

NAME _____

DATE _____

PERIOD _____

UNIT 4 · TOPIC 4.17 · WORKSHEET

Split, Search, Merge

Walk a String, chase a binary search, and merge your way to sorted.

About 30 minutes**9 questions****Cumulative through 4.17**

TOPIC

4.17

**DIRECTIONS**

Work in pencil, and write exactly what prints. Every method shown is a public static method in the same class as main.

PART A · 2 QUESTIONS

A Warm up your brain

1 YOU'RE THE JVM

What does `countE("geese")` return, and how many calls to `countE` are made in all?

```
public static int countE(String s) {  
    if (s.length() == 0) {  
        return 0;  
    }  
    int rest = countE(s.substring(1));  
    if (s.substring(0, 1).equals("e")) {  
        return rest + 1;  
    }  
    return rest;  
}
```

A returns 3, 6 calls**B** returns 3, 5 calls**C** returns 0, 6 calls**D** returns 5, 6 calls

2 YOU'RE THE JVM

Binary search, recursive edition. `nums` is sorted. Trace `search(nums, 15, 0, 10)`: one row per call, in order. In the last column write *left*, *right*, *found*, or *return -1*.

```
int[] nums = {4, 9, 13, 18, 22, 27, 31, 36, 40, 45, 52};
// index:    0  1  2  3  4  5  6  7  8  9 10
```

```
public static int search(int[] nums, int target, int low, int high) {
    if (low > high) {
        return -1;
    }
    int mid = (low + high) / 2;
    if (nums[mid] == target) {
        return mid;
    } else if (nums[mid] < target) {
        return search(nums, target, mid + 1, high);
    } else {
        return search(nums, target, low, mid - 1);
    }
}
```

Call	low	high	mid	nums[mid]	Next
1	0	10	5	27	left
2					
3					
4					



FAIR QUESTION

The loop version of binary search works too. Why use recursion?

You don't have to. Binary search can be written with a `while (low <= high)` loop or with recursion, and both check the same elements in the same order. In the recursive version, the parameters `low` and `high` do the job the loop's variables do, and `low > high` plays the part of the loop condition turning false.

PART B · 4 PUZZLES

B Puzzles

3 CODE TILES

`sum(nums, i)` should add up the elements of an `ArrayList<Integer>` from index `i` to the end. Build its body from four tiles. Two tiles are decoys.

```
public static int sum(ArrayList<Integer> nums, int i) {
    // your tiles go here
}
```

<code>return nums.get(i) + sum(nums, i + 1);</code>	<code>if (i == nums.size() - 1) {</code>
<code>return 0;</code>	<code>return nums.get(i) + sum(nums, i);</code>
<code>}</code>	<code>if (i == nums.size()) {</code>

main runs (nums holds 5, 8, 2)	Target output
<code>System.out.println(sum(nums, 0));</code>	15
<code>System.out.println(sum(nums, 1));</code>	10

THE BODY OF SUM

4 SNIPPET POOL

merge combines two sorted arrays into result, in sorted order. Fill each hole from the pool so that merging {3, 9, 14} with {5, 6, 20, 25} gives [3, 5, 6, 9, 14, 20, 25] and merging {2, 8, 30} with {4, 11} gives [2, 4, 8, 11, 30].

```
public static void merge(int[] result, int[] left, int[] right) {
    int i = 0;
    int j = 0;
    int k = 0;
    while (i < left.length && j < right.length) {
        if (left[i] (1) right[j]) {
            result[k] = left[i];
            i++;
        } else {
            result[k] = right[j];
            j++;
        }
        k++;
    }
    while ((2)) {
        result[k] = left[i];
        i++;
        k++;
    }
    while ((3)) {
        result[k] = right[j];
        j++;
        k++;
    }
}
```

THE POOL

Each snippet can be used only once, and not every snippet is needed.

j < right.length	>=	i < right.length	<=
i < left.length	j < left.length	==	

5 PENCIL CHECK

mergeSort sorts {29, 14, 37, 8, 21, 5}, using the merge method from question 4. Write the arrays in each merge, in the order the merges happen. The first one is done for you.

```
public static void mergeSort(int[] nums) {
    if (nums.length > 1) {
        int[] left = new int[nums.length / 2];
        int[] right = new int[nums.length - left.length];
        for (int i = 0; i < left.length; i++) {
            left[i] = nums[i];
        }
        for (int i = 0; i < right.length; i++) {
            right[i] = nums[left.length + i];
        }
        mergeSort(left);
        mergeSort(right);
        merge(nums, left, right);
    }
}
```

Merge	left	right	Result
1	[14]	[37]	[14, 37]
2			
3			
4			
5			



WATCH OUT

Splitting doesn't sort anything

Merge sort cuts each array at a position and keeps the original order; it compares values only while merging. Splitting {7, 3, 5, 1} gives [7, 3] and [5, 1], exactly as they were, and then [7], [3], [5], [1].

6 FACE-OFF

Tonight: binary search and linear search argue about who finds things faster. Fill in the blanks.

BINARY	Hand me 1,000 elements and I'll find any target in at most _____ checks. Every check throws away _____ of what's left.
LINEAR	Impressive. I might need all _____. But I have one advantage: I work on data in _____ order.
BINARY	True. My data has to be _____, or I can skip right past the target.
LINEAR	And if the target is the very _____ element, I find it in one check, while you start in the middle.
BINARY	Fine. I'm <i>typically</i> faster, not always.

PART C · 3 QUESTIONS

C Catch the mistake

7 ONE-TOKEN FIX

`maxFrom(a, i)` should return the largest value from index `i` to the end of the array. With `a` holding `{4, 9, 2, 7}`, `maxFrom(a, 0)` should return 9, but it returns 2. Change exactly one token to fix it.

```

1 public static int maxFrom(int[] a, int i) {
2     if (i == a.length - 1) {
3         return a[i];
4     }
5     int best = maxFrom(a, i + 1);
6     if (a[i] < best) {
7         return a[i];
8     }
9     return best;
10 }
```

LINE, OLD TOKEN, NEW TOKEN

8 TWO TRUTHS AND A LIE

Someone runs the `search` method from question 2 on an array that isn't sorted. Two of these claims are true and one is a lie. Circle the lie, then rewrite it so it's true.

```
int[] data = {5, 12, 3, 20, 8};
System.out.println(search(data, 12, 0, 4));
```

- 1 It prints -1, even though 12 is at index 1.
- 2 The only elements the search checks are 3 and 20.
- 3 Binary search can't handle unsorted data, so this call throws an exception.

THE LIE, REWRITTEN

9 WHO AM I?

The searching and sorting algorithms are playing a party game. Each gives a clue about itself. Name it.

Clue	I am...
I split until every piece has one element, then merge the pieces back in order.	
I check the middle, then throw away half of what's left.	
I check one element at a time from the front, and any order is fine with me.	
Each pass, I find the smallest value left and swap it into place.	
I take the next element and slide it left into the sorted part.	