Homework 1

Weiss Exercise 2.1

$\frac{2}{N}$, 37, $\sqrt{N}$, N, $N\left(\log\left(\log N\right)\right)$, $N\log N$, $N\log N^2$, $N(\log N)^2$, $N^{1.5}$, N^2, $N^2\log N$, N^3, $2^{\frac{N}{2}}$, 2^N

$N\log N^2 = 2N\log N = O(N\log N)$. Hence it has the same Big-Oh as $N\log N$.

Weiss Exercise 2.6

2.6 (a)

The sequence goes 2^1, 2^2, 2^4, 2^8, $2^16\ldots$ Hence the sequence is 2^{2^x} where x is the number of days.

2.6 (b)

Let the number of days be x and the fine be F. The fine would be $F_N = 2^{2^N}$.

For the fine to reach D dollars,

$$\begin{aligned} D &= 2^{2^x} \\ \log_2 D &= 2^x \\ \log_2 \log_2 D &= x \end{aligned}$$

Big-Oh Analysis

```
int sum = 0;
for (int i = 0; i < 23; i++) // 23 times
    for (int j = 0; j < n; j++) // n times
        sum++; // constant time
```

The inner loop will run $23n$ times, which is $\Theta(n)$.

```
int sum = 0;
for (int i = 0; i < n; i++)
    for(int k=i; k < n; k++)
        sum++; // constant time
```

The inner loop will run n times for the first iteration of the outer loop, $n-1$ times for the second iteration of the outer loop, etc. The total number of iterations for the inner loop is therefore: $n + (n-1) + (n-2) + ... + 1$, i.e. the sum of integers from 1 to n. Using the sum formula for the arithmetic series, we can solve this sum as $n(n+1)/2 = \Theta(N^2)$.

```
public int foo(int x,int k) {
    if (x <= k)
        return 1; // base case: constant time
    else
        return foo(x / k, k) + 1; // constant time for addition
                                  // and return
}
```

Ignoring the recursion, any single run of `foo` takes constant time. We only need to find the total number of recursive calls. We assume that $x > k$. Since we are interested in Big-O running time, we can also assume that x is a power of k, i.e. $x = km$. In each step, we reduce x by a factor of k. Because you can do this $logx$ times, the running time is $\Theta(\log x)$. Note that the base of the logarithm is irrelevant for big-O notation since $\log_k x = \frac{\log_2 x}{\log_2 k} = \frac{1}{\log_2 k}(\log_2 x)$, and $\frac{1}{\log_2 k}$ is a constant factor.

Weiss Exercise 2.11

2.11 (a)

Let runtime be R.

$$R = kN$$

Then, given that $0.5 = k100$, $k = \frac{0.5}{100}$. Then, when $N = 500$, $R = \frac{0.5}{100}500 = 2.5$

2.11 (b)

$$R = kN \log N$$

We use natural log when we write log in this question for convenience.

Then, given that $0.5 = k100 \log 100$. $k = \frac{0.5}{100 \log 100}$ Then when $N = 500$, $R = \frac{0.5}{100 \log 100}500 \log 500 = 3.373713$

2.11 (c)

$$R = kN^2$$

Then given that $0.5 = k100^2$, $k = \frac{0.5}{100^2}$. Then when $N = 500$, $R = \frac{0.5}{100^2} * 500^2 = 12.5$

2.11 (d)

$$R = kN^3$$

Then given that $0.5 = k100^3$, $k = \frac{0.5}{100^3}$. Then when $N = 500$, $R = \frac{0.5}{100^3} * 500^3 = 62.5$

Weiss Exercise 2.15

Let's say that $A_5 > 5$. For example, $A_5 = 6$. By the condition in the question, we know that $A_5 < A_6$. That means $A_6 > 6$. Then the smallest integer possible for A_6 is 7. Then, $A_7 > A_6$ hence $A_7 > 7$. Hence, all integers after A_5 will be greater than their index (ie. $A_i > i$ for $i \geq 5$ if $A_5 > 5$).

Similarly, if $A_5 < 5$, say $A_5 = 4$, then the inverse is true. By the condition in the question, we know that $A_4 < A_5$, hence $A_4 < 4$. Then, the largest possible integer for A_4 is 3. Then, $A_3 < A_4$ and hence $A_3 < 3$. Hence all integers before A_5 in this case will be smaller than their index (ie. $A_i < i$ for $i \leq 5$ if $A_5 < 5$).

Then, let `l` represent an integer smaller than its index; let `e` represent an integer equal to its index; let `g` represent an integer larger than its index. The only possible way to sequence `l`s `e`s and `g`s are: (zero or more `l`s) followed by (zero or more `e`s) followed by (zero or more `g`s):

```
lll...llleee...eeeggg...ggg
```

This means we can apply binary search. We start with the middle integer, and we ask if the middle integer is a `g` or a `l` or an `e`. If it's an `e` we're done (remember we don't have to find all the `e`s, just if there is one `e`). If it's a `g`, then `e`s can only exist prior to the `g`, hence we search the lower half. If it's an `l`, then `e`s can only exist after the `l`, so we search the upper half. Then we recurse.

The algorithm would look like this:

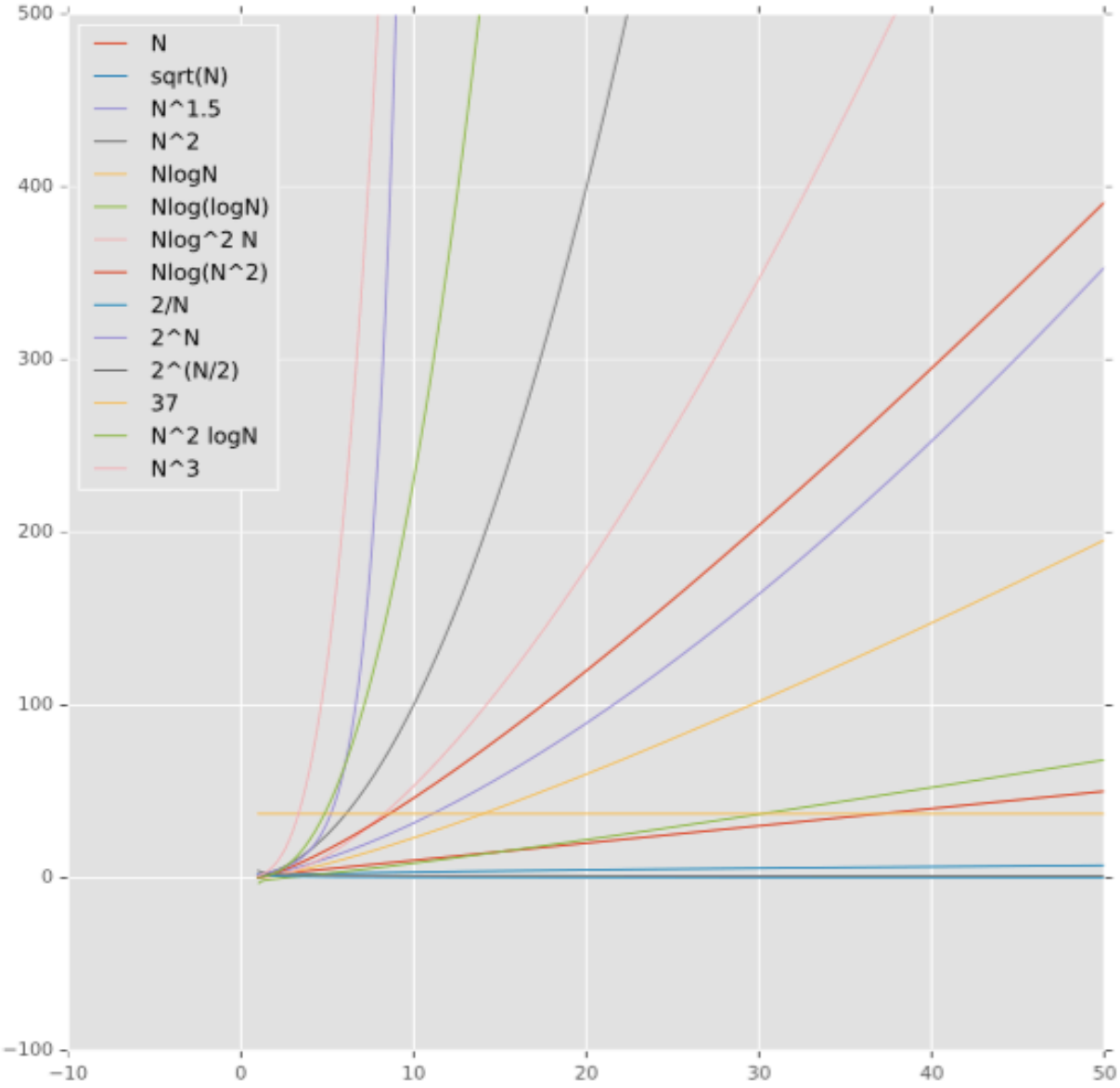
```
public static boolean contains(int[] a) {
  contains(a, 0, a.length - 1);
}
```

```
private static boolean contains(int[] a, int low, int high) {
  if (low > high) {
    return false;
  }
  int mid = (low + high) / 2;

  if(a[mid] == mid) {
    return true;
  } else if (a[mid] < mid) {
    // integer smaller than index ie. l
    // search upwards
    return contains(a, mid + 1, high);
  } else {
    // integer larger than index ie. g
    // search downwards
    return contains(a, low, mid - 1);
  }
}
```

This is almost exactly like binary search. The runtime is $O(\log N)$

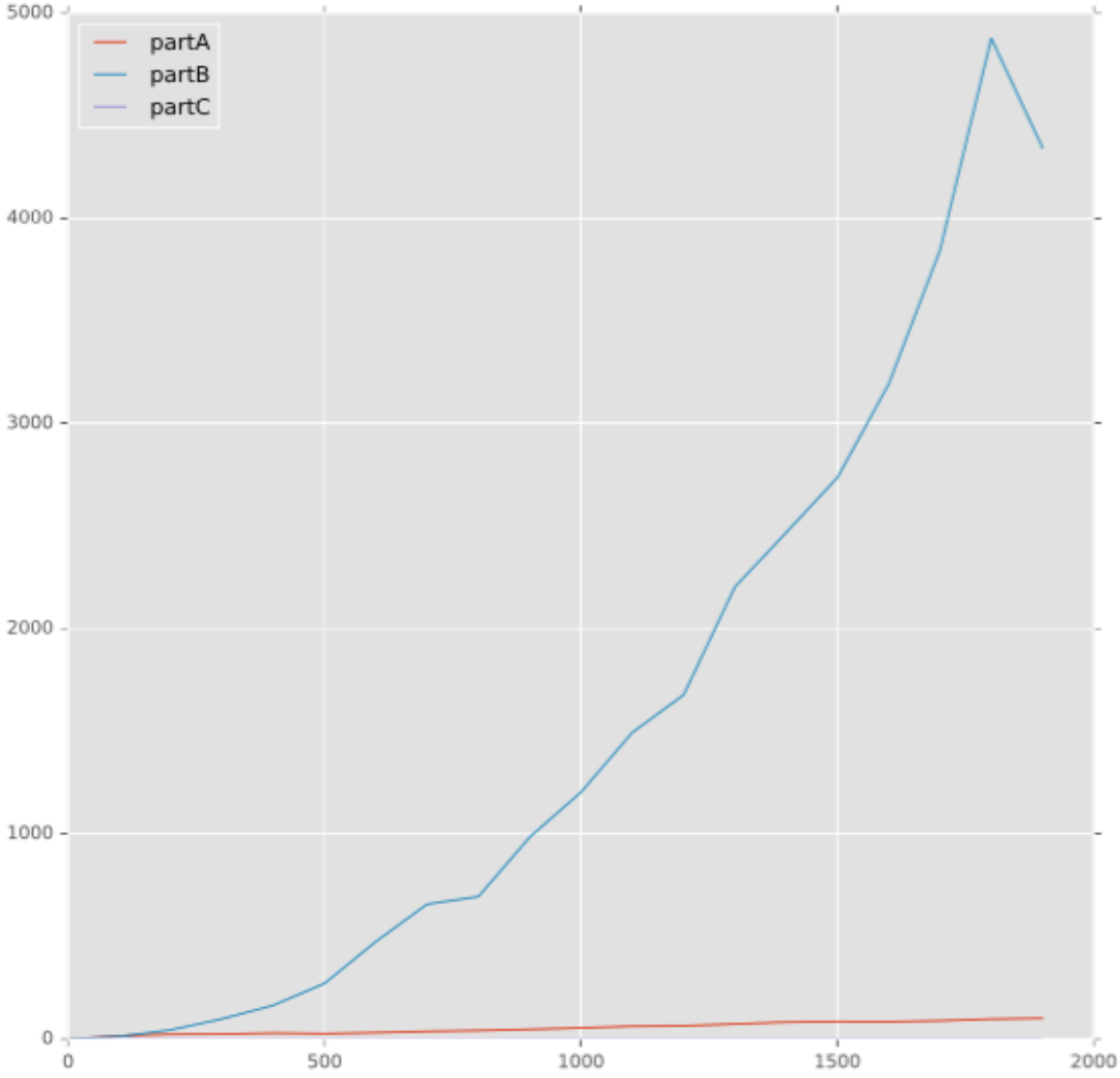
Weiss Exercise 2.1



Programming 3 Sample Output

part,n,runtime
partA,0,0
partA,100,15
partA,200,23
partA,300,24
partA,400,29
partA,500,26
partA,600,31
partA,700,37
partA,800,41
partA,900,47
partA,1000,54
partA,1100,62
partA,1200,64
partA,1300,73
partA,1400,82
partA,1500,85
partA,1600,84
partA,1700,89
partA,1800,97
partA,1900,101

partB,0,0
partB,100,13
partB,200,44
partB,300,98
partB,400,164
partB,500,271
partB,600,475
partB,700,657
partB,800,694
partB,900,984
partB,1000,1203
partB,1100,1494
partB,1200,1677
partB,1300,2203
partB,1400,2468
partB,1500,2736
partB,1600,3192
partB,1700,3844
partB,1800,4877
partB,1900,4344
partC,0,0
partC,100,0
partC,200,0
partC,300,0
partC,400,0
partC,500,0
partC,600,0
partC,700,0
partC,800,0
partC,900,0
partC,1000,0
partC,1100,0
partC,1200,0
partC,1300,0
partC,1400,0
partC,1500,0
partC,1600,0
partC,1700,0
partC,1800,0
partC,1900,0


partA
partB
partC
5000
4000
3000
2000
1000
0
0
500
1000
1500
2000