UM Software Engineering Documentation

Release 1.0.0

Joshua Zahner, Evan Miller

Aug 23, 2018

Contents:

CHAPTER 1

Algorithm Runtimes and Runtime Analysis

Algorithm efficiency is often described in terms of the runtime of the algorithm. It is often important for algorithms to run as fast possible so as to minimize the amount time the computer spends working on any one task. Because of this need to efficient algorithms, many computer scientists have spent a great deal of time devising different ways means of completeing the same operation, but with different runtimes.

A classic example of a problem that has many different implementations, and many different runtimes, is the problem of sorting. To sort an array of integers, you can choose to use the selection sort algorithm, the insertion sort algorithm, bubble sort, merge sort, quick sort, or a variety of other types of sorts. Why are there so many sorting algorthms though? Its because many of them have different runtimes. For example, selection sort, insertion sort, and bubble sort all run in $O(n^2)$ time (more on what this notation means later), while merge sort and quick sort run in O(nlg(n)) time.

It is important to understand the different runtime complexity classes for algorithms, and be able to determine the runtime of a given algorithm.

1.1 Big-O Notation

Typically, algorithm run times are expressed in "Big-O" notation (this is the notation used above). In Big-O notation, the runtime of the algorithm is expressed as a mathenatical function in terms the input size, *n*. The faster the mathematical function grows, the larger the runtime of the algorithm, and, often, the less desirable the algorithm is. Below are a few examples of Big-O notation:

O(c) - constant run time; The algorithm's run time does not change regardless of the input size

O(n) - linear run time; The algorithm's runtime grows linearly with the input size. An input that is 10x larger than another input will take 10x as long to run.

$O(n^2)$ - quadratic run time; the algorithm's runtime grows quadratically with the input size. An input that is 10x large than another input will take 100x as long to run.

O(lg(n)) - logarithmic run time; The algorithm's runtime grows with the lg of the input size.

O(nlg(n)) - polylogarithmic run time; The algorithm's runtime grows with the polylogarithm of the input size

There are, of course, other possible run times, but the above four are the most common ones will you see.

1.2 Related Runtime Notation

We call the above runtime evaluation syntax, Big-O notation. This is not technically correct. In addition to Big-O notation, there is also, Big-Θ notation, as well as Big-Ω notation. In a technical sense, an algorithm's Big-O run time constitutes an upper bound on the run time of the algorithm, Big-Ω run time constitutes a lower bound on the run time, and Big-Θ runtime constitutes a tight bound on the run time.

In other words, any function larger than the true algorithmic run time can be said to be Big-O and any function smaller than true run time could be said to be Big-Ω . Traditionally, when we talk about algorithmic runtimes, we are refering to the tightly bounded Big-Θ runtime.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

The University of Miami's Software Engineering club is joint effort between the ACM and IEEE club chapters to support UM computer scientists, software engineers, programmers, and anyone who wants to learn more about the programming or software industries. We meet twice a week during the semester to teach classes about programming concepts, data structures, and algorithms, as well as present about interesting topics in the computing and software industries, such as databases, parallel and high performance computing, and quantum computing.

We also encourage our members to actively attend and compete at hackathons in Florida and around the country. Hackathons are a great opportunity for new programmers and developers to build software projects over the course of a weekend, while also networking with other developers and companies from around the country.

CHAPTER 3

Meeting Times and Locations

Regular meetings will occur twice week as specified below.

Tuesday: 7:00-8:30 - Introduction to Programming (MEA 202)

Thursday: 7:00-8:30 - Advanced Programming Concepts (MEA 202)

There may also be extra meetings to discuss projects, hackathons, or other club events. Guest speakers or company presentations may also be scheduled at other times throughout the week. Join our Facebook group to stay up to date with club meetings and presentations.

CHAPTER 4

Hackathons

We strongly encourage members to attend at least one hackathon each year. Hackathons are great places to meet recruiters from top tech companies as well as other developers from around Florida and the US. Hackathons take place all over the country and all throughout the year, so there are plenty of opportunities for you to attend if interested. Below are the dates and locations of a few local hackathons:

Event Name	School	Date	Notes
ShellHacks	FIU, Miami	Sept. 14	Registration closes on September 5
HackGT	Georgia Tech, Atlanta	Oct. 21	Registration closes on September 22
UHack	University of Miami	Dec. 1	Registration open in October

For hackathon events outside of the city of Miami, travel assistance is often available from the hackathon event itself. The club also has limited funding alotted for sending members to hackathons around the country. If your interested in attending hackathons with us, stay tuned for more information.

More hackathon events can be found at the MLH website.

CHAPTER 5

UHack

UHack is the University of Miami's Official Hackathon event. It is organized and run by our club for UM students and other students from around the country. UHack is presently scheduled to occur **December 1-2** at the Shalala Center. If you are interested in helping organize UHack please contact one of our EBoard members.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search