

## Write a recursive function for generating all permutations of an input string. Return them as a set.

Don't worry about time or space complexity—if we wanted efficiency we'd write an iterative version.

To start, assume every character in the input string is unique.

Your function can have loops—it just needs to also be recursive.

### Gotchas

Make sure you have a base case ! Otherwise your function may never terminate!

### Breakdown

Let's break the problem into subproblems. How could we re-phrase the problem of getting all permutations for "cats" in terms of a smaller but similar subproblem?

**Let's make our subproblem be getting all permutations for all characters except the last one.** If we had all permutations for "cat," how could we use that to generate all permutations for "cats"?

We could put the "s" in each possible position for each possible permutation of "cat"!

These are our permutations of "cat":

```
cat
cta
atc
act
tac
tca
```

For each of them, we add "s" in each possible position. So for "cat":

```
cat
  scat
  csat
  cast
  cats
```

And for "cta":

```
cta
 scta
 csta
 ctsa
 ctas
```

And so on.

Now that we can break the problem into subproblems, we just need a base case and we have a recursive algorithm!

## Solution

If we're making all permutations for "cat," we take all permutations for "ca" and then put "t" in each possible position in each of those permutations. We use this approach recursively:

```
public Set<String> getPermutations(String inputString) {  
  
    // base case  
    if (inputString.length() <= 1) {  
        return new HashSet<String>(Arrays.asList(inputString));  
    }  
  
    String allCharsExceptLast = inputString.substring(0, inputString.length() - 1);  
    char lastChar = inputString.charAt(inputString.length() - 1);  
  
    // recursive call: get all possible permutations for all chars except last  
    Set<String> permutationsOfAllCharsExceptLast = getPermutations(allCharsExceptLast);  
  
    // put the last char in all possible positions for each of the above permutations  
    Set<String> permutations = new HashSet<String>();  
    for (String permutationOfAllCharsExceptLast : permutationsOfAllCharsExceptLast) {  
        for (int position = 0; position <= allCharsExceptLast.length(); position++) {  
            String permutation = permutationOfAllCharsExceptLast.substring(0, position) + lastChar +  
                permutationOfAllCharsExceptLast.substring(position, allCharsExceptLast.length());  
            permutations.add(permutation);  
        }  
    }  
  
    return permutations;  
}
```

## Bonus

What if our input string had some duplicates?

What if we wanted to bring down the time and/or space costs?

## What We Learned

This is one where *playing with a sample input* is huge. Sometimes it helps to think of algorithm design as a two-part process: *first* figure out how you would solve the problem "by hand," as though the input was a stack of paper on a desk in front of you. *Then* translate that process into code.

---

Want more coding interview help?

Check out **[interviewcake.com](https://www.interviewcake.com)** for more advice, guides, and practice questions.