Asymptotic Analysis

Stirling's formula: $$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

First, simplifications of some functions:

1) $$2^{\log_2\left(\sqrt{n}\right)} = \sqrt{n}$$

2) $$\binom{n}{3} = \frac{n!}{3!(n-3)!} = \frac{(n-3)!\cdot(n-2)\cdot(n-1)\cdot n}{3!(n-3)!} =$$

$$= \frac{(n-2)\cdot(n-1)\cdot n}{3!} = \frac{(n-2)\left(n^2-n\right)}{3!} = \frac{n^3-3n^2+2n}{3!}$$

3) $$\log_2\left(n^n\right) = n\log_2(n)$$

4) $$\left(n+1\right)! \approx \sqrt{2\pi(n+1)}\left(\frac{n+1}{e}\right)^{n+1}$$

5) $$\log_2(n!) \approx \log_2\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)$$

6) $$\binom{n}{n/2} = \frac{n!}{(n/2)!(n-n/2)!} = \frac{n!}{((n/2)!)^2} \approx \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{\left(\sqrt{2\pi\frac{n}{2}}\left(\frac{n/2}{e}\right)^{\frac{n}{2}}\right)^2} =$$

$$= \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{\pi n\left(\frac{n}{2e}\right)^n} = \frac{\sqrt{2}}{\sqrt{\pi n}\left(\frac{1}{2}\right)^n} = \frac{\left(\sqrt{2}\right)2^n}{\sqrt{\pi n}}$$

7) $$8^{\log_2(n)} = 2^{3\log_2(n)} = 2^{\log_2\left(n^3\right)} = n^3$$

$$10logn \to 2^{\log\sqrt{\mathrm{n}}} \to \sqrt{\mathrm{n}} + 91^{2886}\log n \to \frac{n}{2} \to 5n \to$$

$$\to nlogn \to log\left(n^n\right) \to log\left(n!\right) \to \binom{\mathrm{n}}{3} \to 8^{\mathrm{logn}} \to n + 5n^3 \to$$

$$\to \binom{\mathrm{n}}{\mathrm{n}/2} \to 2^{2n} \to 4^n \to 5n + n^n \to n! \to \left(n+1\right)!$$

1)

$a)\ O(n^3)$

$b)\ O(n^2)$

$c)\ O(n^3)\ tighter\ bound\ is\ O(n^{2.25})$

$d)\ O(n^3)$

$\mathrm{e)O(nlog_2(n))}$

$f)\ O(n\log^2{}_2(n))$

2)

$a)\ O(n)$

$b)\ undefined:$

Best case: the array will be sorted, best case is $\mathrm{O(n)}$
"Expected case": if we assume that all the permutations are equally likely, it might find the sorted array permutation in n! trials
"Worst case" : infinite loop and stack overflow

$c)\ infinite\ loop:$

j in the second loop starts with 0 and is multiplied by 2 for each iteration, so it simply is evaluated to 0 infinitely
but, if j in the second for loop started with j = 1 instead of 0, the time complexity would be $\mathrm{O(nlog_2(n))}$

$\mathrm{d)\ O(nlog_2(n))}$

$e)\ O\left(n(\log_2(n))^2\right):$

$first\ iteration\ runs\ logn,\ second\ runs\ logn,\ third\ runs\ n/2$

$f)\ O(n)$

GCD

```
public class GCD {
    public static int gcd(int a, int b) {
        if(b == 0) {
            return a;
        }
        return gcd(b, a % b);
    }

    public static void main(String[] args) {
        System.out.println(gcd(1071, 462));
    }
}
```

SumOfDigits

```
public class SumOfDigits {
    public static int sumOfDdigits(int n) {
        if (n < 10) {
            return n;
        }
        else {
            return n % 10 + sumOfDdigits(n / 10);
        }
    }

    public static void main(String[] args) {
        System.out.println(sumOfDdigits(1850374321));
    }
}
```

MazeSolver

```
public class MazeSolver {

    public static boolean pathExists(char[][] maze, int row, int col) {
        // Base cases
        if (row < 0 || row >= maze.length || col < 0 || col >=
maze[0].length) {
            return false; // Out of bounds
        }
        if (maze[row][col] == 'D') {
            return true; // Destination found
        }
        if (maze[row][col] != '.') {
            return false; // Blocked or already visited
        }

        // Mark the current cell as visited
        maze[row][col] = 'V';

        // Recursive calls in all four directions
        boolean pathExists = pathExists(maze, row - 1, col) || // Up
                             pathExists(maze, row + 1, col) || // Down
                             pathExists(maze, row, col - 1) || // Left
                             pathExists(maze, row, col + 1);  // Right

        // restore the current cell
        maze[row][col] = '.';

        return pathExists;
    }

    public static void main(String[] args) {
        char[][] maze = {{'.','X','X','X'},
                         {'.','.','X','.'},
                         {'X','.','X','X'},
                         {'.','.','.','D'}};
    System.out.println(pathExists(maze, 0, 0));
    }

}
```

InfiniteExponent

```
public class InfiniteExponent {

    public static double countInfiniteExponents(int i) {
        if (i == 0) {
        return 1;
        }
        else {
            double x = 
Math.pow(Math.sqrt(2),countInfiniteExponents(i-1));
            System.out.println("i is: " + i);
            System.out.println("x is: " + x);
            return x;
        }
    }

    public static void main(String[] args) {
        countInfiniteExponents(40);
    }
}
```