Data Structures And Algorithms

Problem Statement

- **How would you search for an element in an array/list whose size is unknown?**

You will proceed by applying the binary search algorithm on sub-arrays. Instead of taking the “end” pointer of the array as the last element of it,
make it second element of the array. “start” as it is(first element).

*public static int **getSubArray**(int array[], int key)*

Apply binary search, and increase “end” pinter to double the previous sub-array bound size after each unsuccessful attempt.

*public static int **binarySearch**(int array[], int start, int end, int key)*

Calculating mid index value using start and end pointer. This formula is used to avoid overflow for large value of int data type.

```
int mid = (start + end) >>> 1;
```

Suppose mid exceeds the array limit, you get “ArrayIndexOutOfBoundsException”. You need to handle the exception gracefully in order to keep it
going and decrement the “end” pointer until you get a valid index.

The code for the same is on next Slides….

```
import java.util.Scanner;
class BinarySearchWithoutArrayLength {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in); // Creating
a Scanner object
int length = scanner.nextInt(); // User input for length of
the array
int array[] = new int[length + 1]; // Declaring and
allocating memory of size "length+1" for 1-indexed array

int key = scanner.nextInt(); // User input for key that
need to be find out in the array
for(int i = 1; i <= length; i++){ // User input of array
elements (Array must be sorted as per our prerequisite)
array[i] = scanner.nextInt();
}

if (length == 0) { // Checking if length is zero
System.out.println("Length should be greater than 0");

}
else {
int index = getSubArray(array, key);
if (index > -1) { // If index is valid
System.out.println(index) ;
}
else { // Key not found
System.out.println("NOT_FOUND");
}
}
}
```

```
public static int getSubArray(int array[], int key) {

int start = 1; // start index as 1 because of 1-indexed type array
int end = 2; // end index as 2 because we are not sure about the length
of array.


// Checking for end value 2 exists in array or not. Although it would not
be the case with element of infinite array.


end = getValidEndIndex(array, start, end);


// While loop will execute until end value of array become greater than
key
while (array[end] < key) {
int tempEnd = end + (end - start + 1) * 2; // Increase end index to double
the previous sub array bound size


start = end + 1; // increase start index by 1
end = getValidEndIndex(array, start, tempEnd); // Get valid end index

if(tempEnd > end) { // Limit reached
break;
}
}

// Apply binary search to find the key in the bound range

return binarySearch(array, start, end, key);
}
```

```
public static int binarySearch(int array[], int start, int end,
int key) {

while (start <= end) {
// Calculating mid index value using start and end index.
This formula is used to avoid overflow for large value fo int
data type
int mid = (start + end) >>> 1;
if (key == array[mid]) { // Checking if value at index mid is
equal to the key
return mid; // Return index
}
if (key <= array[mid]) { // Checking if value at index mid is
greater than the key
end = mid - 1; // Update end index to first half of array

} else {
start = mid + 1; // Update start index to second half of the
array
}
}
// If element is not found.
return -1;
}
```

```java
public static int getValidEndIndex(int array[], int start, int end) {

try {

int value = array[end]; // Throws exception if array is out of bound
return end; // Return end index if value is valid

} catch(ArrayIndexOutOfBoundsException e) {

if(end > start && end >= 1) { // Check if end index is greater than start index

end -= 1; // Reduce end index by 1

return getValidEndIndex(array, start, end); // Call recursively getValidEndIndex() with new end to check its validity

}

else {

return end; // Return valid end

}

}

}

}
```

Thanks….

- Efforts By-
- Kabir Sharma
- SID-21104092