Getting Started

Devere Anthony Weaver

---

1 Insertion sort

*** Refer to the text for the actual algorithm pseudocode**

Loop invariant - a property about a loop that is true before the execution of the loop and is true after each successive iteration

Loop invariant are needed to understand why the insertion sort algorithm is correct.

- **Initialization** - the invariant is true prior to the first iteration (similar to base case)
- **Maintenance** - if it's true before the first iteration, it remains true for the next iteration (similar to inductive step)
- **Termination** - when the loop terminates, the invariant gives a useful property that helps show the algorithm is correct (show correctness and stop "induction")

---

2 Analyzing Algorithms

Analyzing an algorithm is predicting the resources the algorithm requires (e.g. memory, communication bandwidth, computational time, etc.)

In order to analyze algorithms, a computational model must be chosen that will represent the machine the algorithm will execute on. The computational model to be used is known as the **random-access machine (RAM)**. The assumptions using this model are as follows:

- assume each instruction executes one after another
- assume each instruction takes the same amount of time as other instructions and data accesses
- the model contains the same commonly found instructions on real computers (arithmetic, movement, control, etc.)
- data types are integer, floating point, and character
- assume each word of data has a limit on the number of bits
- the model doesn't account for memory hierarchy and virtual memory

Running time for many algorithms depends on the input. **Input size** is a common characteristic of the data that is used when analyzing algorithm runtime. The notion of input size depends on the problem being studied, so for example the input size can mean the total number of items in the input, the total number of bits, or it can even be more than one of these values.

The **running time** of an algorithm on a particular input is the number of instructions and data accesses executed.

The **worst-case running time** is often the most interesting goal of the analysis. The worst-case running time is the longest running time of any input of size n. Some reason include:

- the worst-case running time of an algorithm gives an upper bound on the running time for any input
- for some algorithms, the worst case occurs fairly often
- the "average case" is often roughly as bad as the worst case

The **rate of growth** or **order of growth** describes how an algorithm's runtime scales with the size of the algorithm's input. Since we are often concerned with the worst-case runtime mentioned above, we can abstract away most of the complicated formula and use the highest-order term to describe the rate of growth, ignoring leading coefficients and lower-order terms.

For now, we can use theta-notation to highlight the order of growth. For example $\theta(n^2)$ means the algorithm's rate of growth is "roughly proportional to n^2 when n is large".

---

3 Designing Algorithms

Many algorithms are **recursive** meaning to solve a given problem, the recurse one or more times to handles closely related subproblems.

These types of algorithms follow the **divide-and-conquer** method by breaking the problem into several smaller subproblems that are similar to the original problem. The subproblems are themselves solved recursively and then their solutions are combined to create a solution to the original problem.

The three characteristic steps are:

1. **Divide** the problem into one or more subproblems that are smaller instances of the same problem.
2. **Conquer** the subproblems by solving them recursively.
3. **Combine** the subproblem solutions to form a solution to the original problem.

Note that for this section the **merge sort** algorithm was used in order illustrate this method. The merge sort algorithm was used to demonstrate using recursion to solve and smaller exact subproblems.

When an algorithm contains a recursive call, you can often describe its running time by a **recurrence equation**, which describes the overall running time on a problem of size n in terms of the running time of the same algorithm on smaller inputs.

Note: The majority of this section was used to analyze merge sort, so consult the text for the technical specifics of this analysis.