# Searching

## Searching Algorithms

### Linear Search

* Works in unsorted arrays
* Keep searching one by one until you find it.
* O(n)

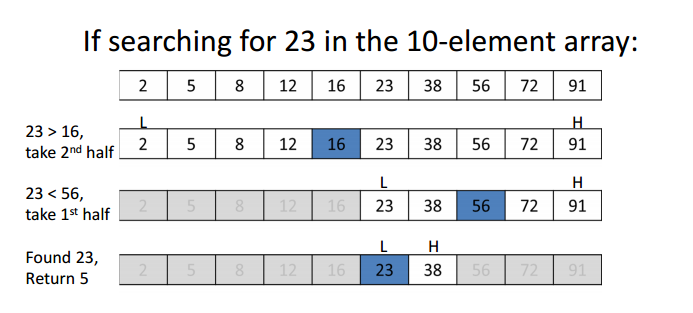
### Binary Search

**How it works**

* Find the mid-point
* If what you wanted was the mid-point then you are done
* If the middle point is smaller than what you want then you search the right part. You call it recursively using mid + 1 and right. Notice that you don't include mid.
* If the middle point is greater than what you want then you search the left part. You call it recursively using left and mid - 1. Notice that you don't include mid.
* If at any point right > left then return with -1 saying that you could not find it
* O(log(n)) efficiency

**Analysis of Searching Algorithm**

|  |  |  |  |
| --- | --- | --- | --- |
| Best Case Time | Worst Case Time | Average Case Time | Space Complexity |
| O(1) | O(logn) | O(logn) | O(1) |



### Jump Search

* Works only in sorted arrays
* Take jumps while search something.
* If the number is small then make the jump
* If the number is bigger then go linear search from previous point to current point
* The best size to jump is sqrt(n) where n is the number of jumps
* O(sqrt(n))
* This is faster than Binary Search

## Searching in java

### Arrays.binarySearch()

* The most efficient way to search in array.
* It only works with sorted arrays.
* In case of unsorted array you have to do Arrays.sort() otherwise it will not work.
* If the return answer is negative then it means the element was not found.

### Collections.binarySearch()

* The most efficient way to search in a list.
* It only works with sorted lists.
* In case of unsorted list you have to do Collections.sort() otherwise it will not work.
* If the return answer is negative then it means the element was not found.

### For searching, in case of array, if you are implementing binary search, then the function should accept a lo and hi so that it knows which part of the array it is searching.

### To find floor and search, always check the next and previous of middle value to see if they are greater and less than the number you are searching for

### Find 3rd largest element in 1 pass

* Make 3 variables. First, second and third.
* When you see an element bigger than first
  + third = second
  + second = first
  + first = new element
* When you see an element bigger than second
  + third = second
  + second = new element
* When you see an element bigger than third
  + third = new element

### Find 2nd largest element using minimum comparisons

* Use merge sort technique.
* Compare every two element and keep track of first and second
* Now compare the result of those and keep track of first and second

### Find missing number in an array from 1 to n

* Take the highest number and take it as 'n'.
* Use (n\*(n+1))/2 to calculate the sum.
* Now subtract each element from this sum.
* The answer left in the end will be your answer.

## **Two numbers in array with +ve and -ve** **numbers whose sum is closest to zero**

* Sort the array
* Find the sum of first and last.
* If sum > 0 then take the sum of first and second last
* If sum < 0 then take the sum of second and last
* If sum = 0 then you have your answer
* Keep doing this recursively

## **Two numbers in array whose sum is closest to a specific number**

* Sort the array
* Keep a minimum index which will save the index for which it was closest
* Find the sum of first and last.
* If sum > number then take the sum of first and second last
* If sum < number then take the sum of second and last
* If the sum is closer than the closest you have seen till now then you have found your numbers.
* Keep doing this recursively

### Find a pair with given difference

* Make a HashMap
* Go through every element and see if it is there in the HashMap
* If it is then print the key and value and return
* If not then add the following to HashMap
  + hm.put(arr[i] - difference, arr[i])
  + hm.put(arr[i] + difference, arr[i])

## **For frequency questions, use a HashMap to count the** **frequency**

## **If an array is sorted and the question is asking for frequency of certain number then use binary search**

* Use binary search to find the first occurrence.
* Use another binary search to find the last occurrence.
* Then use this two information to find the frequency.

## **Find union and intersection of two arrays**

* Make two indices for both the arrays.
* See which one of the index is smaller and add that to the new arraylist. Increment that index.
* If it was same then in union add any one of them to the list and increment both the indices.
* In case of intersection add only in the case when both of them are similar.

## **Only one element is occurring odd number of times**

* The idea is that there a certain number of elements so that the half of them is odd then the middle number and the number before that should be same. If not then there has been an element before that has been odd.
* If it is same then an element after that is odd because it is necessary that an element is odd.
* If the half of it is even then the previous number should not be the same. If it is then there is an odd element on the left.