# APCS Notes

Yicheng Wang

2014-2015

# Contents

# 1 2015-02-05

## 1.1 Do Now

Figure out what the following code does.

```
public void printme(int n) {
    if (N > 0) {
        printme(n - 1);
        System.out.println(n);
    }
}
```

It should print out an increasing sequence of numbers from 1-N.

## 1.2 Stack and ROP

The stack on top is the current function, and each layer beneath that is the function that called the current function.

## 1.3 Recursion

Simple recursive problem: FACTORIAL
Hallmarks of a recursive solutioin:

- Base Case: thing that stops the program, simple case you know the answer of. In the case of factorials, factorial(0) = 1
- Reduction Case: You need to alternate the variable in some sort of way, for example, we should do n * factorial(n - 1)
- Recursion: function A need to eventually call A

Final code:

```
public int factorial(n) {
    if (n == 0) {
        return 1; // Base Case
    }
    else {
        return n * factorial(n - 1); // Reduction Step
    }
}
```

# 2 2015-02-06

## 2.1 Traditional Recursion Examples

### 2.1.1 Fibonacci Numbers

1, 1, 2, 3, 5, 8, 13 ...

Base Case: if n ¡ 2, return 1
Reduction Step: fib(n) = fib(n - 1) + fib(n - 2)
Example Code:

```
public int fib(int n) {
    if (n < 2) {
        return 1;
    }

    else {
        return fib(n - 1) + fib(n - 2);
    }
}
```

### 2.1.2 List/String Manipulation

Example, finding the length of a substring.
Base case: ”” has length of 0
Reduction Step: 1 + ”cdr” of the string, i.e. s.substring(1);
Example Code:

```
public int lenStr(String s) {
    if (s.equals("")) {
        return 0;
    }
    else {
        return 1 + lenStr(s.substring(1));
    }
}
```

# 3 2015-02-09

## 3.1 Getting out of a Maze

1. Maze vs. Laybrith: Laybrith may not have choices, mazes have choices.
2. Strategies in solving the maze:
    - Doesn't work due to loops
3. Greek Way of Solving Mazes:
    - Invented by Odysseus, bring a thread, use process of elimination to loop through all possible intersection.
    - Works really well.

Maze solving in java

1. This is clearly a recursive solution.

2. Using a recursive solution we can easily trace back the stack to find the previous intersection
3. We represent our map as a char array of paths ('#')
4. Base case: location is a wall OR location of exit
5. Reduction step: call solve at a different (x,y) location, if it's a dead end, it'll be peeled off the stack due to the "return"
6. We will try to solve the maze systematically, x+/-1, y+/-1.

# 4 2015-02-11

## 4.1 Blind Search

Trying all possibilities until we find our solutions. Also called exhaustive search, linear search, recursive search, search with backtracking.

The maze algorithm we wrote is sometimes known as a depth first search. It's because you say in one path for as long as possible. Advantage is if the solution is deep, this works pretty well. However, if the solution is closer, breath first search goes n steps in all possible directions.

## 4.2 State Space Search

Search algorithm to solve a problem, searching for a "state space."

Idea is if you have a problem, you can describe said problem as a "state."

State is a configuration of the world, such as turtle in netLogo.

The world is made up of many states, and one can transition from one state to the next.

**State Space Search:** The series of states one has to go through to get to the desirable final state. (like exit in the maze thing)

## 4.3 Graph Theory

Graphs are collections of edges and nodes. Nodes represent the states, edges marks the transition between one node to the next.

# 5 2015-02-12

## 5.1 Space State Search

Examples of space state search:

- Maze path-finding
- 15 puzzle

- Cube
- Chess algorithm – More complex
  - two players!
  - high branching factor
  - harder base case
  - uses mini-max searching: best for me and worst for you
- Description of an entity

### 5.2 Implicit Data Structure

When we did the maze solver, we used an explicit data structure → the 2D array. However, the graph of the transition is also a data structure, it's implicit though, just running in the background. Whenever we call solve, it creates a node on the stack. However, if a call returns, it destroy that branch of the graph.

As the program runs, we can imagine it as a graph.

### 5.3 PROJECT

Do the diagonistic exam in the Barron's Review book, do as follows:

- First do it under test condition
- Go back and tried to fix your problem

**Option 1. Knight Tour:**

Given a $N \times N$ board, find a path such that the knight visits each square once without repetition. (Start with $5 \times 5$);

**Option 2. N-Queen:**

A way to place N mutually non-attacking queens on a $N \times N$ chessboard.

## 6 2015-02-13

Other problem, third variation, one can also do the 15 puzzle.

### 6.1 System.out.printf

When one is doing a knight's tour, the print output may be distorted if one moves from single digit to double digit. We can leave a placeholder in the format string, such as %s, %d, etc.

Example code:

```
...
System.out.printf("%s, helloWorld\n %s\n", "title", "more stuff!");
...

OUTPUT:

title, helloWorld
more stuff!
```

However, printf can take multiple types and it is possible to save spaces for formatters. Like the following:

```
...
System.out.printf("%3d\n%3d", 1, 123);
...

OUTPUT:

  1
123
```

# 7 2015-02-25

Iteninary:

- Tomorrow - Knight's Tour
- Tuesday - USACO

Today, we're going back to sorting.

# 8 Merge Sort

So far, we've covered 3 algorithms: selection sort, insertion sort, and bubble sort. They are all linear algorithms. Selection sort selects the ith index and puts the ith smallest/largest elements there, whereas insertion sort inserts the ith smallest/largest element so far into the ith index of the list.

HOWEVER, we're lazy and need to do this the 6 years old way. If we're sorting a deck of unsorted cards, we'll split the deck in half and give it to more people, so on, and so on. This continues until everyone has 1 card at hand. The process then goes backwards and the people merges the sorted lists passed on to him/her. This is known as a merge sort, it is a **divide and conquer algorithm.**

Sample code:

```
...
    public static int[] mergeSort(int[] data) {
        if (data.length == 1) {
            return data;
        }

```

```
        else {
            int[] A = Arrays.copyOfRange(data, 0, data.length / 2);
            int[] B = Arrays.copyOfRange(data, data.length / 2, data.length);

            int[] AS = mergeSort(A);
            int[] BS = mergeSort(B);
            return merge(AS, BS);
        }
    }

    public static int[] merge(int[] A, int[] B) {
        int[] result = new int[A.length + B.length];
        int position = 0;
        int APos = 0;
        int BPos = 0;
        while (APos < A.length && BPos < B.length) {
            if (A[APos] < B[BPos]) {
                result[position] = A[APos];
                APos++;
            }
            else {
                result[position] = B[BPos];
                BPos++;
            }
            position++;
        }

        for (int i = APos ; i < A.length ; i++) {
            result[position] = A[i];
            position++;
        }
        for (int i = BPos ; i < B.length ; i++) {
            result[position] = B[i];
            position++;
        }
        return result;
    }

...
```

# 9 2015-03-04

## 9.1 On the Algorithm of the Merge Sort

In insertion and selection sorts, the average operation has a complexity of $n^2$. This is because the inner loop runs through the list ($n$ operations) and the outer loop instructs us to run through the inner loop $n$ times, hence the total operations is $n^2$.

For merge sort, each step we split the list in half and sort the smaller part first. If we imagine this as a tree, we divide the list in half each time. So the vertical step takes $\log_2 n$ steps. Then at each step, we copy the array once so there is also a $n$ component to the total complexity. Then, the sum of each level's merge is also going to be $n$ because merging takes the same amount of work as splitting. Therefore, the total complexity is:

$$O(n) = n \log n$$

Let's look at the different rate of growth of the two curves ($n^2$ and $n \log n$). Take, 1,000,000 as $n$, $n^2 = 1.0 \times 10^{12}$, but $n \log n = 6 \times 10^6$

### 9.2 Big O Notation

A function $f(n)$ is said to be $O(g(n))$ if there exists some constant $k$ such that $kg(n) > f(n)$ over the long term.

Note that the Big O Notation is always and **Upper bound**, and is extracted from the worst-case scenario of the function. However, it is a very tight upper bound.

## 10 2015-03-05

We shall write an algorithm to find the $k^{th}$ smallest element.

A few ways to do this:

One, we can recursively delete the smallest data point, but this isn't very efficient. In fact, this is a $O(n^2)$ algorithm.

The other, cheap (my) way is to first merge sort it and then just do a lookup for the element. This has the complexity of only $O(n \log n)$.

Note that for hard problems, it is very hard for complexity to go below $O(n \log n)$. Therefore, merge sorting something is practically free, complexity-wise. Case in point: is it worthwhile to first sort an array and then use binary search with the merge sort or just use the linear search.

However, there is a more efficient way of doing this.

We can use a binary-search-like algorithm.

## 11 2015-03-09

### 11.1 Efficiency of Quick Select

Given that we are lucky, and if the pivot is good such that it bisects the array, on the first operation we only go through $\frac{n}{2}$ operations after going through the entire array. Then it just divides by 2 each time, so the final operation number is:

$$n + \frac{1}{2}n + \frac{1}{4}n + ... = 2n$$

So the quickselect runs on linear time!