Lec. 6 Arrays and Abstract Data Type in Data Structure:

ADTs → Abstract Data Types

ADT's are the way of classifying data structures by providing a minimal expected interface & set of Methods.

Minimal Requirements

Operations

MRF → Minimal Requirement Functionality Required.

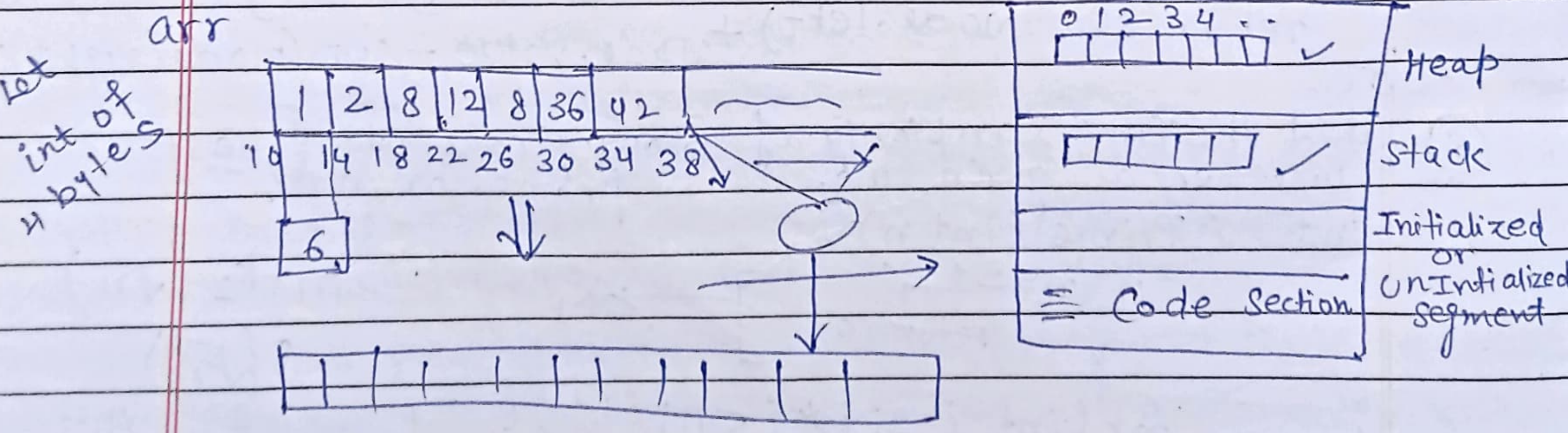
Arrays → get (i)
set (i, num)

Hiding the details

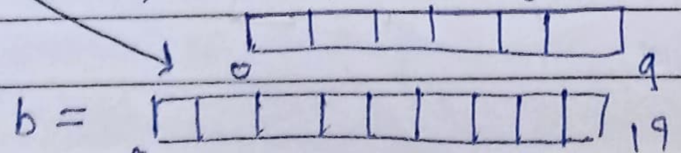
Abstraction
⇓
Implimentation ✗
Usage ✓

Methods/Operations
→ Insert
→ delete
→ Add
→ Resize.

Arrays → Collection of elements accessible by an index.



int *
a → (int *) malloc (10 *sizeof (int))



10+4×(i)

arr	0	1	2	3	4	5	6
	1	2	8	12	8	36	42
	10	14	18	22	26	30	34

Static and Dynamic Arrays:

Static Arrays → Size cannot be changed.
Dynamic Arrays → Size can be changed.

Lec.

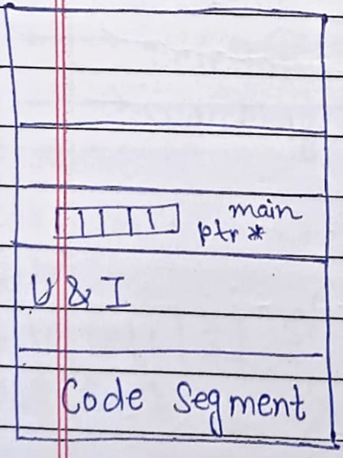
(7) Array as An Abstract Data Type in Data Structure:

ADT's
→ Set of values + Set of operations

int → 9, 10, 12

$\boxed{9+12=21}$
↓
operator
↓
Details abstracted

myArray → total_size
used_size
base address
+ max()
get(i)
set(i, num)
add(arr)
→ Set of operations.

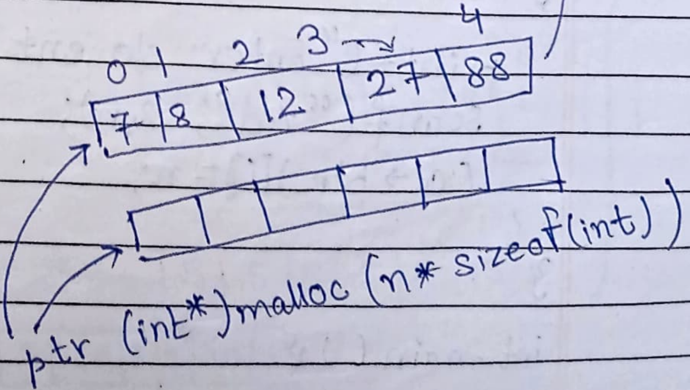


Memory

heap → Dynamic memory

stack → static memory

Initialized & Uninitialized Segment


0
1
2
3
4
7
8
12
27
88
cannot extend using realloc
ptr (int*)malloc (n* sizeof(int))

Lec 8 Implementing Array as an Abstract Data Type in C

```
#include <stdio.h>  → #include<stdlib.h>
struct myArray
{
    int total_size;
    int used_size;
    int *ptr;
};
void createArray(struct myArray * a, int tsize, int usize)
{

    // (*a).total_size = tsize;
    // (* a).used_size = usize;
    // (*a).ptr = (int *) malloc (tsize * sizeof(int));

    a-> total_size = tsize;
    a-> used_size = usize;
    a-> ptr = (int *) malloc (tsize * sizeof(int));
}
void show(struct myArray *a){
    for(int i=0; i< a-> used_size; i++)
    {
        printf("%d \n", (a->ptr)[i]);
    }
}
void SetValue(struct myArray * a){
    int n;
    for(int i=0; i< a->used_size; i++)
    {
        printf("Enter element %d", i);
        scanf("%d", &n);
        (a-> ptr)[i] = n;
    }
}
int main(){
```

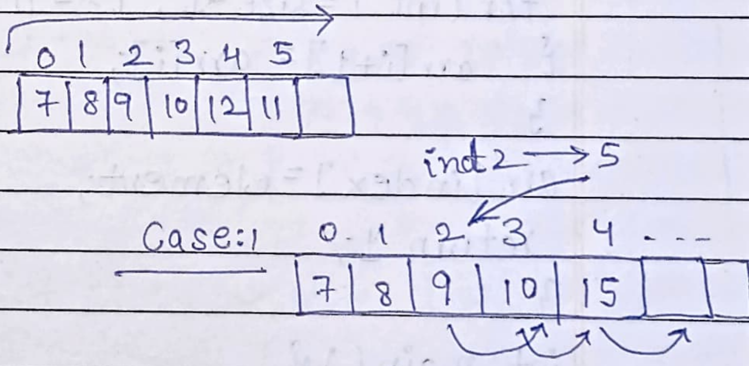
```
{
struct myArray marks;
createArray(& marks, 10, 2);
printf("we are running Setval Now \n");
setVal(& marks);

printf("we are running show now \n");
Show(& marks);
return 0;
}
```

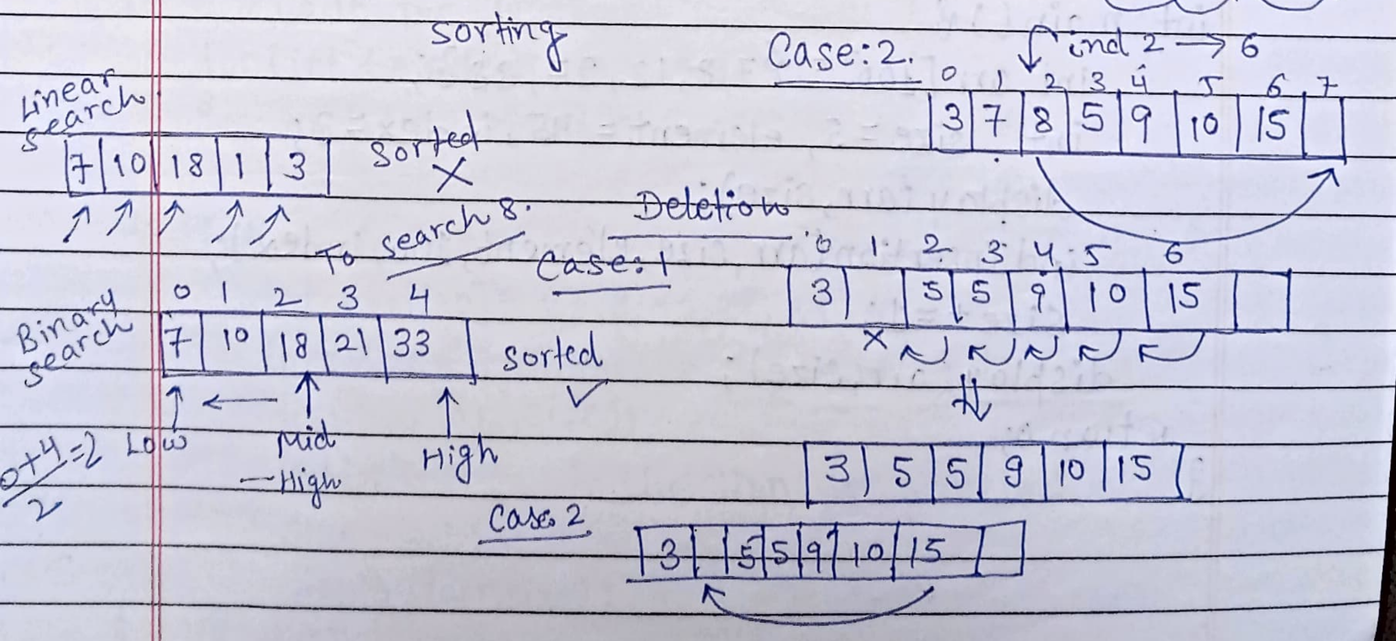
```
We are running Setval Now
Enter element 0  3
 Enter Element 1  4
We are running Show Now
 3
 4
```

Lec.
(9)

Operations on Arrays in Data Structures : Traversal, Insertion Deletion and Searching :

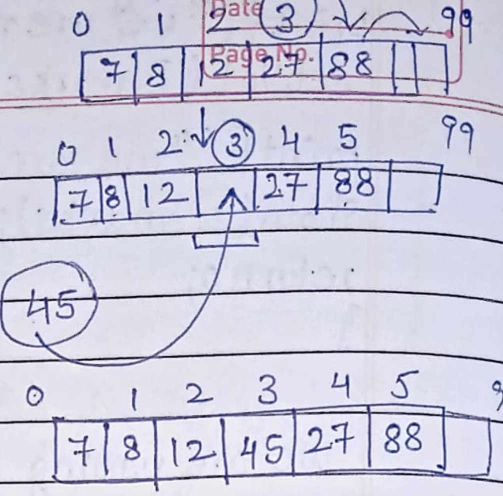
Operations on Arrays:
→ Traversal → visit
→ Insertion →
→ Deletion
→ Searching





Lec(10) Coding Insertion Operation in Array in Data structure.

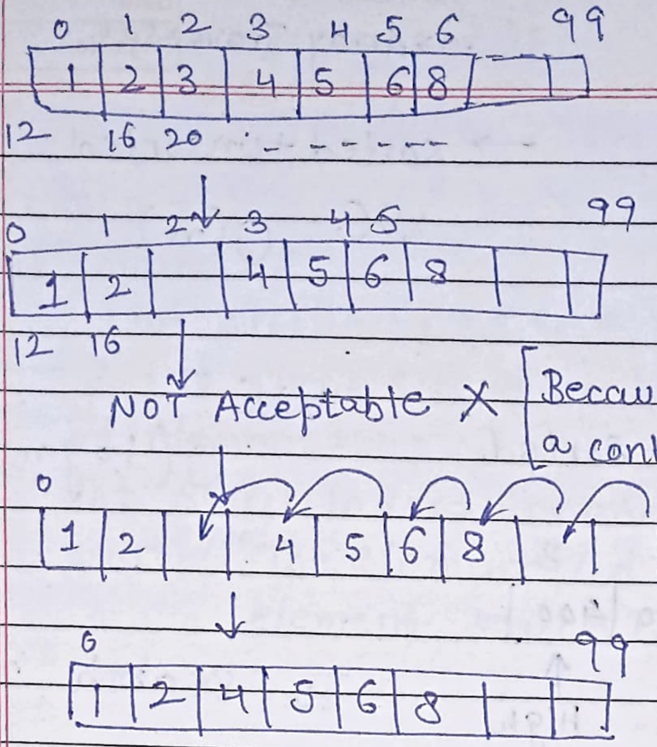
Insertion.c



```
#include<stdio.h>
void display (int arr[], int n){
    //Code for Traversal
    for (int i=0; i<n; i++)
    {
        printf ("%d", arr[i]);
    }
    printf ("\n");
}
int indInsertion(int arr[], int size, int element, int capacity,
                                                   int index)
{
    //Code for Insertion
    if(size>= Capacity){
         return -1;
    }
  for(int i=size-1; i>= index; i--)
  {  arr[i+1]= arr[i];
  }
 arr [index]= element;
  return 1;
}
int main(){
      int arr [100]= {7,8,12,27,88};
      int size=5, element= 45, index=3;
      display (arr, size);
      indInsertion(arr, size, element, 100, index);
      size+=1;
    display(arr, size);
 return 0;
}
```

```
7 8 12 27 88
7 8 12 45 27 88
```

Lec-11. Coding Deletion Operation in Array Using C Language:



[Because Array is a continuous blocks of memory]

After Deletion.

```
# include <stdio.h>
  void  display ( int arr [ ], int n) {
          // Code for Traversal
     for ( int i=0; i<n; i++)
     {     printf(“%d”, arr[i]);
     }
     printf(“\n”);
  }
void indDeletion ( int arr[], int size, int index )
  {     // Code for Deletion
     for (int i=index; i<size-1; i++)
     {     arr[i] = arr[i+1];
     }
  }
     int main () {
          int arr[100]= {7, 8, 12, 27, 88};
          int size= 5;~~element=45~~, index=1;
          display (arr, size);
          ~~indDeletion=~~   indDeletion(arr, size, index);
          size -=1;
          display (arr, size);
  }  return 0;
```

```
7 8 12 27 88
7 12 27 88
```