# 34.5 Exercises

## Factual

1. Which of the following function(s) have the slowest order of growth in terms of Big Theta?
   - [ ] $N \log N$
   - [ ] $\log N!$
   - [ ] $N^2$
   - [ ] $N! \log N!$
2. To solve puppy, cat, dog for 12 items, what is the theoretical minimum number of comparisons we have to make, based on the argument used in lecture? Please round your answer up to the nearest whole number.
3. Which of the following statements are true?
   - [ ] The optimal sorting algorithm takes at most $\Theta(N \log N)$ time.
   - [ ] It is impossible to find a comparison-based sorting algorithm that takes less than $\Theta(N \log N)$ comparisons.
   - [ ] It is impossible to find a comparison-based sorting algorithm that takes less than $\Theta(N \log N)$ time.
   - [ ] It is impossible to find a sorting algorithm that takes less than $\Theta(N \log N)$ time.

Problem 1

In lecture, we proved that $\log N! \in \Theta(N \log N)$. Thus, both $N \log N$ and $\log N!$ have the same order of growth, and are slower than $N^2$ or $N! \log N!$.

- [x] $N \log N$
- [x] $\log N!$
- [ ] $N^2$

- [ ] $N! \log N!$

<details>
<summary>Problem 2</summary>

$ceil(\log_2 12!) = 29$, based on the equation we saw in lecture.
</details>

<details>
<summary>Problem 3</summary>

- [x] **The optimal sorting algorithm takes at most $\Theta(N \log N)$ time**. We know of several sorts (for example, mergesort) that take $\Theta(N \log N)$ time in the worst case. So the hypothetical "best" sorting algorithm cannot be worse than this.
- [x] **It is impossible to find a comparison-based sorting algorithm that takes less than $\Theta(N \log N)$ comparisons.** This is the comparison sorting bound proved in lecture.
- [x] **It is impossible to find a comparison-based sorting algorithm that takes less than $\Theta(N \log N)$ time**. Each comparison must take at least constant time, so the time complexity of comparison-based sorts has the same lower bound as the number of comparisons.
- [ ] **It is impossible to find a sorting algorithm that takes less than $\Theta(N \log N)$ time.** The lower bound proved in lecture only applies to comparison-based sorts. We will see in future lectures how to use non-comparison sorts to reduce this runtime even further.
</details>

# Metacognitive

1. Suppose we add a new method to `Arrays.sort` that takes in an array of strings. What algorithm should `Arrays.sort(String[] x)` use?

<details>
<summary>Problem 1</summary>

Quicksort. We don't need stability, since strings are only equal by `.equals` if they are exactly the same. Quicksort, then, is the best algorithm since it is
</details>

empirically the fastest.

Previous
34.4 Summary

Next
35. Radix Sorts

Last updated 1 year ago