

Split, Search, Merge

Recursive searching and sorting



Guess my number

PAUSE AND TRY

I'm thinking of a number from 1 to 100. I'll only say higher or lower. What's your first guess, and why?

TWO STRATEGIES

- A. Guess 1, then 2, then 3, ...
- B. Guess the middle, then the middle of what is left, ...

With a partner: at most how many guesses does each strategy need?

By the end of class you can...

- 1 Trace recursion that walks a String, an array, or an ArrayList one position at a time.
- 2 Explain why binary search needs sorted data, and trace each step of one.
- 3 Read binary search written with a loop and with recursion.
- 4 Compare how many elements binary search and linear search check.
- 5 Trace merge sort as it splits an array and merges it back in sorted order.

A large, white, rounded square containing the number '1' in a bold, black, sans-serif font.

1

PART 1

One position at a time

A String, one letter shorter each call

CALLED WITH REVERSE("FROG")

```
public static String reverse(String s) {  
    if (s.length() == 0) {  
        return "";  
    }  
    return reverse(s.substring(1)) + s.substring(0, 1);  
}
```

OUTPUT
gorf

Progress: **substring(1)** is one letter shorter. Base case: **nothing left**.

Call	reverse("frog")	reverse("rog")	reverse("og")	reverse("g")	reverse("")
Builds	"gor" + "f"	"go" + "r"	"g" + "o"	" " + "g"	base case
Returns	"gorf"	"gor"	"go"	"g"	" "

Read the Returns row from right to left: the answer is built on the way back.

WORKED EXAMPLE

An array, one index further each call

```
A = {6, 3, 8, 5}, CALLED WITH COUNT EVEN(A, 0)

public static int countEven(int[] a, int i) {
    if (i == a.length) {
        return 0;
    }
    int rest = countEven(a, i + 1);
    if (a[i] % 2 == 0) {
        return rest + 1;
    }
    return rest;
}
```

OUTPUT
2

i	a[i]	Returns
0	6	1 + 1 = 2
1	3	1
2	8	0 + 1 = 1
3	5	0
4	none	base case: 0

Fill the Returns column from the bottom up.

Searching an ArrayList recursively

PAUSE AND TRY

names holds "Ana", "Bo", "Lu", "Mia". What does each call return, and how many calls does it make?

```
public static int find(ArrayList<String> names,
                      String target, int i) {
    if (i == names.size()) {
        return -1;
    }
    if (names.get(i).equals(target)) {
        return i;
    }
    return find(names, target, i + 1);
}
```

Called as `find(names, target, 0)`

target	Returns	Calls
"Lu"		
"Zed"		

Pass along something smaller



Every recursive call must move toward the base case: a **shorter String** or the **next index**. Otherwise the calls pile up until the program crashes.

CALLED WITH REVERSE("FROG")

```
public static String reverse(String s) {  
    if (s.length() == 0) {  
        return "";  
    }  
    return reverse(s) + s.substring(0, 1);  
}
```

OUTPUT

```
Exception in thread "main"  
java.lang.StackOverflowError
```

Fix: **reverse(s.substring(1))**

A large, white, rounded square containing a bold teal number '2'.

2

PART 2

Binary search

Check the middle, throw away half



Binary search needs **sorted** data. It starts at the middle and **eliminates half** of what is left with each check.

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91

low 0, high 9: $16 < 23$, go right

2	5	8	12	16	23	38	56	72	91
---	---	---	----	----	----	----	----	----	----

low 5, high 9: $56 > 23$, go left

2	5	8	12	16	23	38	56	72	91
---	---	---	----	----	----	----	----	----	----

low 5, high 6: found 23 at index 5

Searching for 23. Gray cells are thrown away; yellow is the middle being checked.

Binary search with a loop

CALLED WITH `BINARYSEARCH(NUMS, 60)`

```
public static int binarySearch(int[] nums,
                               int target) {

    int low = 0;
    int high = nums.length - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (nums[mid] == target) {
            return mid;
        } else if (nums[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}
```

low	high	mid	nums[mid]	Next
0	9	4	16	go right
5	9	7	56	go right
8	9	8	72	go left
8	7	none	none	loop ends

OUTPUT

-1

nums = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91}

The same search, written recursively

CALLED WITH SEARCH(NUMS, 23, 0, 9)

```
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);
    }
}
```

OUTPUT

5

Loop version	Recursive version
low, high: local variables	low, high: parameters
keeps going while low <= high	base case: low > high
low = mid + 1, then loop	call with mid + 1

Unsorted data: no crash, just a wrong answer



On unsorted data, binary search can throw away the half that holds the target. It returns **-1** for a value that is really there.

```
int[] mixed = {9, 2, 14, 6, 11};  
System.out.println(search(mixed, 2, 0, 4));
```

OUTPUT

-1

0	1	2	3	4
9	2	14	6	11

14 > 2, go left

9	2	14	6	11
----------	---	----	---	----

9 > 2, go left; then low > high

Your turn: search for 8

PAUSE AND TRY

Trace binary search for 8. Fill in a row for each check. What is returned?

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91

low	high	mid	nums[mid]	Next
0	9			

Returns: _____

Binary search vs. linear search

Elements	Linear, worst case	Binary, worst case
10	10 checks	4 checks
100	100 checks	7 checks
1,000	1,000 checks	10 checks
1,000,000	1,000,000 checks	20 checks

Binary search

Needs sorted data. Wins on big data.

Linear search

Any order. Wins when the target is near the front.



Binary search is **typically** more efficient than linear search.

3

PART 3

Merge sort

Split down to one element, then merge up

START

41	8	33	15	27	4	19	50
----	---	----	----	----	---	----	----

SPLIT

41	8	33	15	27	4	19	50
----	---	----	----	----	---	----	----

SPLIT

41	8	33	15	27	4	19	50
----	---	----	----	----	---	----	----

SPLIT

41	8	33	15	27	4	19	50
----	---	----	----	----	---	----	----

MERGE

8	41	15	33	4	27	19	50
---	----	----	----	---	----	----	----

MERGE

8	15	33	41	4	19	27	50
---	----	----	----	---	----	----	----

MERGE

4	8	15	19	27	33	41	50
---	---	----	----	----	----	----	----

Merging two sorted halves

```
left  = {8, 15, 33, 41}
right = {4, 19, 27, 50}
```

OUTPUT

```
[4, 8, 15, 19, 27, 33, 41, 50]
```



Always take the **smaller** front value.
When one half runs out, copy the **leftovers** of the other.

Compare	Copy
8 vs 4	4
8 vs 19	8
15 vs 19	15
33 vs 19	19
33 vs 27	27
33 vs 50	33
41 vs 50	41
left is used up	50 (leftover)

List the merges, in order

PAUSE AND TRY

Merge sort runs on {12, 7, 3, 10, 5}. The left part gets length / 2 elements.
Write each merge in order.

Merge	left	right	Result
1			
2			
3			
4			

Words to know

Binary search

Checks the middle of sorted data and eliminates half each step.

Linear search

Checks elements one at a time, in any order.

Merge sort

Recursively splits into one-element pieces, then merges them in order.

Merge

Combine two sorted pieces by taking the smaller front value each time.

low and high

The ends of the range binary search still has to check.

Worst case

The most checks any target can need.

Both are recursive. How are they different?



Explain to your partner how binary search and merge sort each use recursion.

SENTENCE STARTER

Binary search calls itself on _____ of the data, but merge sort calls itself on _____ and then _____.

Next: the Split, Search, Merge puzzle worksheet.