Big-O Basics

1. Basic Order:
   $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n) < O(n!)$
   (a) n^3 __ $n^2 \log(n)$
   (b) 3^n __ $n!$
   (c) $\log(\log(n))$ __ $\log(n)$
2. Useful Rules:
   (a) Lower order terms doesn't matter. $a_0 + a_1 n + ... + a_d n^d$ is $O(n^d)$ if $a_d > 0$
      i. $n^2 + n =$ __
      ii. $2^n + n^5 + 2n =$ __
      iii. $n + \log(n) =$ __
   (b) Log base doesn't matter. $O(\log_a n) = O(\log_b n)$ for any $a, b > 0$
      i. $O(\log_2(n))$ __ $O(\log_4(n))$
   (c) Log grows slower than every polynomial. $log(n) = O(n^x)$ for any $x > 0$.
      i. $\log_(n)$ __ $n^{0.01}$
   (d) Every exponential grows faster than every polynomial. $n^d = O(r^n)$ for all $r > 1$ and all $d > 0$.
      i. n^{100} __ 1.01^n
3. Big-O Definition
   (a) Upper bound: $g(n)$ is in $O(f(n))$ if there exist positive constants c and n_0 such that $g(n) \leq c * f(n)$ for all $n \geq n_0$
      i. $g(n) = 4n, f(n) = n, c =$ ____, $n_0 =$ ____
      ii. $g(n) = 100n, f(n) = n^2, c =$ ____, $n_0 =$ ____
      iii. $g(n) = n^3, f(n) = 2^n, c =$ ____, $n_0 =$ ____
   (b) Lower bound: $g(n)$ is in $\Omega(f(n))$ if there exist positive constants c and n_0 such that $g(n) \geq c * f(n)$ for all $n \geq n_0$
   (c) Tight bound: $g(n)$ is in $\Theta(f(n))$ if: $g(n)$ is in $O(f(n))$; $g(n)$ is in $\Omega(f(n))$

Algorithm Analysis

1.
```
int min(int[] arr) {
    int min = arr[0]
  for (int i=1; i < arr.length; i++) {
    if (arr[i]<min) {
      min = arr[i]
    }
  }
  return min;
}
```

2.
```
int min(int[][] arr) {
    int min = arr[0][0]
  for (int i=1; i < arr.length; i++) {
    for (int j=0; j<arr[0].length; j++) {
      if (arr[i][j]<min) {
        min = arr[i][j]
      }
    }
  }
  return min;
}
```

3.
```
int sum (int n) {
    int sum = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < i * i * i; j++){
      sum += j;
    }
  }
  return sum;
}
```

4.
```
int factorial (int n) {
    if (n == 1) {
  return 1;
  } else {
    return factorial(n - 1) * n;
  }
}
```

5.
```
boolean find(int[]arr, int k) {
    return help(arr,k,0,arr.length);
}
boolean help(int[]arr, int k, int lo, int hi) {
  int mid = (hi+lo)/2;
  if(lo==hi) return false;
  if(arr[mid]==k) return true;
  if(arr[mid]< k) return help(arr,k,mid+1,hi);
  else return help(arr,k,lo,mid);
}
```