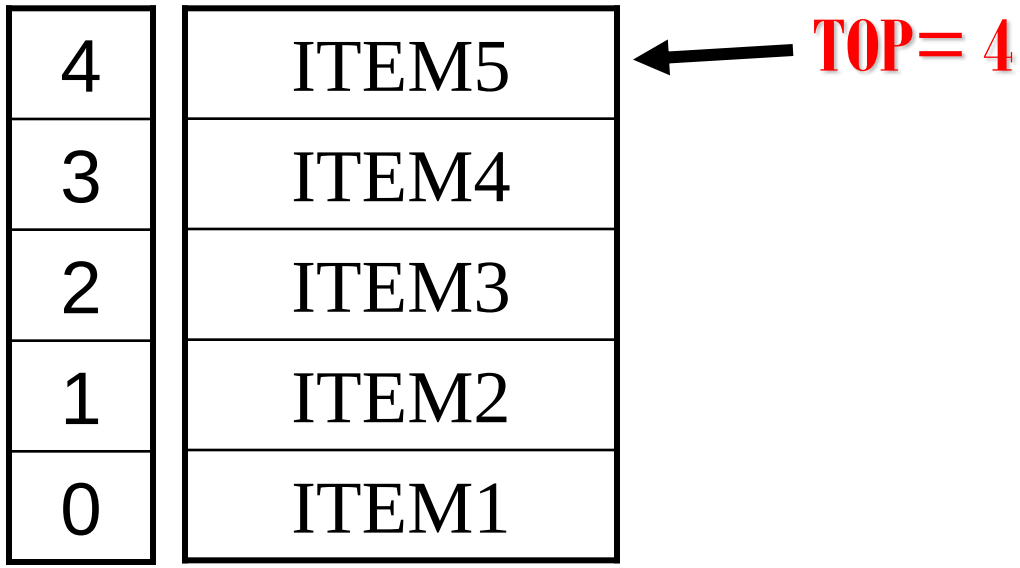


TOP=TOP+1

A[TOP]=ITEM5

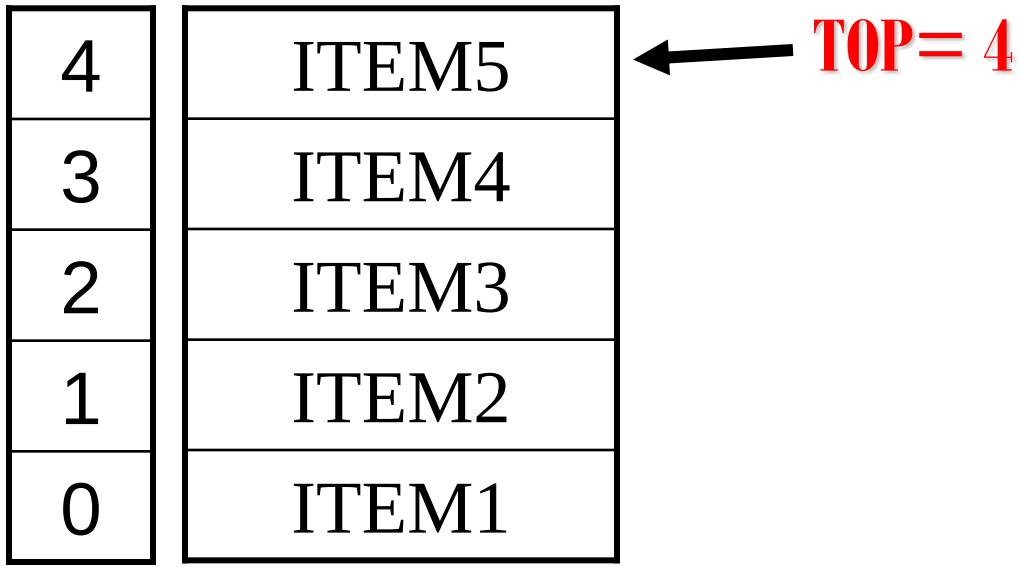
A

PUSH ITEM5



PUSH ITEM6

A

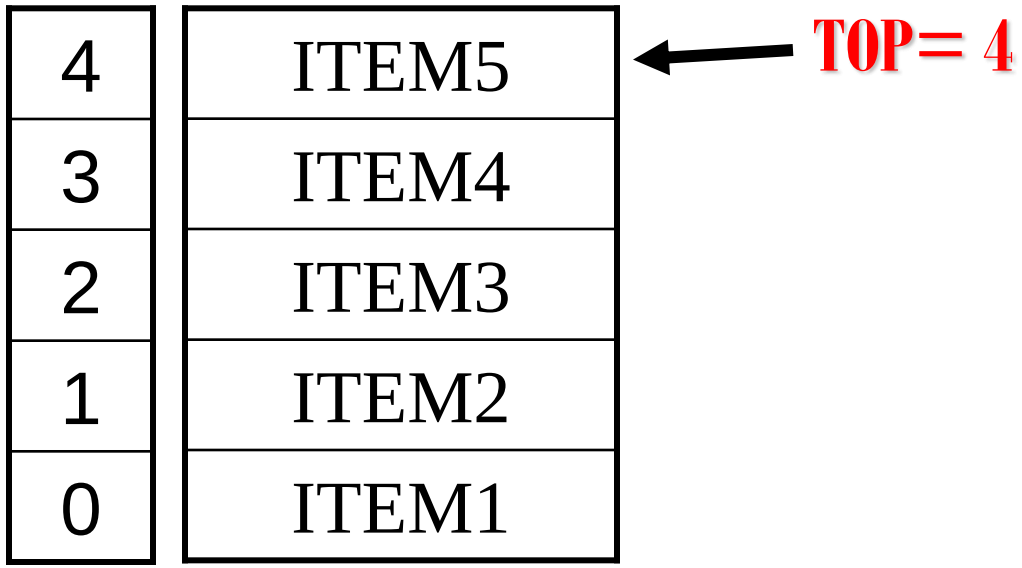


If TOP=SIZE-1 then

Stack is full

PUSH ITEM6

A



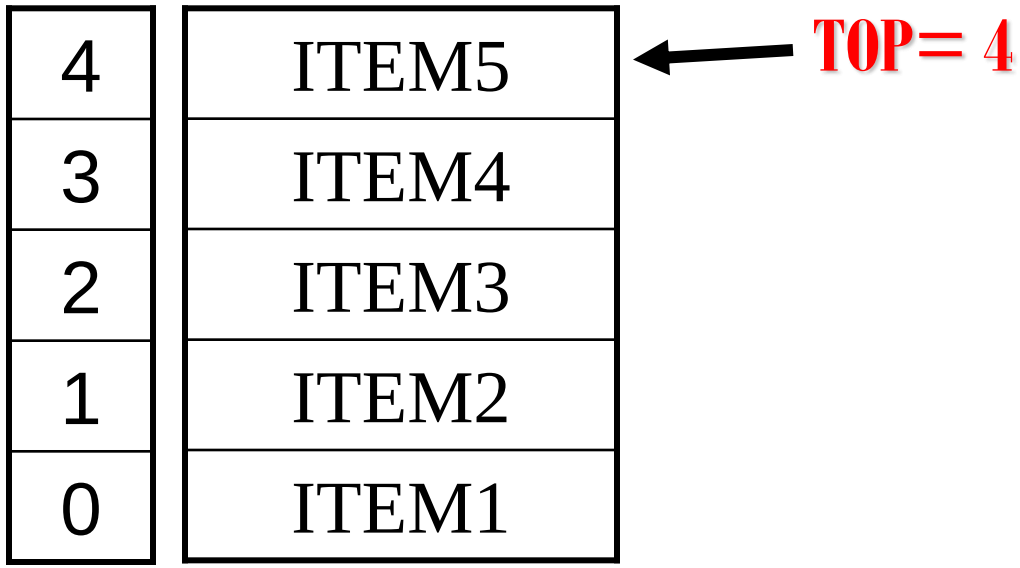
STACK – PUSH ALGORITHM

Algorithm PUSH(ITEM)

```
{
        if  TOP = SIZE – 1 then
                Print “Stack is FULL”
        else
        {
                TOP=TOP+1
                A[TOP] = ITEM
        }
}
```

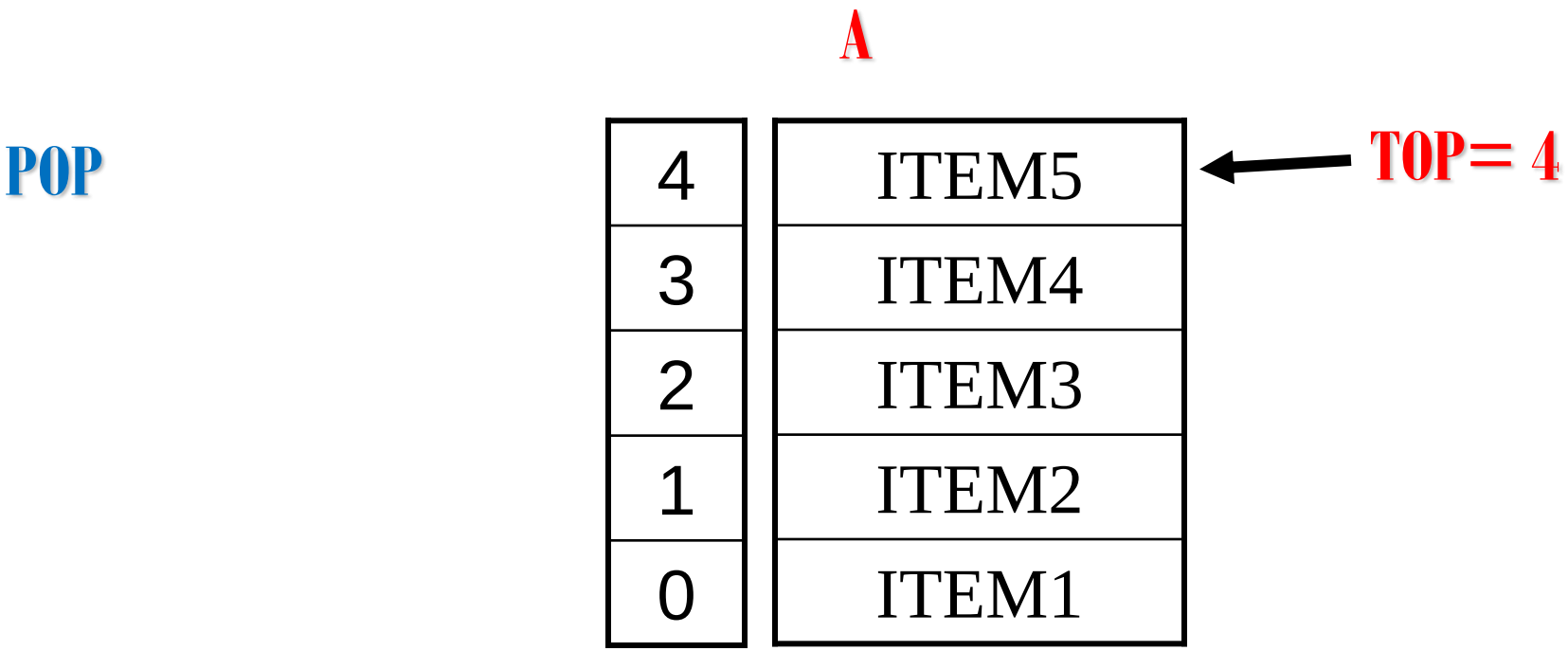
POP

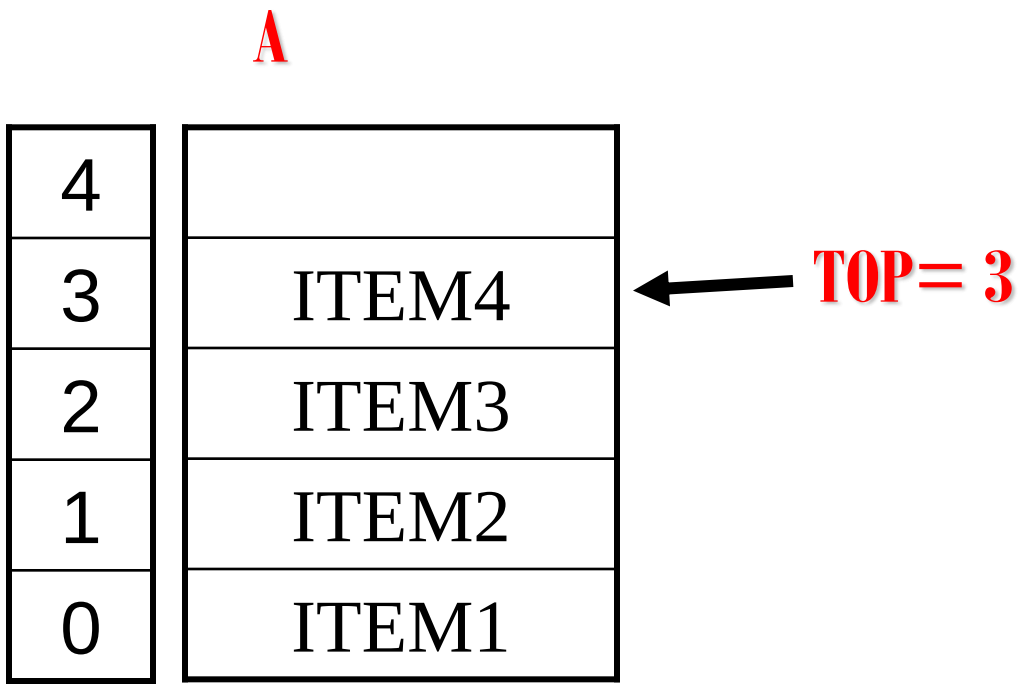
A



ITEM=A[TOP]

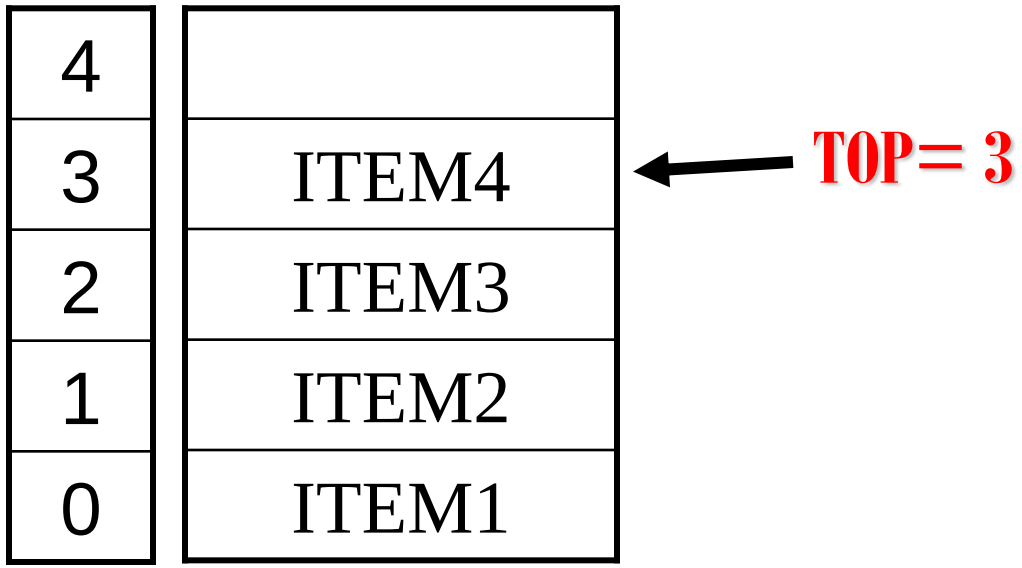
TOP=TOP-1




A
4
3
2
1
0
ITEM4
ITEM3
ITEM2
ITEM1
TOP= 3

POP

A

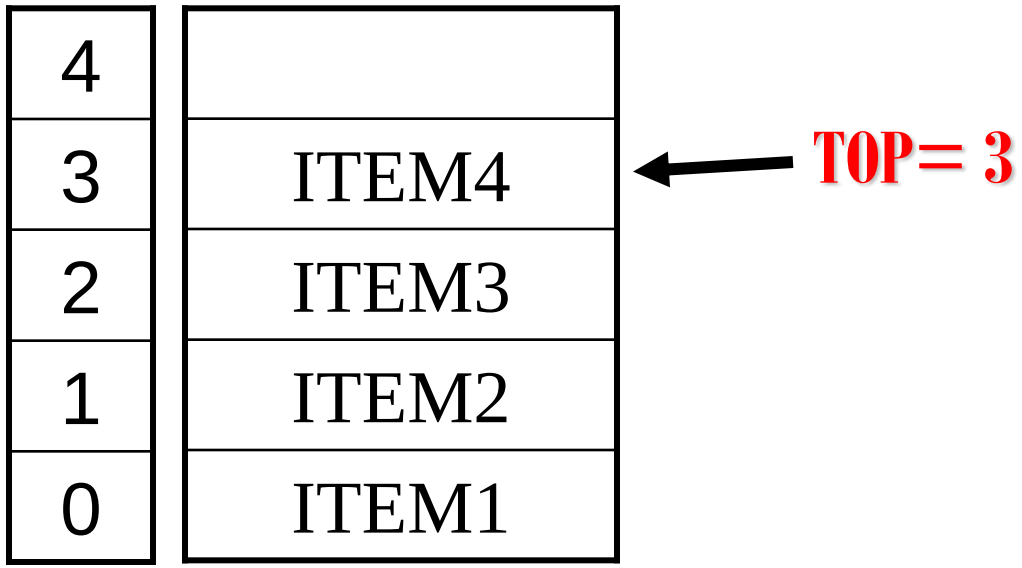


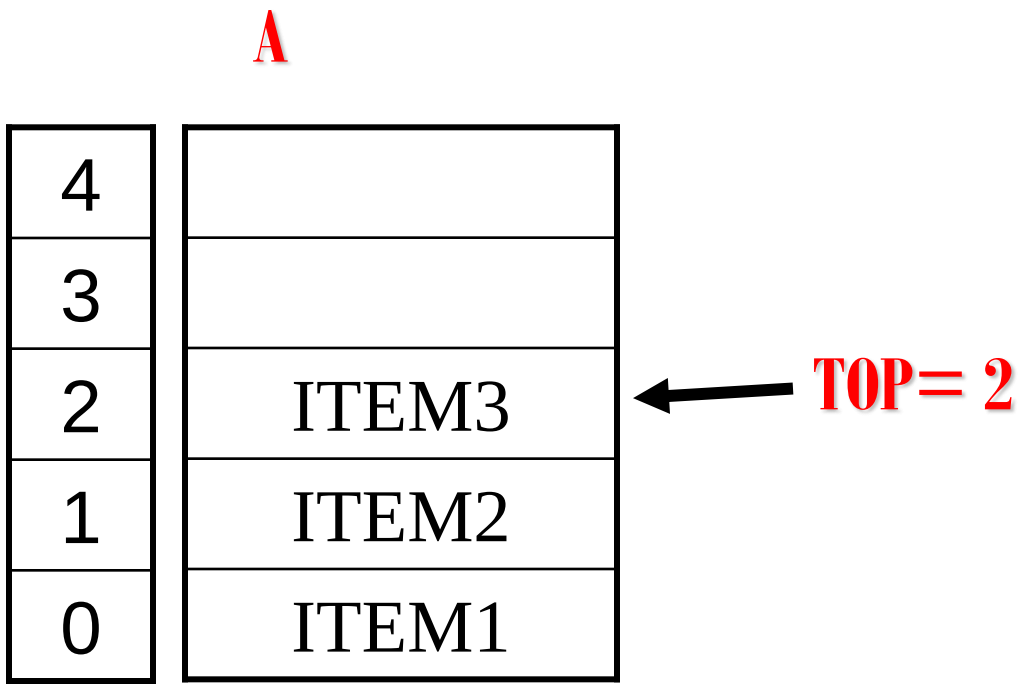
ITEM=A[TOP]

TOP=TOP-1

POP

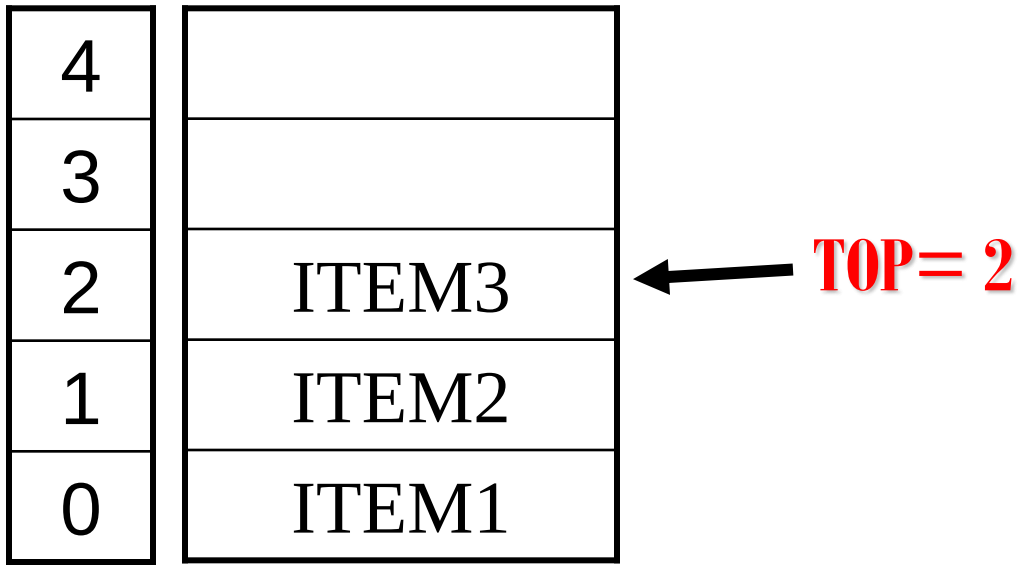
A




A
4
3
2
1
0
ITEM3
ITEM2
ITEM1
TOP= 2

POP

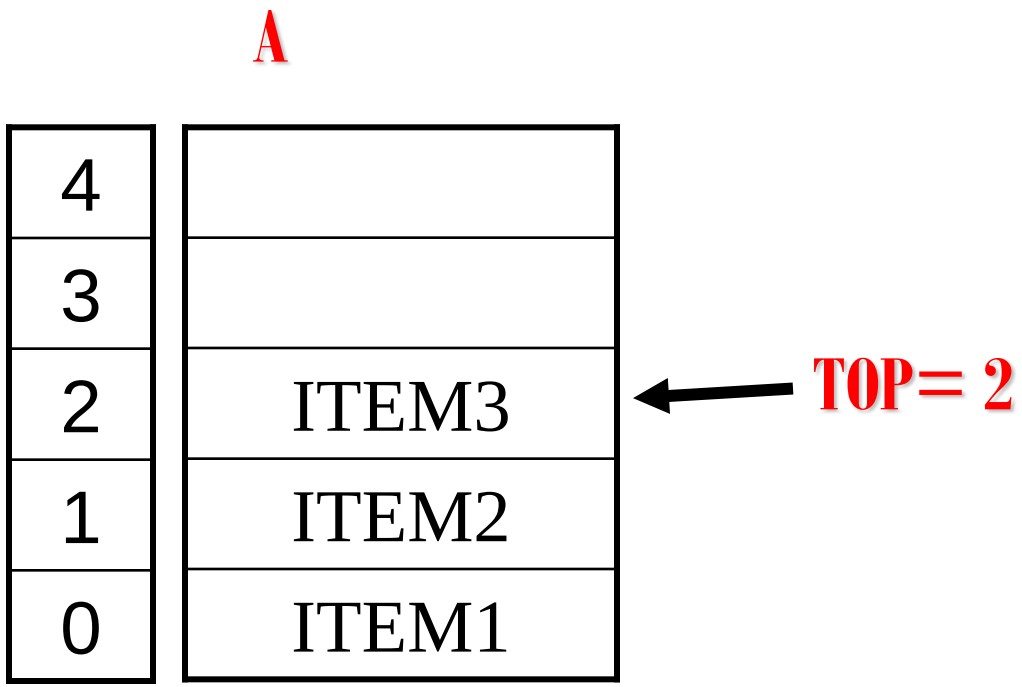
A

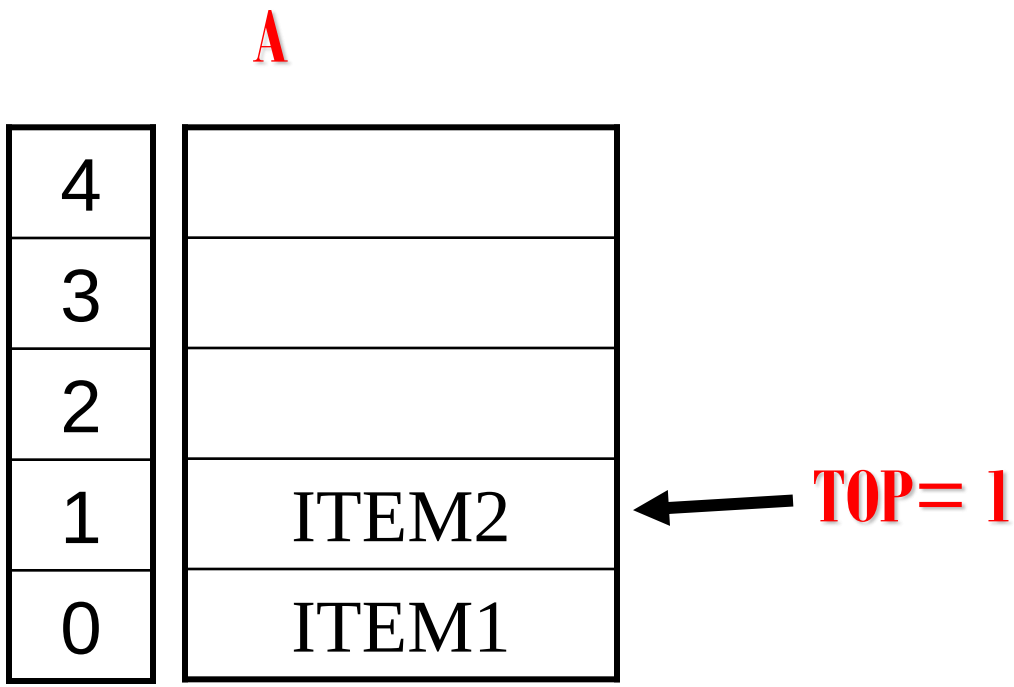


ITEM=A[TOP]

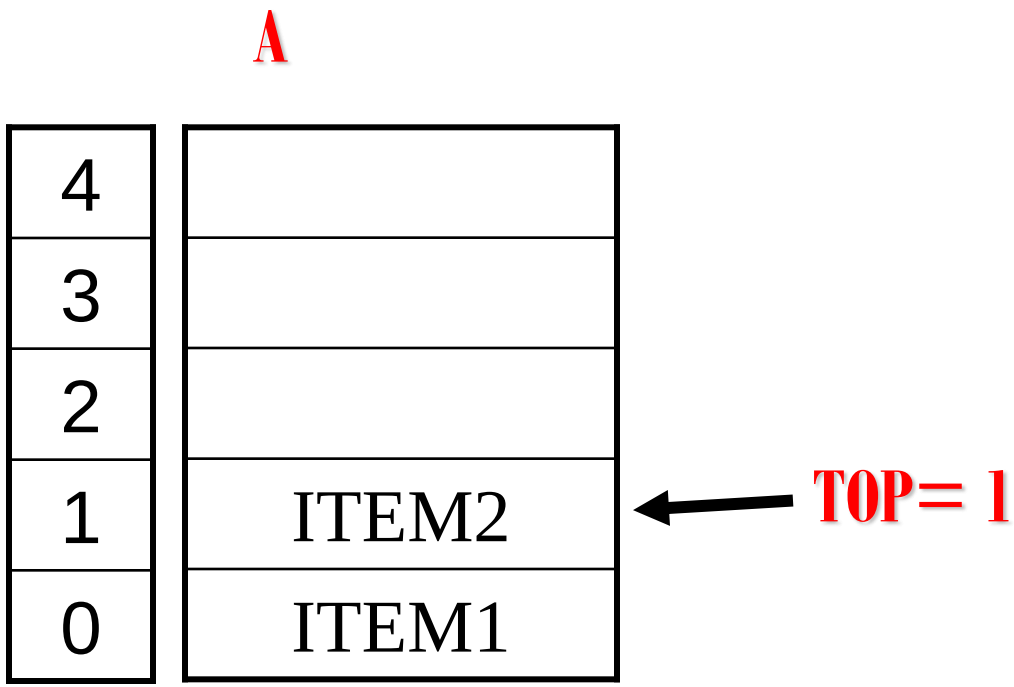
TOP=TOP-1

POP




A
4
3
2
1
0
ITEM2
ITEM1
TOP= 1

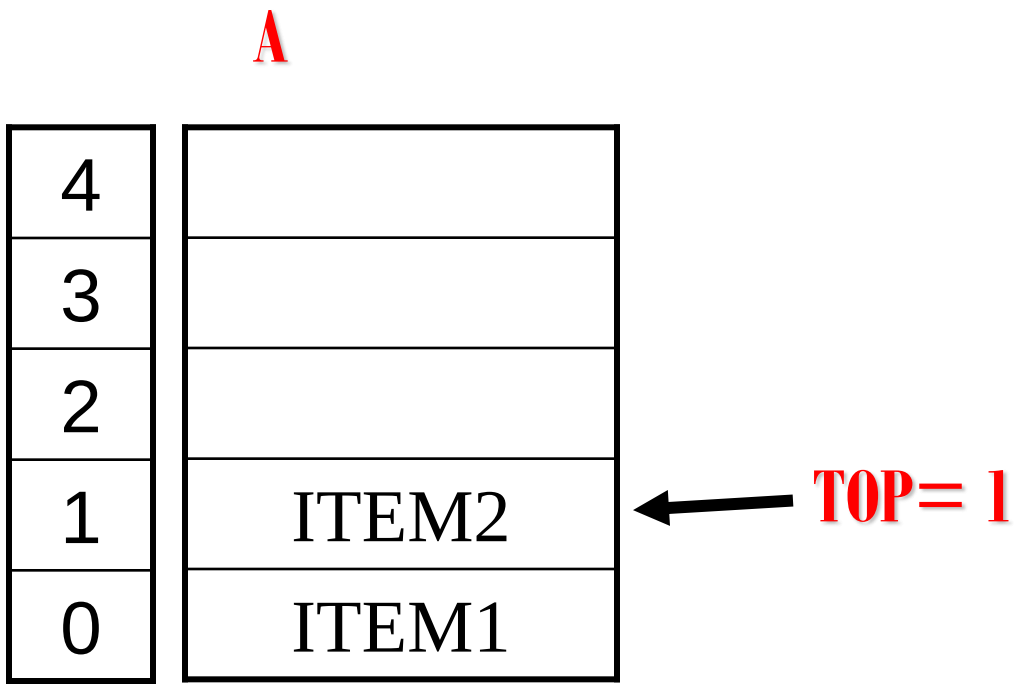
POP

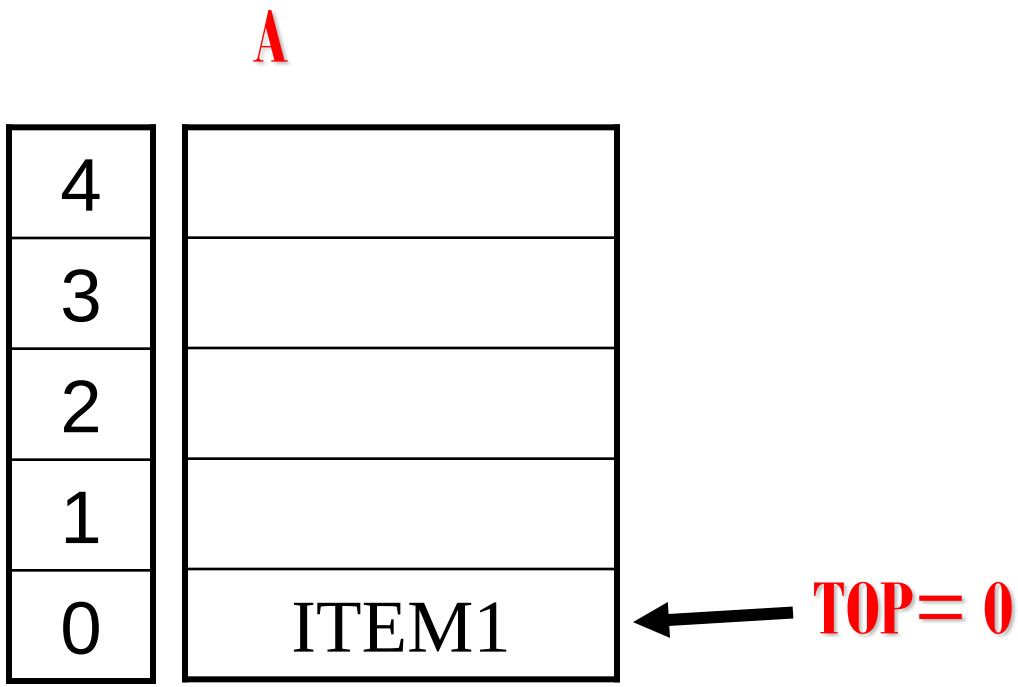


ITEM=A[TOP]

TOP=TOP-1

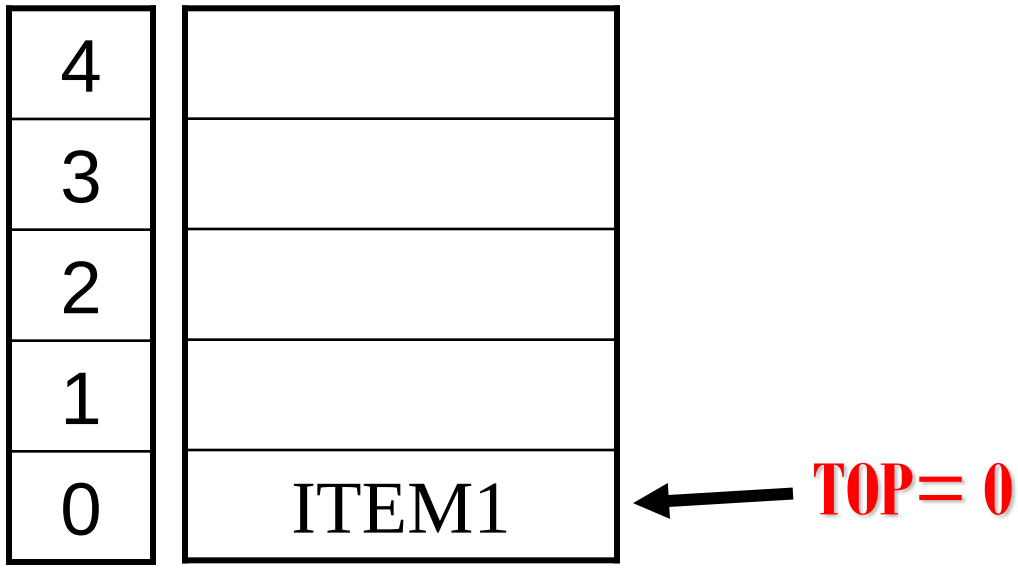
POP




A
4
3
2
1
0
ITEM1
TOP= 0

POP

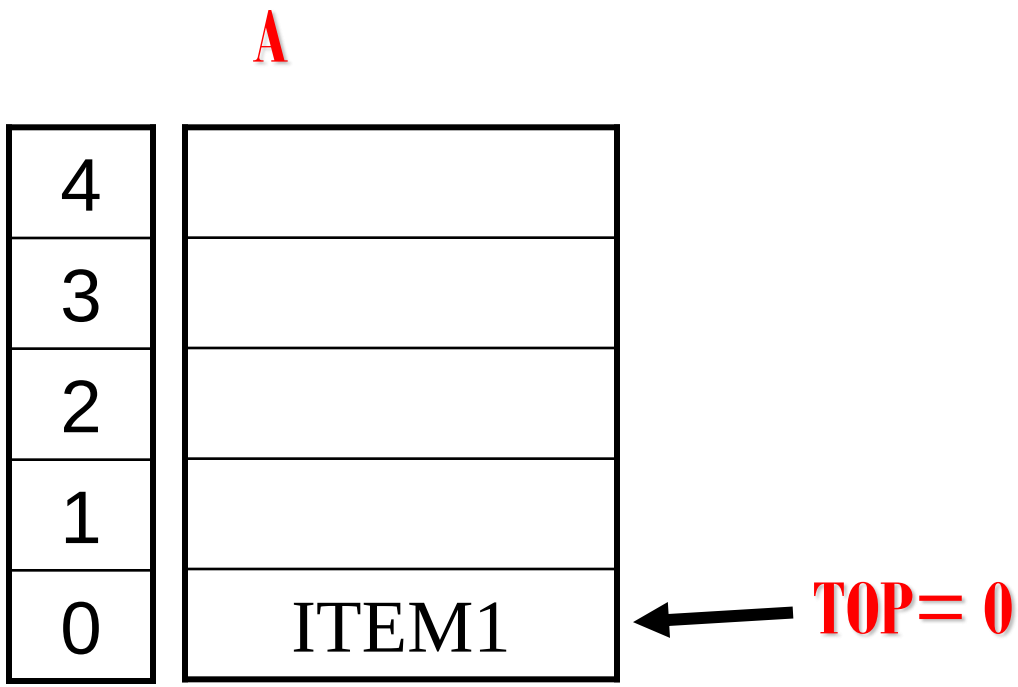
A

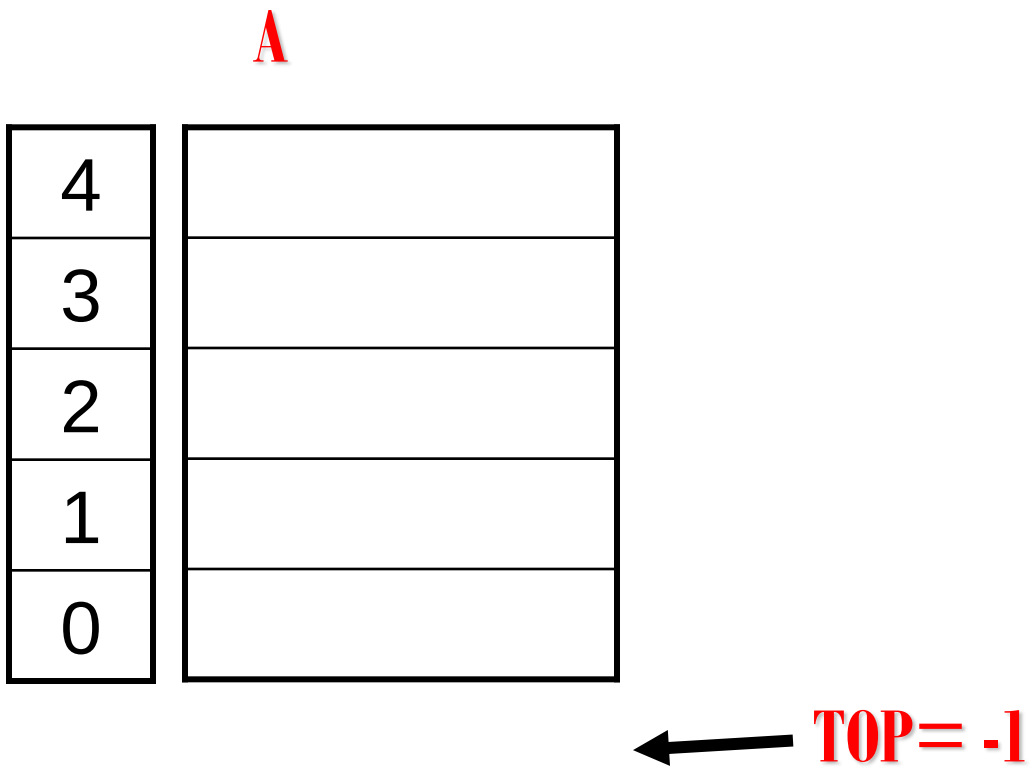


ITEM=A[TOP]

TOP=TOP-1

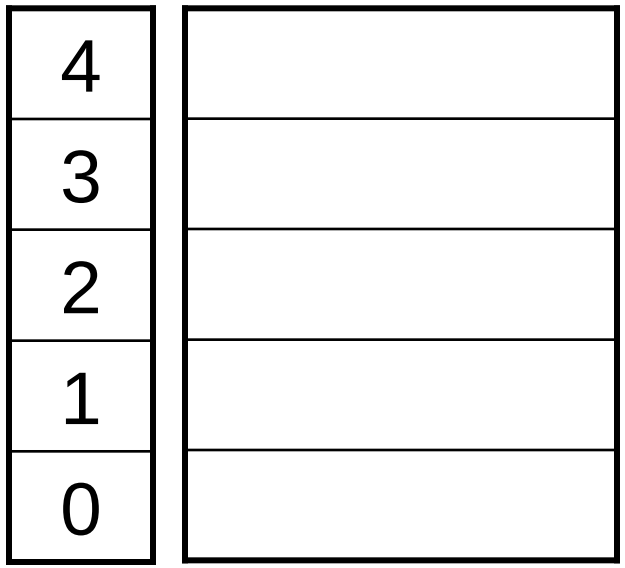
POP




A
4
3
2
1
0
TOP= -1

POP

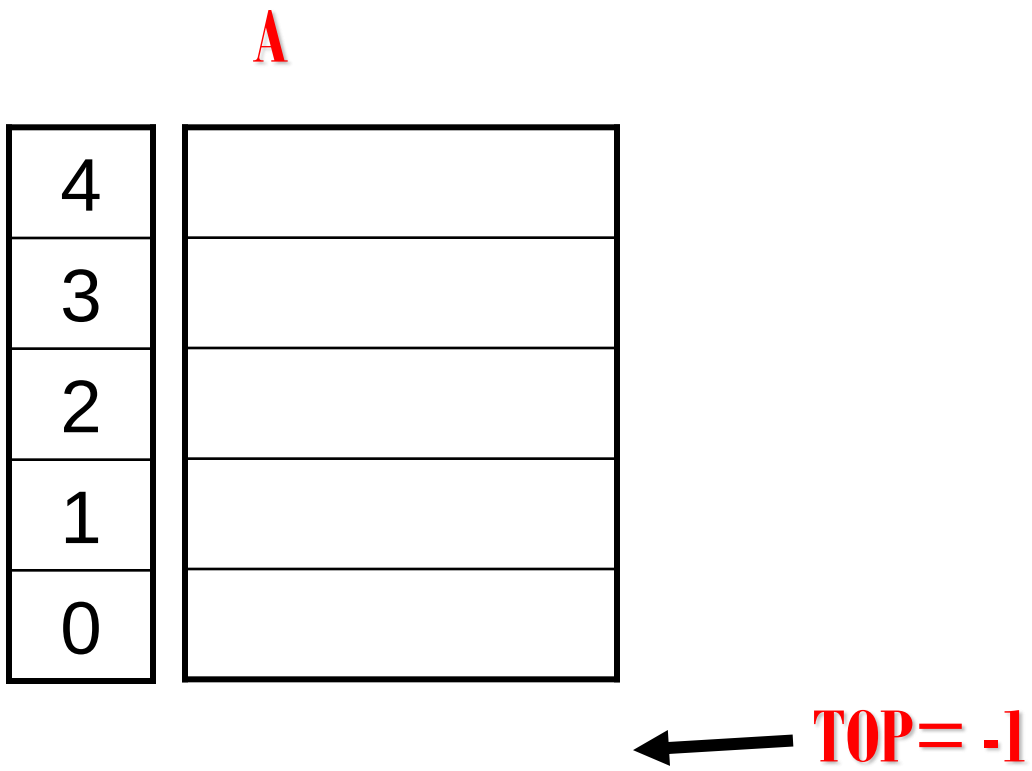
A



TOP= -1

If TOP==-1 then

Stack is empty

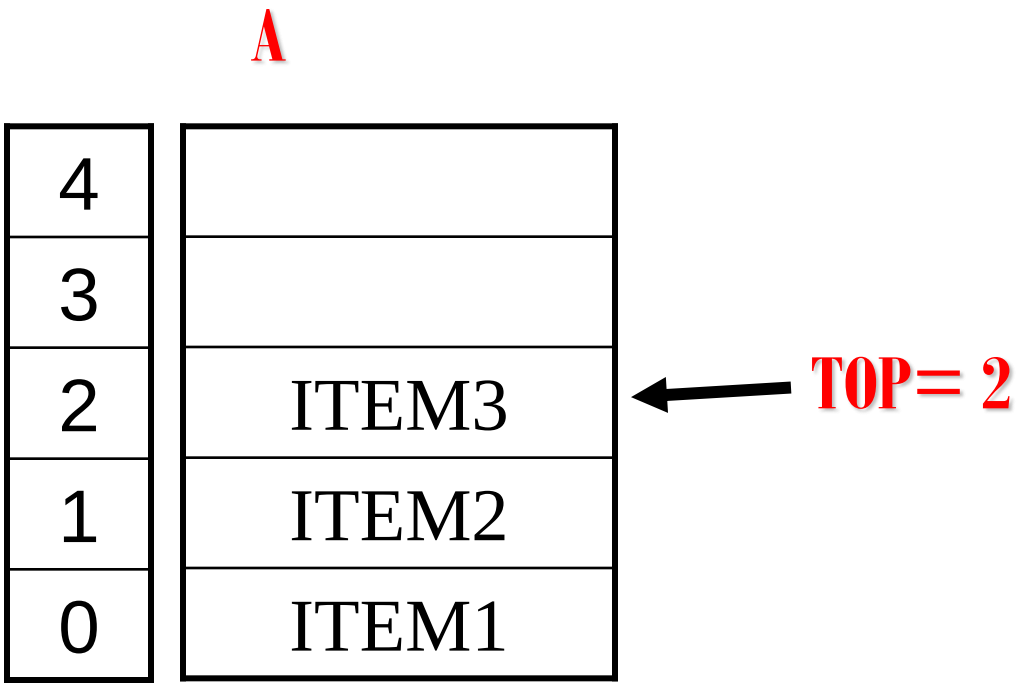


STACK – POP ALGORITHM

Algorithm POP()

```
{
        if  TOP = -1 then
                Print “Stack is empty”
        else
        {
                ITEM = A[TOP]
                TOP=TOP-1
                Print “Item to be popped is “ ITEM
        }
}
```

Display the contents of the stack from A[0] to A[TOP]



STACK – DISPLAY ALGORITHM

Algorithm DISPLAY()

```
{
        if  TOP = -1 then
                Print “Stack is empty”
        else
        {
                for i=0 to TOP do
                        Print A[i]
        }
}
```

STACK – STATUS ALGORITHM

Algorithm STATUS()

```
{       if  TOP = -1 then
                Print “Stack is empty”
        else
        {       Print “Stack top element is ” A[TOP]
                if TOP=SIZE-1 then
                        Print “Stack is full”
                else
                {       free=((SIZE-TOP-1)/SIZE)x100
                        Print “Free space is “ free
                }
        }
}
```

Implement Stack using Array

```
#include<stdio.h>
int A[100],size,top;
void push(int item)
{    if(top==size-1)
                printf("\nSTACK is full");
     else
     {          top++;
                A[top]=item;
     }
}
```

```
void pop()
{
    if(top==-1)
        printf("\nStack is empty");
    else
    {
        printf("\nThe popped elements is %d",A[top]);
        top--;
    }
}
```

```
void display()
{
        int i;
        if(top==-1)
                printf("\nThe STACK is empty");
        else
        {
                printf("\nThe elements in STACK \n");
                for(i=0; i<=top; i++)
                        printf("\n%d",A[i]);
        }
}
```

```
void status()
{       float free;
        if(top==-1)
                printf("Stack is empty");
        else
        {       printf("Stack top element is %d",A[top]);
                if(top==size-1)
                        printf("\nStack is full");
                else
                {       free=(float)(size-top-1)*100/size;
                        printf("\nFree space=%f%",free);
                }
        }
}
```

```
void main()
{
    int choice,n;
    top=-1;
    printf("\nEnter the size of STACK:");
    scanf("%d",&size);
    do
    {
        printf("\n1.PUSH\t2.POP\t3.DISPLAY\t
                                    4.STATUS\t5.EXIT");
        printf("\nEnter the Choice:");
        scanf("%d",&choice);
```

```
    switch(choice)
   {    case 1:          printf("Enter a value to be pushed:");
                         scanf("%d",&n);
                         push(n);        break;
        case 2:          pop();          break;
        case 3:          display();      break;
        case 4:          status();       break;
        case 5:          break;
        default:printf("\nPlease Enter a Valid Choice(1/2/3/4/5)");
    }

  }while(choice!=5);
}   //end of main()
```