-BINARY SEARCH-

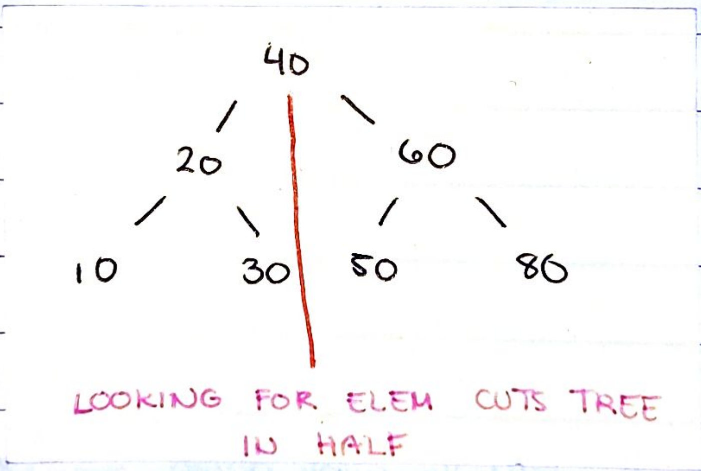
* Binary search

- searching algorithm used in a SORTED ARRAY by repeatedly dividing the search interval in HALF.

- $O(\log_2 N)$

* Conditions

1. The data structure must be sorted.
2. Access to any element of the data structure takes constant time.



LOOKING FOR ELEM CUTS TREE IN HALF

*When to Use?

1. When searching a large dataset
2. Dataset is sorted.
3. Data is stored in contiguous memory.
4. Data does not have a complex structure or relationships.

* Implementations

> Iterative Binary Search Algorithm

> Recursive Binary Search Algorithm

* Steps

1. Set the LOW INDEX / LOWER BOUND to the 1st element of the array and HIGH INDEX/ UPPER BOUND to the last element (count-1);

2. Set MIDDLE INDEX to the average of the LOW & HIGH.

$$\boxed{``\, mid = \frac{LB + UB}{2} \,"}$$

- If element at MID == x, return MID / TRUE;
- Otherwise, using value of MID, decide next search space:

$mid < x$	$mid > x$
$LB = mid + 1$	$UB = mid - 1$

3. Repeat step 2 until elem is found OR search space is exhausted.

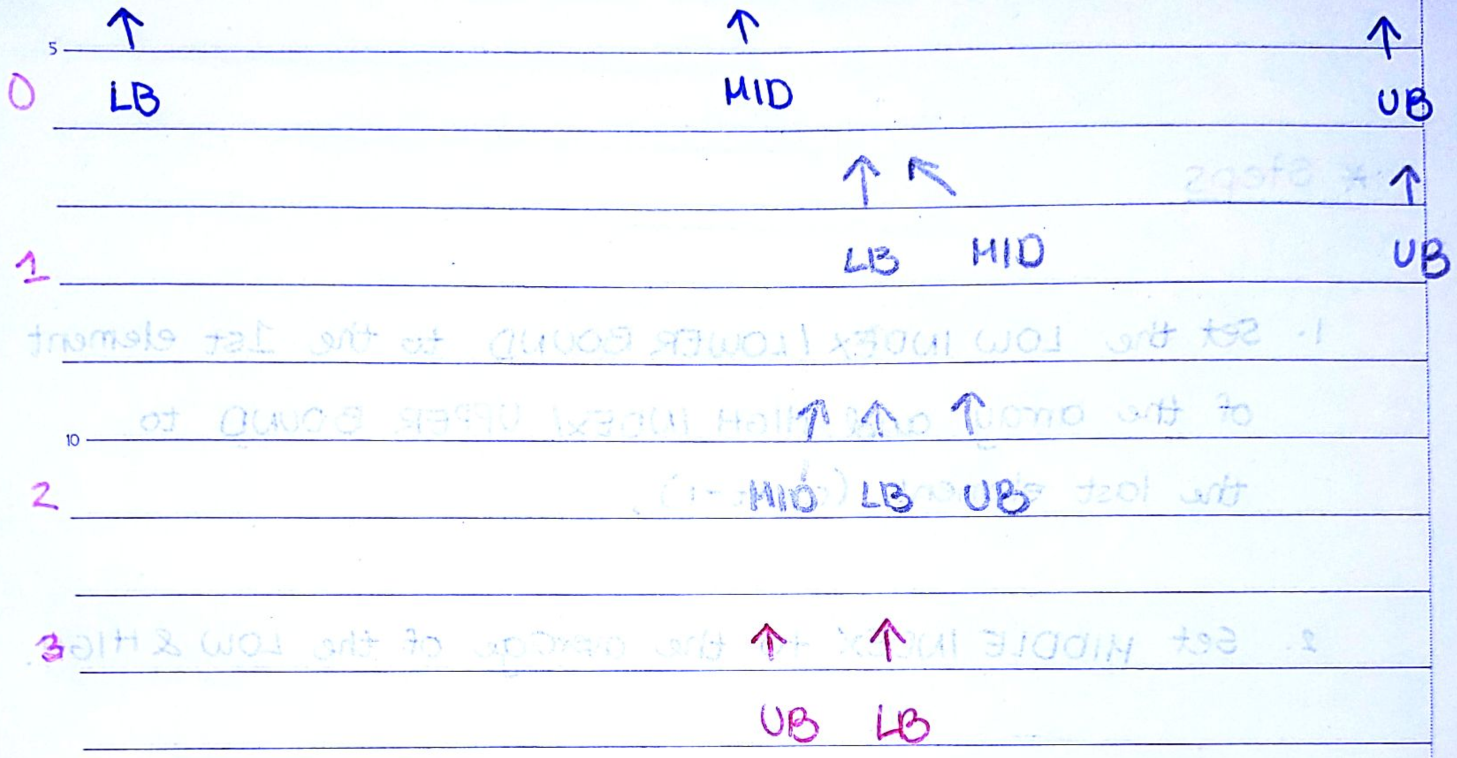
DATE:

Example

ELEM NOT FOUND

Look for Elem: 70

0	1	2	3	4	5	6	7	8	9	10	11
10	20	30	40	50	60	80	90	100	105	110	120



X = 70

Loop	LB	UB	MID	Elem	x&elem
1	0	11	5	60	70>60
2	6	11	8	100	70<100
3	6	7	6	80	70<80
4	6	5	5		

LB > UB = STOP

ELEM NOT FOUND

① set LB, UB, mid;

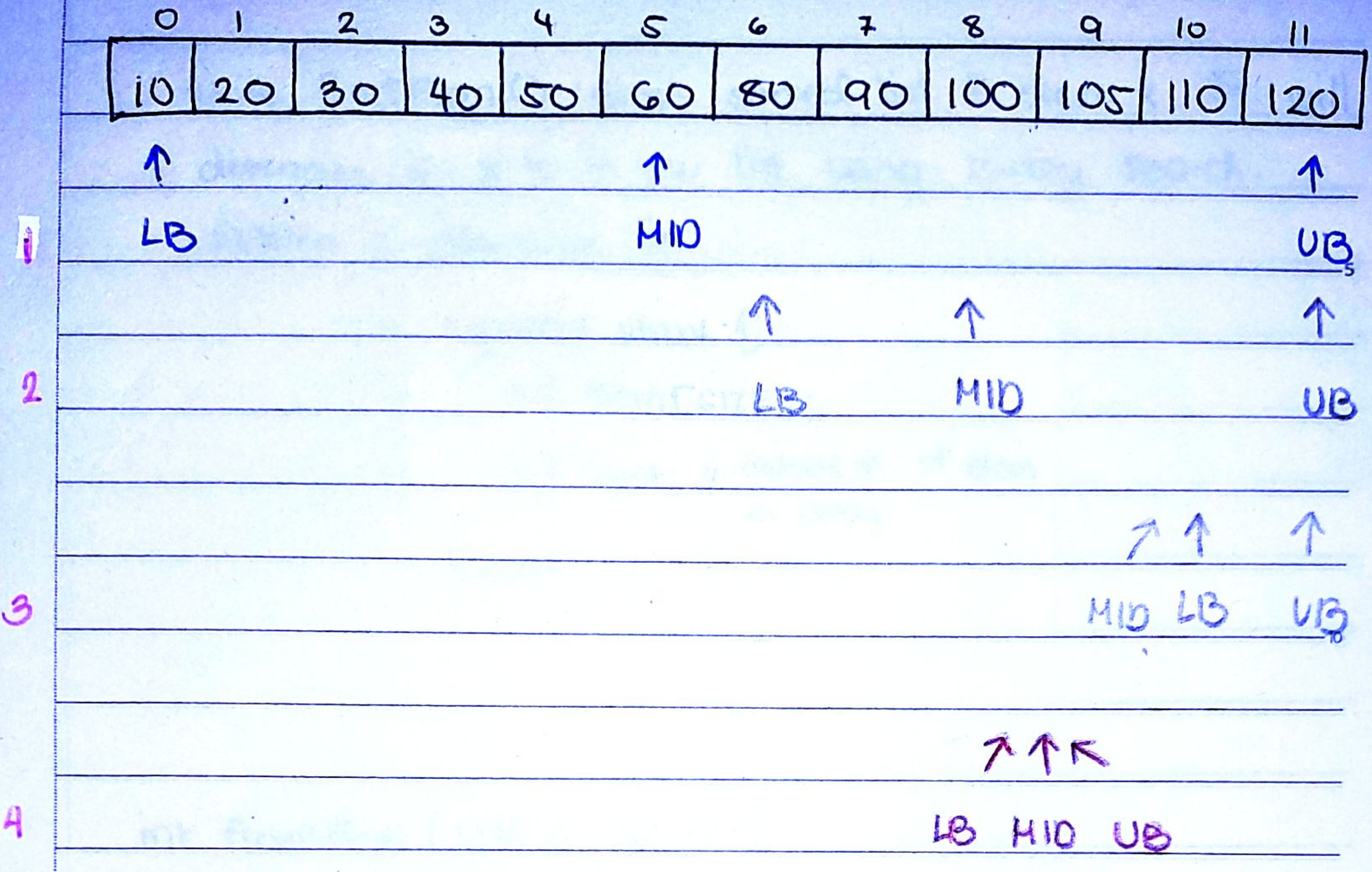
* LB will be 0;
* UB depends on size of array.
* mid = (UB+LB)/2

② Loop thru list & check if mid == X.

> Adjust LB & UB accordingly.

③ stop if element is found OR LB > UB.

Example ELEM FOUND Look for Elem: 105



0	1	2	3	4	5	6	7	8	9	10	11
10	20	30	40	50	60	80	90	100	105	110	120

x= 105

Loop	LB	UB	MID	Elem	x & elem
1	0	11	5	60	105>60
2	6	11	8	100	105>100
3	9	11	10	110	105<110
4	9	9	9	105	105 =105

LB = UB = MID = x

ELEM IS FOUND

// Binary search ()

Write findElem(). Given sorted list & elem x. Fn. will determine if x is in the list using Binary search. Return 1, otherwise 0.

```
typedef struct {
    int Elem[SIZE];
    int count; // actual # of elem
                  in array
} LIST;
```

```
int findElem (LIST L, int x) {
    int LB = 0, UB = count -1, mid;

    while ( LB <= UB && L.Elem[mid = ((LB+UB) /2))] != x){
        (( x > L.Elem[mid] < x)? (LB = mid+1) :
                                  (UB = mid-1);
    }

    return (L.Elem[mid] != x)? 0:1;
}
```

* can also just remove Mid variable altogether & just replace with (LB+UB)/2. Mid variable for easier readability.