**Searching**

In the syllabus:

1. Sequential Searching
2. Binary Searching

**Sequential Searching**

Here, we search is linearly. Suppose, the element to search is “key”. We first compare the “key” with arr[0] if we find the match, then index position 0 is returned. Otherwise, we compare the key with arr[1], if we find the match, then 1 is returned. Otherwise, this process will go on until we find a match.

If we don’t find a match even if we traverse to the last element of the array, then -1 is returned.  
 **The program:**

**import java.io.\*;**

**class Sequentialsearch**

**{**

**public static int sequential\_search(int arr[],int key)**

**{**

**for(int i=0;i<arr.length;i++)**

**{**

**if(key==arr[i])**

**{**

**return i;**

**}**

**}**

**return -1;**

**}**

**public static void main(String args[])throws IOException**

**{**

**System.out.println("\u000c");**

**BufferedReader z=new BufferedReader(new InputStreamReader(System.in));**

**int n;**

**System.out.print("Enter n:");**

**n=Integer.parseInt(z.readLine());**

**System.out.println();**

**int arr[]=new int[n];**

**for(int i=0;i<n;i++)**

**{**

**arr[i]=Integer.parseInt(z.readLine());**

**}**

**int key;**

**System.out.print("Enter the element to search:");**

**key=Integer.parseInt(z.readLine());**

**//call the searching function**

**int result=sequential\_search(arr,key);**

**if(result==-1)**

**{**

**System.out.println("The element is not found in the array");**

**}**

**else**

**{**

**System.out.println("The element is first found at the index:"+result);**

**}**

**}**

**}**

This sequential\_search algorithm (or we can say all searching algorithms) has a disadvantage. If an element is found multiple times in the array, then it will only print the first position where the element is found (by the respective algorithm…for instance, binary search algorithm has similar kind of backdrop…though we can optimize the sequential search algorithm to print all the positions..but we can’t do the same for binary search algorithm).

**Binary search**

Most important point about binary search is that it can only be applied for sorted array.

The following algorithm mentioned will only work for arrays sorted in ascending order.

We have to modify the algorithm a little if we want to make it work for descending order . (But cannot work for both at the same time)

program for binary search:

import java.io.\*;

**class Binarysearch**

**{**

**public static int binarysearch(int arr[],int low,int high,int key)**

**{**

**int mid=(low+high)/2;**

**if(low<=high)**

**{**

**if(arr[mid]==key)**

**{**

**return mid;**

**}**

**if(arr[mid]>key)//that means element is present in the first half**

**{**

**return binarysearch(arr,0,mid-1,key);**

**}**

**if(arr[mid]<key)**

**{**

**return binarysearch(arr,mid+1,high,key);**

**}**

**}**

**return -1;**

**}**

**public static void main(String args[])throws IOException**

**{**

**System.out.println("\u000c");**

**BufferedReader z=new BufferedReader(new InputStreamReader(System.in));**

**int n;**

**System.out.print("Enter n:");**

**n=Integer.parseInt(z.readLine());**

**System.out.println();**

**int arr[]=new int[n];**

**for(int i=0;i<n;i++)**

**{**

**arr[i]=Integer.parseInt(z.readLine());**

**}**

**int key;**

**System.out.print("Enter the element to search:");**

**key=Integer.parseInt(z.readLine());**

**//call the searching function**

**int result=binarysearch(arr,0,n-1,key);**

**if(result==-1)**

**{**

**System.out.println("The element is not found in the array");**

**}**

**else**

**{**

**System.out.println("The element is first found at the index:"+result);**

**}**

**}**

**}**

**How does this algorithm work?**

**Suppose, the initial array is:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **22** | **33** | **44** | **55** | **66** | **77** | **88** |
|  |  |  |  |  |  |  |  |

and we have to search element 88 in it

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **22** | **33** | **44** | **55** | **66** | **77** | **88** |
| **low** |  |  | **mid** |  |  |  | **high** |

Now, arr[mid]<key

So, low will be mid+1 for next call

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **22** | **33** | **44** | **55** | **66** | **77** | **88** |
|  |  |  |  | **low** | **mid** |  | **high** |

arr[mid]<key

So, low will be mid+1 for the next call

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **22** | **33** | **44** | **55** | **66** | **77** | **88** |
|  |  |  |  |  | **low** | **mid** | **high** |

Now, x[mid]<key

So, low will be mid+1 for the next call

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **22** | **33** | **44** | **55** | **66** | **77** | **88** |
|  |  |  |  |  |  |  | **low,high,mid** |

Now, arr[mid]=key

So, mid (which is 7 now) is returned to the main

Now, let’s try to find the steps for an element which is not in the array. Let’s take the same initial array

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **22** | **33** | **44** | **55** | **66** | **77** | **88** |
|  |  |  |  |  |  |  |  |

and try to find element 45 in the array

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **22** | **33** | **44** | **55** | **66** | **77** | **88** |
| **low** |  |  | **mid** |  |  |  | **high** |

Now, x[mid]<key

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **22** | **33** | **44** | **55** | **66** | **77** | **88** |
|  |  |  |  | **low** | **mid** |  | **high** |

Now, x[mid]>key

So, high becomes mid-1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **22** | **33** | **44** | **55** | **66** | **77** | **88** |
|  |  |  |  | **low,high,mid** |  |  |  |

Now, x[mid]>key

So, low=mid+1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **22** | **33** | **44** | **55** | **66** | **77** | **88** |
|  |  |  |  | **high** | **low** |  |  |

Now, low<=high condition is not maintained anymore.

So, index will be returned as -1

There’s another searching technique named interpolation sort. It also works only sorted array and produces better result than binary search if and only the elements are well distributed in the given range (between first and last element). However, it is not in the syllabus.