Day 5

Searching

```
- It is a process of finding location( index / reference ) of an element
inside collection.
```

Searching algorithms:

```
1. Linear Search algorithm
2. Binary Search algorithm
3. Hashing
```

Linear Search algorithm

```
- Linear search algorithm is also called as sequential search algorithm.
- We can use linear search algorithm to search element inside any sorted
as well as unsorted collection.
- If number of elements in a collection are less then it is considered as
efficient searching algorithm.
- Consider Linear search on array
```

```
class Program{
    public static int search( int[] arr, int key ){
        if( arr != null ){
            for( int index = 0; index < arr.length; ++ index ){
                if( key == arr[ index ] )
                    return index;
            }
        }
        return -1;
    }
    public static void main( String[] args ){
        int key = 30;
        int[] arr = new int[ ]{ 10, 20, 30, 40, 50 }
        int index = Program.search( arr, key );
        if( index != -1)
            System.out.println(key+" found at index "+index);
        else
            System.out.println(key+" not found");
    }
}
```

```
- Consider Linear search on LinkedList
```

```
public Node find( int data ){
    Node trav = this.head();
    while( trav != null ){
        if( data == trav.data )
            return trav;
        trav = trav.next;
    }
    return null;
}
```

```
arr[ 0 ]    =>  10;     //no of comparisons required to search 10 are : 1
arr[ 1 ]    =>  20;     //no of comparisons required to search 10 are : 2
arr[ 2 ]    =>  30;     //no of comparisons required to search 10 are : 3
arr[ 3 ]    =>  40;     //no of comparisons required to search 10 are : 4
arr[ 4 ]    =>  50;     //no of comparisons required to search 10 are : 5
```

```
- Time required to search everly element is different.
```

Binary Search algorithm

```
- Divide and Conquer is algorithm design technique.
- Binary Search algorithm is based on divide and Conquer technique.
- If we want to reduce no of comparisons required to search element then
we should use binary search algorithm.
- If we want to use binary search algorithm then underlying collection
must be sorted.
- Consider following example
```

```
-------------------------------------------------------------

left                                mid                     right <= Index
variables
-------------------------------------------------------------
0           1           2           3        4        5      6       <=
Array Index
-------------------------------------------------------------
10,         20,         30,         40,     50,       60,    70      <=
```

```
arr
-------------------------------------------------------------
arr[0]      arr[1]      arr[2]      arr[3]  arr[4]  arr[5]  arr[6]
-------------------------------------------------------------
```

- Let us search 40

```
int left = 0;
int right = 6
int mid = ( left + right ) / 2;
        = ( 0 + 6 ) / 2;
        = 3
arr[ mid ]  =>  40
- Only one comparison is required to search 40.
```

- Let us search 10

```
int left = 0;
int right = 6
int mid = ( left + right ) / 2;
        = ( 0 + 6 ) / 2
        = 3
- key is smaller than arr[ mid ] => 40;
- Let us change value of right;
left = 0;
right = 2
mid = ( 0 + 2 ) / 2;
    = 1
- key is smaller than arr[ mid ] => 20;
- Let us change value of right;
left = 0;
right = 0
mid = ( 0 + 0 ) / 2;
    = 0
```

- Binary search algorithm reduces number of comparisons but time required to search every element is different.
- Let us search 60

```
int left = 0;
int right = 6;
int mid = ( 0 + 6 ) / 2;    =>  3
arr[ mid ] is 40 but key is 60;
- Let is change left
left = 4;
right = 6;
```

```
int mid = ( 4 + 6 ) / 2;    =>  5
arr[ mid ] is 60.
```

Implementation

```
class Program{
    public static int binarySearch( int[] arr, int key ){
        int left = 0;
        int right = arr.length - 1;
        while( left <= right ){
            int mid = ( left + right ) / 2 ;
            if( key == arr[ mid ])
                return mid;
            else if( key > arr[ mid ] )
                left = mid + 1;
            else
                right = mid - 1;
        }
        return -1;
    }
    public static void main( String[] args ){
        int key = 40;
        int[] arr = new int[ ]{ 10, 20, 30, 40, 50, 60, 70 };
        int index = Program.binarySearch( arr, key );
        if( index != -1)
            System.out.println(key+" found at index "+index);
        else
            System.out.println(key+" not found");
    }
}
```

- recursive binarySearch algorithm

```
public static int binarySearch( int[] arr, int left, int right, int key ){
    if( left > right )
        return -1;
    int mid = ( left + right ) / 2;
    if( key == arr[ mid ] )
        return mid;
    else if( key > arr[ mid ] )
        return binarySearch(arr, mid + 1, right, key);
    else
        return binarySearch(arr, left, mid - 1, key);
}
```

Hashing

- Using linear search and binary search algorithm, we can not search element in constant time.

- If we want to search element in constant time then we should use hashing technique.
- Hashing algorithm is based on hashcode.
- Hashcode is logical integer number that we can generate by processing state of the object.
- To generate hashcode, we should define a method. It is called hash function/method.
- Consider example of hash method

```
public static int getHashCode( int element ){
    final int PRIME = 31;
    int result = 1;
    result = result * PRIME * PRIME + element;
    return result;
}
public static void main( String[] args ){
    int x = 10;
    System.out.println( Program.getHashCode(x));    //971

    int y = 10;
    System.out.println( Program.getHashCode(y));    //971
}
```

- If state of instances are same then we will get same hashcode.

```
public static void main( String[] args ){
    int x = 10;
    System.out.println( Program.getHashCode(x));    //971
    int y = 15;
    System.out.println( Program.getHashCode(y));    //976
}
```

- If state of instances are different then we will get different hashcode.
- hashcode is not a address/refereference/index rather it is logical number that we can generate by processing variable/instance/object.
- Hashcode is used to find out slot/index.

```
public static final int SIZE = 7;   //Array size
public static void main( String[] args ){
    int x = 10;
    int hashCode = Program.getHashCode( x );    //971
    int slot = hashCode % SIZE; //5

    int y = 15;
    hashCode = Program.getHashCode( x );    //976
    slot = hashCode % SIZE; //3
}
```

```
public static void main( String[] args ){
    int x = 10;
    int hashCode = Program.getHashCode( x );    //971
    int slot = hashCode % SIZE; //5

    int y = 10;
    hashCode = Program.getHashCode( y );    //971
    slot = hashCode % SIZE; //5
}
```