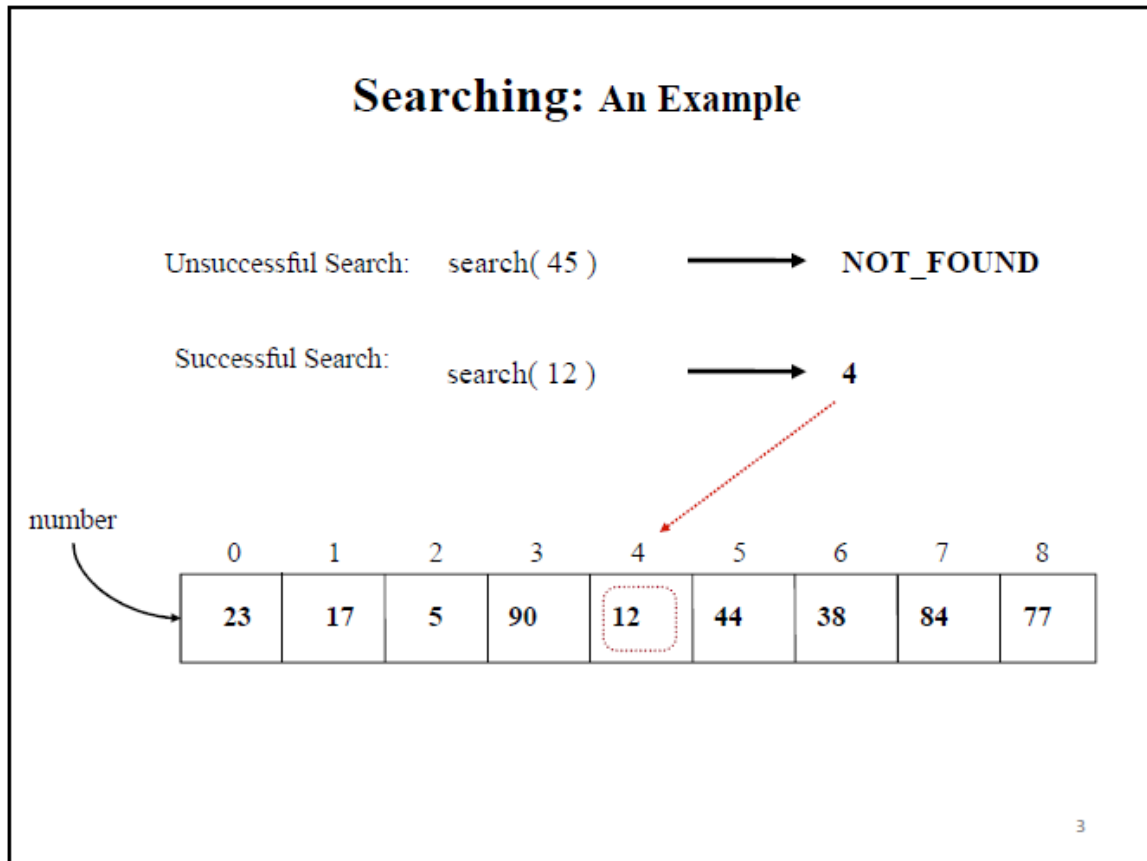


البحث Searching

المحاضرة العاشرة Data structure

- **Searching** is the process of finding a target element among a group of items (the search pool), or determining that it isn't there.



1-Linear Search:

- **Linear Search:** A linear search simply examines each item in the search pool, one at a time, until either the target is found or until the pool is exhausted.
- This approach does not assume the items in the search pool are in any particular order
- We just need to be able to examine each element in turn (in a linear fashion)
- It's fairly easy to understand, but not very efficient

Linear Search: Performance

- We analyze the successful and unsuccessful searches separately.
- We count how many times the search value is compared against the array elements.
- **Successful Search**
 - **Best Case :** 1 comparison
 - **Worst Case:** N comparisons (N – array size)

- **Unsuccessful Search**

- **Best Case:**

- **Worst Case:** N comparisons

```
#include <iostream>

using namespace std;
int main()
{
    int arr[10];
    int no_of_elements, key;
    bool found = false;
    cout << "Enter the number of element: ";
    cin >> no_of_elements;
    for (int i = 0; i < no_of_elements; i++) {
        cout << "arr[" << i << "]: ";
        cin >> arr[i];
    }
    cout << "Enter the value to search: ";
    cin >> key;
    for (int i = 0; i < no_of_elements; i++) {
        if (key == arr[i]) {
            found = true;
            cout << "The value is found at index arr[" << i << "];";
        }
    }
    if (!found) {
        cout << "Key not found!";
    }
    return 0;
}
```

2-Binary Search:

في خوارزمية البحث الثنائي يجب ان تكون المصفوفة مرتبة تصاعديا قبل عملية البحث

1-نحدد العنصر الذي في وسط المصفوفة middle

2-اذا كان العنصر الذي نبحت عنه موجود في الوسط ,اذا تتوقف الخوارزمية.

3-اذا كان العنصر الذي نبحت عنه اصغر من ال middle نذهب الى الجهة اليسرى من ال middle.

4-واذا كان العنصر الذي نبحت عنه اكبر من ال middle نذهب الى الجهة اليمنى.

- If the search pool is sorted, then we can be more efficient than a linear search
- A binary search eliminates large parts of the search pool with each comparison
- Instead of starting the search at one end, we begin in the middle

Binary Search: An Example

- If the array is sorted, then we can apply the binary search technique.

number

0	1	2	3	4	5	6	7	8
5	12	17	23	38	44	77	84	90

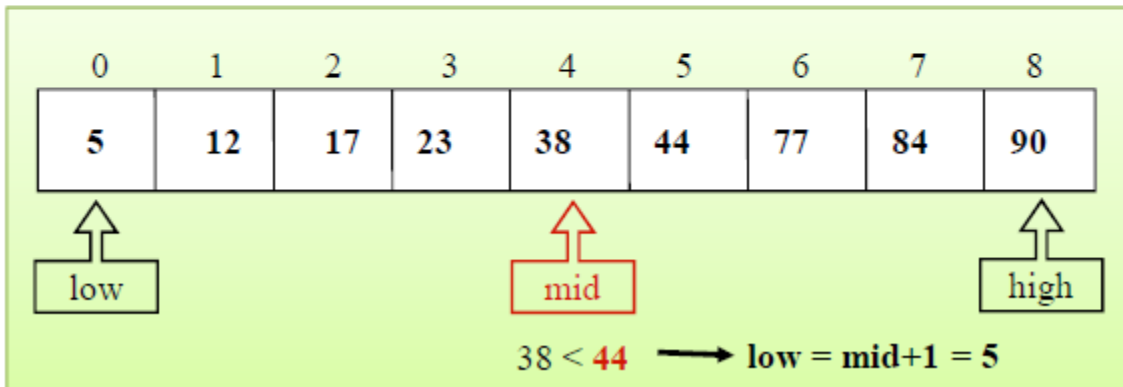
- The basic idea is straightforward. First search the value in the middle position. If X is less than this value, then search the middle of the left half next. If X is greater than this value, then search the middle of the right half next. Continue in this manner

Binary Search: An Example

	low	high	mid
#1	0	8	4

search(44)

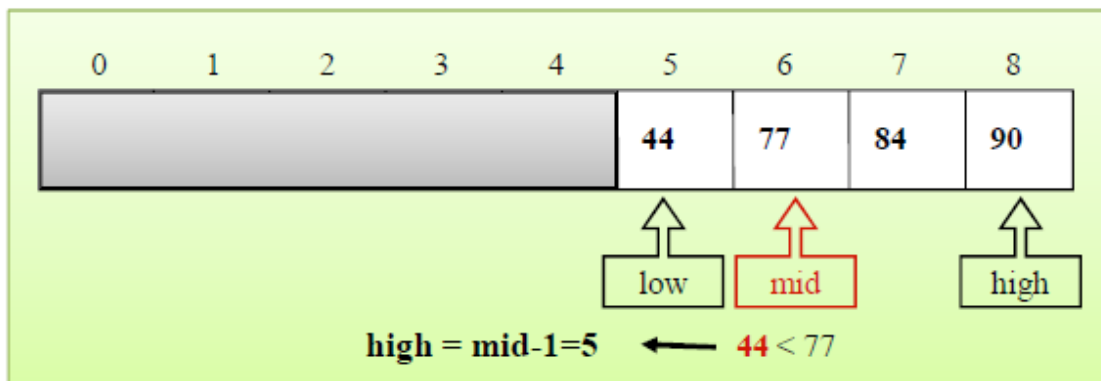
$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$



Binary Search: An Example

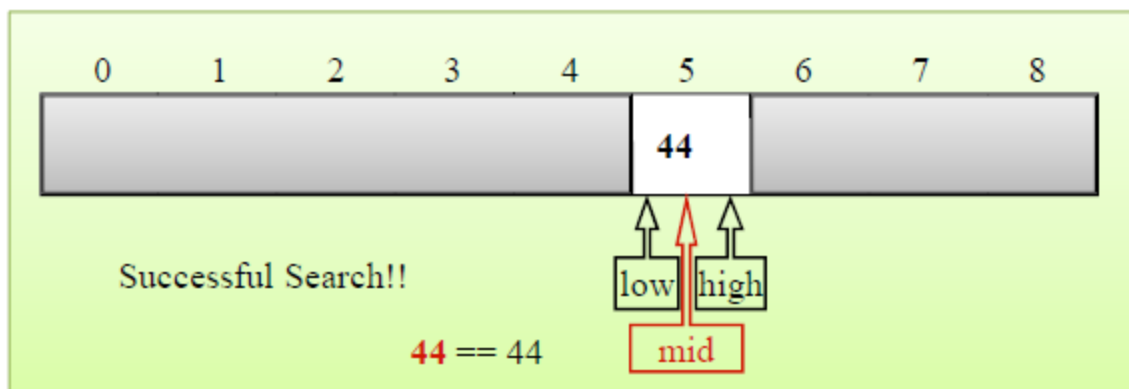
	low	high	mid
#1	0	8	4
#2	5	8	6

search(44)

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$


Binary Search: An Example

	low	high	mid	
#1	0	8	4	search(44) $mid = \left\lfloor \frac{low + high}{2} \right\rfloor$
#2	5	8	6	
#3	5	5	5	



Example:

If searching for 23 in the 10-element array:

	2	5	8	12	16	23	38	56	72	91
23 > 16, take 2 nd half	L									H
	2	5	8	12	16	23	38	56	72	91
23 < 56, take 1 st half										
	2	5	8	12	16	23	38	56	72	91
Found 23, Return 5										
	2	5	8	12	16	23	38	56	72	91

```

#include<iostream>
using namespace std;

int binarysearch(int [],int ,int ) ;

int main()
{
    int n, y, a[50], key;

    cout<<"Enter total number of elements :";
    cin>>n;
    cout<<"Enter "<< n <<" number :";
    for (int i=0; i<n; i++)
    {
        cin>>a[i];
    }
    cout<<"Enter a number to find :";
    cin>>key;

    y=binarysearch(a,n,key);

    if(y== -1)
        cout<<"Not found! "<< key <<" is not present in the list.";
    else
        cout<< key <<" found at location "<<y+1<<"\n";

    return 0;
}

/*-----*/

int binarysearch(int arr[],int n,int key)
{ int first,last,middle;

    first = 0;
    last = n-1;
    middle = (first+last)/2;
    while (first <= last)
    {
        if(arr[middle] < key)
        {
            first = middle + 1;
        }
        else if(arr[middle] == key)
        {
            return middle;
        }
        else
        {
            last = middle - 1;
        }
        middle = (first + last)/2;
    }
    return -1;
}

```

}

H.W: suppose you have the following sorted array {3,5,6,8,11,12,14,15,17}, using the recursive binary search algorithm ,which group of numbers correctly show the sequins of comparison used to find the key 8 ?