Naive Solution

Suppose that we want merge two sets a and b of sizes n and m, respectively. One possiblility is the following:

```cpp
for (int x : b) a.insert(x);
```

which runs in $\mathcal{O}(m \log(n+m))$ time, yielding a runtime of $\mathcal{O}(N^2 \log N)$ in the worst case. If we instead maintain a and b as sorted vectors, we can merge them in $\mathcal{O}(n+m)$ time, but $\mathcal{O}(N^2)$ is also too slow.

Better Solution

With just one additional line of code, we can significantly speed this up.

```cpp
if (a.size() < b.size()) swap(a, b);
for (int x : b) a.insert(x);
```

Note that swap exchanges two sets in $\mathcal{O}(1)$ time. Thus, merging a smaller set of size m into the larger one of size n takes $\mathcal{O}(m \log n)$ time.

Claim: The solution runs in $\mathcal{O}(N \log^2 N)$ time.

Proof: When merging two sets, you move from the smaller set to the larger set. If the size of the smaller set is X, then the size of the resulting set is at least $2X$. Thus, an element that has been moved Y times will be in a set of size at least 2^Y, and since the maximum size of a set is N (the root), each element will be moved at most $\mathcal{O}(\log N)$ times.

3 This is in fact $\mathcal{O}(n)$. Proof: When merging a shallower subtree, mark each node in a chain that goes to the deepest leaf in the subtree as used. Since the deepest subtree is never merged to anything else, by induction, there will remain an unmarked chain to one of the deepest leaves. The amount of marked chains is proportional to the amount of work done while merging, thus linear in the amount of nodes since each node is marked at most once. – ollpu Jul 20, 2020 at 9:45