1 Results

1. Explain what choices you made when running your experiments in Section 3.2. What values of \( n \) did you record? Why did you make these choices? How many replications did you run for each experiment?

I ran my experiments for values of \( n \) up to \( n = 1000000 \). This is large enough that I was able to see the asymptotic trends in the graph, but small enough that the experiments didn’t take forever to run. The specific values of \( n \) that I chose were: 10, 16, 32, 50, 64, 100, 128, 256, 500, 512, 1000, 1024, 2048, 4096, 5000, 8192, 10000, 16384, 32768, 50000, 65536, 100000, 131072, 262144, 500000, 524288, 1000000. (For the sorted array and unsorted array, I only went up to 500000 to save time). I used a mix of powers of 2 and non-powers of 2. This is because the add methods all call resize on powers of 2 (and do not call resize on non-powers of 2) and we want to be sure that our results capture both cases. I ran 10 replications of each experiment.

2. Graph the results you obtained in your experiments. You should have a separate graph for each implementation (sorted array, unsorted array, and heap). The \( x \)-axis should be \( n \) (the number of elements in the priority queue) and the \( y \)-axis should be the average time per operation in milliseconds. Each graph should include two data sets, one for the add operations and one for the remove operations.

![Heap Graph]

It’s sort of hard to see what is going on with the heap with this \( y \)-axis, so here is another zoomed-in version:
Heap - zoomed in

Sorted Array

Unsorted Array
3. Write a short (~1-2 paragraph) discussion of your observations. Your response should address at least the following questions:

- In general, is the asymptotic analysis a good predictor of the shape of the empirical runtime results?
- Does the asymptotic analysis give you the full picture of the empirical runtimes? What do you learn from the graphs that you don’t learn from the asymptotic analysis? What do you learn from the asymptotic analysis without having to run any experiments?
- In what cases (which implementations and operations) does the worst-case asymptotic analysis give a tight upper bound for all $n$? When does it not? Why not?

Feel free to note any other interesting observations that you made about the results!

In general, the asymptotic analysis is a pretty good predictor of the shape of the empirical results. The asymptotic analysis tells us that for the sorted array, `add` is $O(n)$ and and `remove` is $O(1)$, and sure enough this is what the graph shows. Similarly, for the unsorted array the empirical results for `remove` look linear, and for `add` it looks like there is a linear upper bound (when we have to resize the array). And for the heap, both `add` and `remove` look something like $O(\log n)$, though again with `add` we have the linear shape on the powers of 2.

Without having to run any experiments, the asymptotic analysis tells us that as $n$ gets very large, the heap probably will have better performance than the sorted array for `add` and probably will have better performance than the unsorted array for `remove`. Sure enough, this turns out to be true. However, this doesn’t tell the full story. For example, the graphs give us information about the constant factors associated with each operation; we can’t learn this from the asymptotic analysis. We can see in the graph, for example, that the constant factor associated with `add` in the sorted array turns out to be smaller than the constant factor associated with `remove` in the unsorted array.

The worst-case asymptotic analysis doesn’t tell us the full story for the `add` operation for the unsorted array and the heap. In the worst case, for both implementations `add` takes time $O(n)$. However, this worst-case runtime only occurs on powers of 2, when the underlying array needs to be resized. Most of the time, `add` takes times $O(1)$ in the unsorted array and $O(\log n)$ in the heap. Our empirical results show this. This is one advantage of using the amortized analysis for array expansion that we discussed in class: while the worst-case cost of resizing is $O(n)$, we saw that the amortized cost over a sequence of $n$ operations is $O(1)$. 