CSSE2002/7023

Semester 2, 2021

Programming in the Large

Week 11.2: Recursion & Sorting

In this Session

- Recursion
- Sorting in Java
- Merge Sort
- Quick Sort

Recursion — Reminder

- Functions call themselves
  - possibly indirectly
- Make sure the recursion will stop eventually
  - base case
- Recursive calls should be a reduction case
  - smaller / simpler instances
- Beware stack overflow

Why Recursion?

Some people may say "It's easier".

- Ok, what does "easier" mean?
- . . . to read?
- . . . to write?
- . . . for whom?
  - Me?
  - Someone else?

Is the person saying it:

- A theoretical computer scientist?
  - Is it "easy" because of 20 years experience?
- A fan of "functional programming" languages[1]?
  - Not that there is anything wrong with that.
  - These are designed around using a recursive model of computation. So it probably is easier.

[1] Lisp, Haskell, Erlang, . . .

Why Not Iteration?

Can't we use iteration[2] instead?

Sort of.

There is a theoretical result that yes, recursive algorithms can be converted into algorithms which use loops (and a stack).

- Using a loop, we would have to manually manage the call stack (can become tedious). This is done automatically if we use recursion.

But, there are some algorithms which are easier and simpler to express in a recursive form. e.g. Fibonnaci

$F(1) = 1$
$F(2) = 1$
$F(n) = F(n-1) + F(n-2)$

How would we express this iteratively in a loop?

[2] loops

Iteration to Recursion Example

Write a recursive method reverseWordsInString that given a string as a parameter, will reverse the order of the words in the string, and return the result.

E.g. Passing “this is a sentence” as the parameter would return “sentence a is this”.

You cannot use the split() method or regular expressions, in any way.

All other methods are allowed though.

Iteration to Recursion Example

Write a recursive method reverseWordsInString that given a string as a parameter, will reverse the order of the words in the string, and return the result.

E.g. Passing “this is a sentence” as the parameter would return “sentence a is this”.

You cannot use the split() method or regular expressions, in any way.

All other methods are allowed though.

```java
public static String reverseWordsInString(String text) {
  String result = ""; // the entire string
  String word = ""; // current word
  int previousSpace = 0;

  // need to work out if there is another word to check
  int nextSpace = text.indexOf(' ', previousSpace);

  // base case. i.e. there is no more words
  if (nextSpace == -1) {
        return text;
  }

  // continue case. There is at least one more word.
  else {
        // get the next word
        word = text.substring(previousSpace, nextSpace);
        previousSpace = nextSpace + 1;
        // calls the same function.
        // append result of function call to start of result.
        return reverseWordsInString(text.substring(previousSpace))
                + " " + word + result;
  }
}
```

Recursion to Iteration Example

```java
public class DataProcessor {
  public static int process(int[] data, int i, int l) {
    if (l == 1) { // base case
          return data[i];
    }
    else { // continue case
      int value = process(data, i + 1, l - 1);
      if (data[i] < value) {
        return data[i];
      }
      return value;
    }
  }

  public static void main(String[] args) {
    int[] data = {4, 6, 2, 8, 1, 9};

    System.out.print(process(data, 0, data.length)); //usage example
  }
}
```

a) In the code above, explain the purpose of the publicly accessible function process(int[] data, int i, int l)?

b) Write an iterative version of the function process(int[] data, int i, int l), called processIter, which achieves the same functionality. You can use any number parameters, and the data type for the parameters is not restricted. You cannot use any additional Java functions or Java libraries in your implementation

Recursion to Iteration Example

a) This program finds the smallest number in an array of numbers.

b)

```
public static int processIter(int[] data) {
  int smallestValue = 0;
  smallestValue = data[0];
  for (int i = 0; i < data.length; i++) {
    if (data[i] < smallestValue) {
      smallestValue = data[i];
    }
  }
  return smallestValue;
}
```

Sorting

We'll look at sorting algorithms as an example of a task which recursion can make easier.

Sorting a `List` in Java

```
List l;
...
l.sort(null);   // use default ordering
```

See: `Sort.java`

Doesn't give us much insight into how Java's sort() method works though.

Read the implementation note for insight:
https://docs.oracle.com/javase/8/docs/api/java/util/List.html#sort-java.util.Comparator-

Merge Sort

Task: sort `arr` — an array of n `ints`.

- Suppose we know how to sort $n/2$ ints.
- Split `arr` in half
- Sort each half
- Merge the two sorted parts into one sorted whole

See: `MSort.java`

- Works because a single element array is automatically sorted
- Sorts “bottom up”

Quick Sort

Suppose we have a method `partition(arr)`

- Returns the index of one element in arr which is guaranteed to be in the correct place.
- Everything to the left of that element is $\leq$ to it.
- Everything to the right of that element is $\geq$ to it.

See: `QSort.java`

Which is Better?

It depends ...

Criterion	Better
Simplicity	Mergesort
Worst case performance	Mergesort
Expected performance	Quicksort
Memory requirements	Quicksort

Why? See an algorithms course (COMP3506).

Further Exploration

https://visualgo.net/en/sorting

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

http://sorting.at/