Timing

Can define the total time, and then the best-case, worst-case, and average-case times, for a program and the set of inputs.

Complexity Analysis Techniques

Could try running a program on all possible inputs of a certain size to determine the running time of the program. However, there are too many inputs for this.

Instead, we could try random sampling, find the average running time, and use that an approximation of the overall average. We would have to do this on many samples to get a consistent result, and even then we're not sure it's a representative result.

Comparison Time

To simplify, you can count some steps carried out by an algorithm, instead of all of them. Doing this we can make the analysis as machine-independent as possible, and then form machine-specific results from this result.

We're gonna look at sorting algorithms. One of the main options is comparisons, so we usually focus on that (even if assignments or swaps are happening, typically they're the result of a comparison).

The comparison time of an algorithm on an input L is defined to be the total number of comparisons A makes in computing the output for L.

Constant Time Algorithms

These take the same amount of time on all inputs of a fixed size. We're gonna start by looking at these, because they're easy to analyse.

For these algorithms, the best case, worst case, and average case times are all the same.

Two examples are Bubble Sort and Selection Sort.

Bubble Sort

On the first run, we look at n-1 pairs (compare each element to the one after it in the list), on the second we look at n-2, and so on. The total number of comparisons is $1 + 2 + \ldots + (n - 1)$, which is equal to $(n)(n - 1) / 2$.

This result is O(n^2), which is typically the case when loops are nested two-deep. This won't always be the case, but it often is.

Selection Sort

By a similar analysis, selection sort also comes out as (n)(n - 1) / 2.

Non-Constant Time Algorithms

Many algorithm don't run in constant time. To analyse these, it can help to identify groups of inputs (constant time groups) on which the algorithm does have constant time. This simplifies the average-case analysis of these algorithms quite a bit.

The total time is the time for each group multiplied by the size of the group.

This makes the analysis easier if the constant time groups are all the same size, which is quite common. Then the average time is the group size multiplied by the sum of the constants, divided by [...].

Example: Sequential Search

- assume the target is always in the list
- assume all elements in the list are distinct

How many possible lists are there with the target in the first position? (E.g. what is the size of the group which only requires 1 comparison.)

Timing Measures

Clock-based Timing

Basically, use a clock to time an input.

There are two problems with this:

- it's machine-dependent
- it's input dependent

Static Time Analysis

Assign a time to unit to each "basic step", and count those.

For comparison-based algorithms (algorithms where every operation is based on a preceding comparison), we look only at comparisons when counting steps.

Size-based Measures

Focus on groups of inputs of the same size.

Repeated Elements

To account for this, we assign tie-breakers (a random permutation of distinct values is assigned to the list, and we use those when two elements are the same).