CIS 575. Introduction to Algorithm Analysis
Material for January 24, 2024

Worst-Case vs Best-Case Analysis

The topic of this note in covered in *Cormen*'s Section 3.2 (p.56–57).

1 Estimating Running Times

In the rest of this course, we shall often describe the running time (or space use) of an algorithm using Big-O, Big-Omega, or Big-Theta notation, and say for example that a given algorithm runs in time $\Theta(n^2)$. We shall now clarify what this means; there are two interpretations, described in the following and illustrated using the insertion sort algorithm.

1.1 Worst-Case Interpretation

Big-O We say that an algorithm runs in time $\mathbf{O(g(n))}$ iff there exists a function f such that

- for all n, and **all** input of size n, the algorithm runs in time at most $f(n)$
- $f \in O(g)$.

Thus $g(n)$ is (in the long run, disregarding constants) an upper bound for how the algorithm may behave on input of size n.

For example, Insertion Sort runs in time $O(n^2)$ but it also correct (though potentially misleading) to say that it runs in time $O(n^3)$, $O(n^4)$, etc.

Big-Omega We say that an algorithm runs in time $\mathbf{\Omega(g(n))}$ iff there exists a function f such that

- for all n, there **exists** input of size n, such that the algorithm runs in time at least $f(n)$
- $f \in \Omega(g)$.

Thus $g(n)$ is (in the long run, disregarding constants) a lower bound for how the algorithm may behave for certain input of size n.

For example, Insertion Sort runs in time $\Omega(n^2)$ since for each n there will exist input of size n (an array sorted in reverse order) that causes a running time proportional to n^2.

Big-Theta We say that an algorithm runs in time $\Theta(g(n))$ iff if runs in time $O(g(n))$ and in time $\Omega(g(n))$.

From our previous results, we see that with the worst-case interpretation, Insertion Sort runs in time $\Theta(n^2)$.

1.2 Best-Case Interpretation

Big-O We say that an algorithm runs in time $\mathbf{O(g(n))}$ iff there exists a function f such that

- for all n, there **exists** input of size n, such that the algorithm runs in time at most $f(n)$
- $f \in O(g)$.

Thus $g(n)$ is (in the long run, disregarding constants) an upper bound for how the algorithm may behave for certain input of size n.

For example, Insertion Sort runs in time $O(n)$ since for each n there will exist input of size n (an array that is already sorted in the desired order) that causes a running time proportional to n.

Big-Omega We say that an algorithm runs in time $\mathbf{\Omega(g(n))}$ iff there exists a function f such that

- for all n, and **all** input of size n, the algorithm runs in time at least $f(n)$
- $f \in \Omega(g)$.

Thus $g(n)$ is (in the long run, disregarding constants) a lower bound for how the algorithm may behave on input of size n.

For example, Insertion Sort runs in time $\Omega(n)$ since it can never have a running time that is less than proportional to n.

Big-Theta We say that an algorithm runs in time $\Theta(g(n))$ iff if runs in time $O(g(n))$ and in time $\Omega(g(n))$.

From our previous results, we see that with the best-case interpretation, Insertion Sort runs in time $\Theta(n)$.

1.3 Conclusion

For many algorithms, the running time depends solely on the size of the input; in that case, the two interpretations will coincide.

In general, the "worst case" interpretation will be our default, that is the one we shall use unless we explicitly state otherwise.