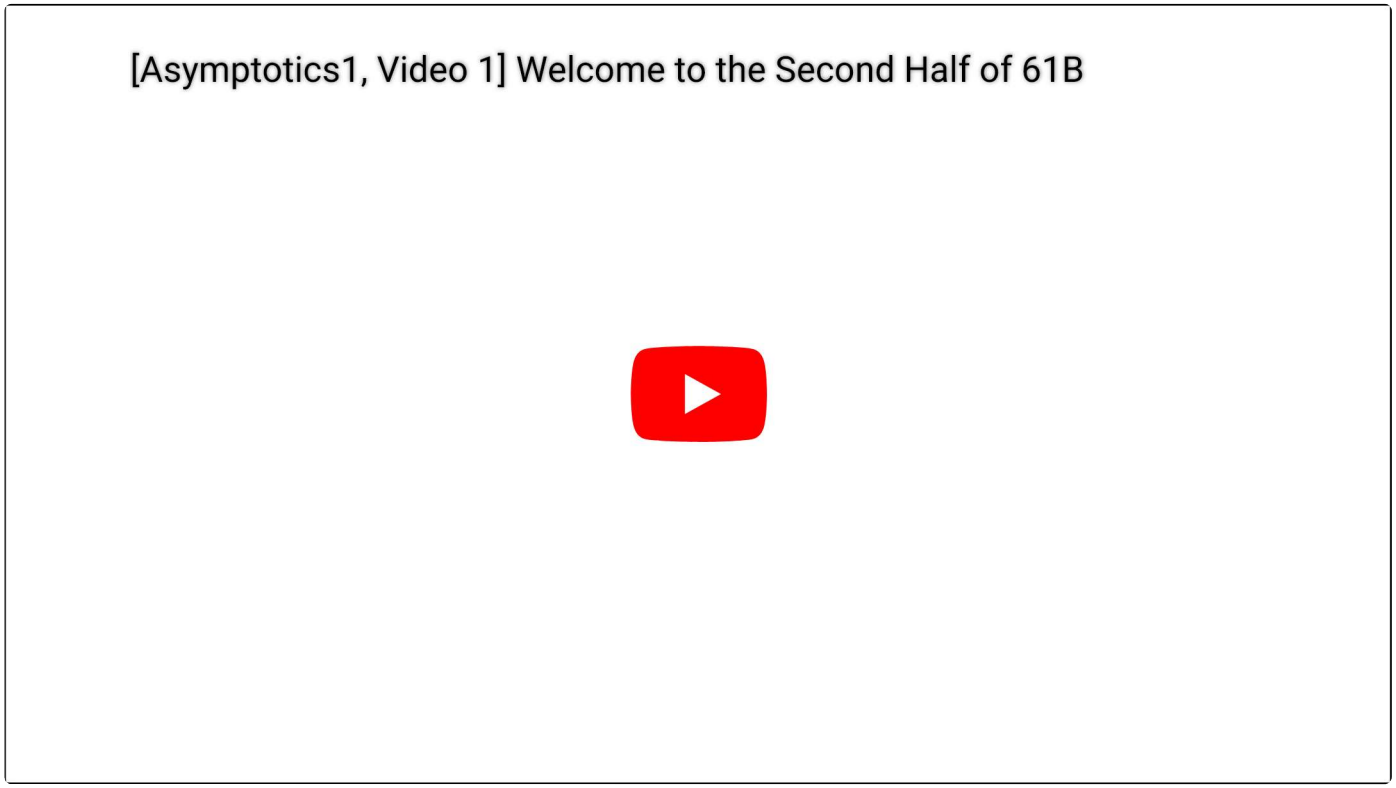
13.1 An Introduction to Asymptotic Analysis

We always have to start somewhere.



Previously, we have focused on how to save time *writing* the program. Now, we will learn how to make the best use of our computer's time and memory.

We can consider the process of writing efficient programs from two different perspectives:

1. Programming Cost *(everything in the course up to this date)*
    1. How long does it take for you to develop your programs?
    2. How easy is it to read or modify your code?
    3. How maintainable is your code? (very important — much of the cost comes from maintenance and scalability, not development!)
2. Execution Cost *(everything in the course from this point on)*
    1. **Time complexity**: How much time does it take for your program to execute?

2. **Space complexity**: How much memory does your program require?

To give a sense of what is coming up, consider a **sorted** array. Our goal is to determine if there is a duplicate element in the list.

```
List<Integer> example = [-3, -1, 2, 4, 4, 8, 10, 12];
```

A **naïve algorithm** would be to compare every pair of elements. In the above example, we would compare -3 with every element in the list, then -1, then 2, etc.

A **better algorithm** would be to take advantage of the sorted nature of the list! Instead of comparing every pair of elements, we can compare each element with just the element next to it.

We can see that the **naïve algorithm** seems like it's doing a lot more unnecessary, redundant work than the **better algorithm**. But how much more work is it doing? How do we quantify how efficient a program is? This chapter will provide you the formal techniques and tools to compare the efficiency of various algorithms!

Previous
13. Asymptotics I

Next
13.2 Runtime Characterization

Last updated 1 year ago