# Binary Search (I)

CSCD 300 – Data Structures

Eastern Washington University



EASTERN WASHINGTON UNIVERSITY
start something big

# Outline



EASTERN WASHINGTON UNIVERSITY
start something big

# Binary Search [1]

## Problem statement [a]

[a] You can view the array element numbers to be of any type of object, depending on the applications, as long as there is a well defined total order on the objects meaning they can be compared to each other. You can also view the array as any type of "container", as long as the container supports random access to its locations.

Given a sorted array $A[\,]$ of numbers and another number $x$,

- return a location $i$ in the array $A$ such that $A[i] = x$, if $x$ exists in $A$;
- return -1, otherwise.

## Example 1

- Input:

  $A[\,] = \{2, 4, 7, 8, 9, 12, 16\}$
  $x = 8$

- Output: 3

## Example 2

- Input:

  $A[\,] = \{2, 4, 7, 8, 9, 12, 16\}$
  $x = 6$

- Output: $-1$

[1] We use 0-based indexing.

# Algorithmic idea: recursion based

```
We start at the middle location of A[] and compare A[mid] with x.

If A[mid] == x
   return mid

//then x must be on the right half portion of A, if x exists.
Else if A[mid] < x
   We RECURSIVE work on the right half portion of A

//then x must be on the left half portion of A, if x exists.
Else
   We RECURSIVE work on the left half portion of A
```

# Pseudo code

```
/* Find a location in A[low...high] whose value is equal to x */
binary_search_recursion(A, low, high, x)
{
   if(low > high)
      return -1

   mid = floor((low+high)/2)
   if(A[mid] == x)
      return mid
   else if(A[mid] < x)
      return binary_search_recursion(A, mid+1, high, x)
   else
      return binary_search_recursion(A, low, mid-1, x)
}
```

Initial call: `binary_search_recursion(A, 0, A.length-1, x)`

# However ...

We don't like the recursion-based structure as it has overhead in the system stack maintenance, so let's see how to implement the binary search idea using the iteration based structure.

# Binary search: iteration based

```
/* Find a location in A[ ], where the array element's
   value is equal to x */
binary_search_iteration(A, x)
{
   low = 0; high = A.lengh - 1;
   while (low <= high)
      mid = floor((low + high)/2)
      if(A[mid] == x)
         return mid
      else if(A[mid] < x)
         low = mid + 1
      else
         high = mid - 1

   return -1
}
```

In the next lecture, we will measure the speed of the binary search algorithm.

# Java code [2]

```
public class test_binary_search{
    public static void main(String[] args){
        int [] A = {1,3,5,7,9};
        System.out.println("BinarySearch(A,4): " + BinarySearch(A,4));
        System.out.println("BinarySearch(A,5): " + BinarySearch(A,5));
    }
    public static int BinarySearch(int[] A, int x)
    {
        int low = 0, high = A.length-1, mid = (low + high) / 2;
        while(low <= high){
           mid = (low + high) / 2;
           if(A[mid] == x)  return mid;
           else if(A[mid] < x) low = mid + 1;
           else high = mid - 1;
        }
        return -1;
   }
}
```

[2] In the future, I will only provide pseudocode based description most of the time. You all do the job of translating the description into Java code.