Flip Flop

Here is a video walkthrough of the solutions.

Suppose we have the `flip` function as defined below. Assume the method `unknown` returns a random `integer` between `1` and `N`, exclusive, and runs in constant time. For each definition of the `flop` method below, give the best and worst case runtime of `flip` in $\Theta(.)$ notation as a function of `N`.

```java
public static void flip(int N) {
    if (N <= 100) {
        return;
    }
    int stop = unknown(N);
    for (int i = 1; i < N; i++) {
        if (i == stop) {
            flop(i, N);
            return;
        }
    }
}
```

(a)
```java
public static void flop(int i, int N) {
    flip(N - i);
}
```

Best Case: $\Theta(\quad)$, Worst Case: $\Theta(\quad)$

Solution:
Best Case: $\Theta(N)$, Worst Case: $\Theta(N)$
Explanation: Consider some arbitrary value of `stop`. When `stop = x`, we do `x` work inside of `flip` (the `for` loop) and recursively call `flip(N - x)` through `flop`. This results in a total of `N / x` calls before reaching our base case, and `x` work per call, for a total of $\Theta(N)$ work. Note that this holds for any value of `x`, so our best and worst case are the same.

(b)
```java
public static void flop(int i, int N) {
    int minimum = Math.min(i, N - i);
    flip(minimum);
    flip(minimum);
}
```

Best Case: $\Theta(\quad)$, Worst Case: $\Theta(\quad)$

Solution:
Best Case: $\Theta(1)$, Worst Case: $\Theta(N \log(N))$
Explanation: In the best case, `stop = 1`. This hits the base case immediately, so we make 2 calls to `flip` then stop for $\Theta(1)$ work.
In the worst case, `stop = N / 2`. This results in `flip` making 2 recursive calls

to itself with the argument `N / 2`. Note the similarity of this recurrence and mergesort; the runtime is the same $\Theta(N \log N)$.

(c)
```java
public static void flop(int i, int N) {
    flip(i);
    flip(N - i);
}
```

Best Case: $\Theta(\quad)$, Worst Case: $\Theta(\quad)$

Solution:
Best Case: $\Theta(N)$, Worst Case: $\Theta(N^2)$
Explanation: In the best case, suppose `stop = 1`. Then `flip(N)` makes recursive calls to `flip(1)` and `flip(N - 1)`, the first of which terminates immediately in the base case. `flip(N - 1)` then calls `flip(1)` and `flip(N - 2)`. The pattern is a linear recursion: constant work per call, N calls total for $\Theta(N)$ work.
In the worst case, suppose `stop = N - 1`. Note that this case is symmetrical to the best case in terms of recursive calls; however we do work proportional to `N` inside of `flip` each time because of the `for` loop. The overall work is $(N-1) + (N-2) + (N-3) + ... + 2 + 1 = \Theta(N^2)$.