Runtime Analysis

Constant-time operations:

For primitive types:
- arithmetic
- comparison
- assignment

For arrays:
- indexing
- .length

For pointers:
- comparison
- dereference

Function calls
Branching

Each such operation has an upper-bound $c_1$ and lower-bound $c_2$ such that $c_2 \leq t \leq c_1$.

Beware: some simple-looking operations are not constant-time, such as string concatenation or a split in Python.
int max (int[] a) {
    int m = Integer.MAX_VALUE;
    for (int i=0; i<a.length; ++i) {
        if (a[i] > m)
            m = a[i];
    }
    return m;
}

Let's consider the cost of this function on a worst-case input, one in which the then clause is taken every time.

In this case, the function consists of many constant-time steps, some of which are inside a loop that runs \( n \) times. It is straightforward, by adding individual constant costs, to see that there are constants \( c_1 \) and \( c_2 \) such that

\[ t \geq n c_1 + c_2 \]

and that there are constants \( c_3 \) and \( c_4 \) such that

\[ n c_3 + c_4 \geq t \]

So we have established upper and lower bounds on the worst-case running time. Both are linear in the size of the input, though with different constants.

Now consider a best-case input, one where the then clause is only used once. The very same analysis holds with different constants. We have linear upper and lower bounds in the best case, too.
public void insertionSort (int[] a) {
    int n = a.length;
    for (int i = 1; i < n; ++i) {
        int val = a[i];
        int j = i;
        while (j > 0 && val < a[j-1]) {
            a[j] = a[j-1];
            j--;
        }
        a[j] = val;
    }
}
Worst-case input v. best-case input

Upper bounds on running time v. lower bounds

Let's find an upper bound on the time for insertion sort:

Body of inner loop: \( t \leq C_1 \)

Body of outer loop: \( t \leq i \cdot C_1' + C_2 \)

Whole method:
\[
    t \leq C_3 + \sum_{i=1}^{n-1} (i \cdot C_1' + C_2')
\]
\[
    \leq C_3 + C_1' \cdot \sum_{i=1}^{n-1} i + (n-1)C_2'
\]
\[
    \leq C_3 + C_1' \cdot n(n-1) + (n-1)C_2'
\]
Similarly, for different constants
\[ t \geq C_3 + \sum_{i=1}^{n-1} iC_1 + (n-1)C_2 \]
\[ \geq C_3 + C_1 \frac{n}{3} \cdot \frac{n}{3} + (n-1)C_2 \]

We have established upper and lower bounds on worst-case running time, both of which are quadratic in the length of the array.

For best-case inputs, the body of the inner loop never runs, so there is effect only one loop. We have linear upper and lower bounds on running time.
Definition

\[ f(n) \in O(g(n)) \iff \exists c, n_0 > 0 \text{ s.t. } \forall n \geq n_0, \quad f(n) \leq c \cdot g(n) \]

"at most proportional to"

\[ 3n^2 \in O(n^2) \quad n_0 = 1, \ c = 3 \]

\[ 3n^2 \in O(n^2) \quad n_0 = 1, \ c = 3 \]

\[ 3n^2 + n \log n \in O(n^2) \quad n_0 = 1, \ c = 4 \]
Define \( F(n) \in \Omega(g(n)) \) if \( \exists c, n_o > 0 \)

s.t. \( \forall n \geq n_o, \ F(n) \geq c \cdot g(n) \)

\[
\frac{n^2}{4} \in \Omega(n^2) \quad n_o = 1 \quad c = \frac{1}{14}
\]

\[
\frac{n^2}{4} + n \in \Omega(n^2) \quad n_o = 1 \quad c = \frac{1}{14}
\]

\[
\frac{n^2}{4} - n \in \Omega(n^2) \quad n_o = 2 \quad c = \frac{1}{14}
\]

\[
(\forall n \geq 2, \quad n^2 - n \geq n^2 - \frac{n^2}{2} = \frac{n^2}{2})
\]
An easy argument that insertion sort has worst-time $\Theta(n^2)$.

- The inner loop always runs at most $n$ times, so it's clearly $O(n^2)$.

- For half $(n/2)$ of the iterations of the outer loop, at least $n/2$ elements would need to be shifted in the worst case. That's $n^2/4$ shifts, meaning that the running time is $\Omega(n^2)$. 

Definition:

$F(n) \in \Theta(g(n))$ if $\exists n_0, c_1, c_2 > 0$ s.t. $\forall n \geq n_0$, $c_1 g(n) \leq F(n) \leq c_2 g(n)$.

(i.e., $F(n) \in O(g(n))$ and $F(n) \in \Omega(g(n))$)
\[ n^2 - 40n + 100 \leq O(n^2) \]

**Proof**

Let \( F(n) = n^2 - 40n + 100 \). We need to show that \( \exists n_0, c > 0 \) such that \( \forall n \geq n_0, F(n) \leq cn^2 \).

This holds if \( n_0 = 1 \) and \( c = 101 \), because \( \forall n \geq 1 \),

\[
F(n) = n^2 - 40n + 100 \leq n^2 + 100 \\
\leq n^2 + 100n^2 \\
\leq 101n^2
\]

**Strategy:** pick \( n_0 \) first & see what \( c \) works.
**Theorem**

\[ n^2 - 40n + 100 \in \mathbb{O}(n^2) \]

**Proof**

Let \( f(n) = n^2 - 40n + 100 \). We need to show that \( \exists n_0, c > 0 \) s.t.

\[ \forall n \geq n_0, \quad f(n) \geq cn^2. \]

This holds if \( n_0 = 80 \) and \( c = \frac{1}{2} \), because \( \forall n \geq 80 \),

\[
\begin{align*}
f(n) &= n^2 - 40n + 100 \\
&\geq n^2 - 40n \\
&= n(n - 40) \\
&\geq n(n - \frac{n}{2}) = \frac{n^2}{2}.
\end{align*}
\]

**Strategy:** Let’s guess that \( c = \frac{1}{2} \).

Is it true that:

\[
\begin{align*}
n^2 - 40n + 100 &\geq \frac{n^2}{2} \\
n^2 - 40n &\geq \frac{n^2}{2} \\
n - 40 &\geq \frac{n}{2} \\
\frac{n}{2} &\geq 40
\end{align*}
\]

Yes, for all \( n \geq 80 \).
Running times

Examples

\( \Theta(n) \)

\( \Theta(n \log n) \)

\( \Theta(1) \)

\( \Theta(\log \log n) \)

\( \Theta(m \log n) \)

Limit test

If \( \lim_{n \to a} \frac{f(n)}{g(n)} \) exists and

- \( c > 0 \):
  \( f(n) \in \Theta(g(n)) \)

- \( c = 0 \):
  \( f(n) \in O(g(n)) \)

- \( c = \infty \):
  \( f(n) \in \Omega(g(n)) \)

- \( f(n) \in \omega(g(n)) \)
Let's compare
\[ f(n) = n \log n \]
and
\[ g(n) = n \sqrt{n}. \]

\[
\lim_{n \to \infty} \frac{n \log n}{n \sqrt{n}} = \lim_{n \to \infty} \frac{\log n}{\sqrt{n}} = \lim_{n \to \infty} \frac{1/n}{\frac{1}{2}n^{-\frac{1}{2}}} = \lim_{n \to \infty} \frac{2 \sqrt{n}}{n} = 0
\]

So \( f(n) \in o(g(n)) \)

Suppose
\[ f(n) = n^{1000} \]
and
\[ g(n) = 2^n. \]

\[
\lim_{n \to \infty} \frac{n^{1000}}{2^n} = \lim_{n \to \infty} \frac{n^{1000}}{2^n} = \lim_{n \to \infty} \frac{(\log x)^{1000}}{x}
\]

This clearly equals zero,
so \( f(n) \in o(g(n)) \).
Bonus notes, not part of today's class.
Reminder:
\[ f(n) = O(g(n)) \iff \exists \ n_0, c > 0 \text{ s.t. } \forall n \geq n_0, \]
\[ f(n) \leq c \cdot g(n) \]

To prove "set membership," we show that the conditions of membership are met.
Examples

\[ f(n) = \begin{cases} 
2n^2 + n \log n, & \text{if } n \geq 1000 \\
3n^2, & \text{o.w.}
\end{cases} \]

\textbf{Thm:} \quad f(n) = O(n^2)

\textbf{Proof:} \quad \exists c, n_0 > 0 \text{ s.t. } \forall n \geq n_0,
\[ f(n) \leq cn^2. \]

Let \( n_0 = 2^{1000} \).

Then \( \forall n \geq n_0, \quad f(n) = 2n^2 + n \log n \leq 3n^2 \).

It follows that \( f(n) = O(n^2) \).
Thm

\[ n^2 \log n \not\leq O(n \log^2 n) \]

Pf \quad \text{by contradiction}

Suppose \( n^2 \log n \leq O(n \log^2 n) \).
Then \( \exists c, n_0 > 0 \) s.t.
\[ \forall n > n_0, \quad n^2 \log n \leq c \cdot \log^2 n \]

Let \( c, n_0 \) be the particular constants for which this holds. Then

\[ \forall n > n_0, \quad n^2 \log n \leq c n \log^2 n. \]

Therefore
\[ \forall n > n_0 \quad n \leq c \cdot \log n \]

and
\[ \forall n > n_0 \quad \frac{n}{\log n} \leq c. \]

This is impossible.
Examples

\[ f(n) = n^2 - 1000n \]

**Thm**

\[ f(n) \in \Omega(n^2) \]

**Proof**

We need to show that \( \exists c, n_0 > 0 \) s.t. \( \forall n \geq n_0, f(n) \geq c \cdot n^2 \).

Let \( c = \frac{1}{2} \).

\[
\begin{align*}
\text{Is it true that } & n^2 - 1000n \geq \frac{n^2}{2} \\
\text{Yes, if} & \quad 2n^2 - 2000n \geq n^2 \\
\text{Yes, if} & \quad n^2 \geq 2000n \\
\text{Yes, if} & \quad n \geq 2000
\end{align*}
\]

Let \( n_0 = 2000 \).

For all \( n \geq n_0 \),

\[
\begin{align*}
n^2 - 1000n &= \frac{1}{2} (2n^2 - 2000n) \\
&\geq \frac{1}{2} (2n^2 - n^2) \\
&= \frac{1}{2} n^2
\end{align*}
\]
\[ f(n) = 3n^2 + 10n \log n \]

\[ \begin{align*}
  f(n) &\in \Theta(n^2) \quad \text{"proportional to"} \\
  &\in O(n^3) \quad \text{"at most proportional to"} \\
  &\in \Omega(n) \quad \text{"at least proportional to"}
\end{align*} \]

If
\[ \lim_{n \to \infty} \frac{f(n)}{g(n)} \]
exists and equals
\[ c > 0, \quad f(n) \in \Theta(g(n)) \quad \text{and x.v.} \]
\[ c < 0, \quad f(n) \in O(g(n)), \quad g(n) = \Omega(f(n)) \]

\[ f(n) \text{ grows more slowly than } g(n) \]
\[ f(n) \in o(g(n)) \]

\[ \infty, \quad f(n) \text{ grows more quickly than } g(n) \]
\[ f(n) \in \omega(g(n)) \]
\[ f(n) \in \Omega(g(n)) \]
Formally, \( f(x) \in \Theta(g(x)) \) if \( \exists \ c_1, c_2, n_0 > 0 \) s.t. \( \forall n \geq n_0, \ c_1 \cdot g(x) \leq f(x) \leq c_2 \cdot g(x) \)

To prove \( f(x) \in \Theta(g(x)) \) either

1) Show that \( \lim_{x \to \infty} \frac{f(x)}{g(x)} = c > 0 \), for some \( c \).

2) Show that you can pick \( c_1, c_2, n_0 \) s.t.

\[
\forall n \geq n_0, \ c_1 \cdot g(x) \leq f(x) \leq c_2 \cdot g(x)
\]

Note: to prove \( f(x) \in O(g(x)) \), leave out \( c_1 \) part.

- to prove \( f(x) \in \Omega(g(x)) \), leave out \( c_2 \) part.

Hint: often \( n_0 \) can be 1.
**Theorem**

If \( f(n) \) and \( g(n) \) are functions yielding positive values for all \( n \),

\[
\Theta \left( \max \{ f(n), g(n)^2 \} \right) = \\
\Theta \left( f(n) + g(n) \right)
\]

**Proof**

We show that membership in each set implies membership in the other.

Suppose \( h(n) \in \Theta \left( \max \{ f(n), g(n)^2 \} \right) \).

There exist \( n_0, c_1, c_2 > 0 \) s.t.

\[
\forall n \geq n_0, \quad c_1 \left( \max \{ f(n), g(n)^2 \} \right) \leq h(n) \leq c_2 \left( \max \{ f(n), g(n)^2 \} \right)
\]

It must be that

\[
\forall n \geq n_0, \quad c_1 \left( \frac{f(n) + g(n)}{2} \right) \leq h(n) \leq c_2 \left( f(n) + g(n) \right)
\]

It follows that \( h(n) \in \Theta (f(n) + g(n)) \).
Now suppose \( h(n) \in \Theta(f(n) + g(n)) \).

There exist \( n_0, c_1, c_2 \) such that
\[
\forall n \geq n_0, \quad c_1 (f(n) + g(n)) \leq h(n) \leq c_2 (f(n) + g(n))
\]

It follows that
\[
c_1 \left( \max \{f(n), g(n)\} \right) \leq h(n) \leq c_2 \cdot 2 \max \{f(n), g(n)\}
\]

and therefore
\[
h(n) \in \Theta\left( \max \{f(n), g(n)\} \right).
\]
Binary search (recursive)

```plaintext
int search (int[] a, int x) {  
    // Pre: a is sorted  
    // Post: returns position of x in a,  
    // or -1 if it isn't present  
    return search (a, 0, a.length - 1, x);  
}

int search (int[] a, int lo, int hi, int x) {  
    // Pre: a is sorted  
    // Post: returns position of x in a[lo..hi]  
    // or -1 if it isn't present  
    if (hi < lo)  
        return -1;  
    else  
        int mid = (lo + hi) / 2;  
        if (x == a[mid])  
            return mid;  
        else if (x < a[mid])  
            return search (a, lo, mid-1, x);  
        else  
            return search (a, mid+1, hi, x);  
}
```
Binary search (iterative)

```c
int search (int[] a, int x) {
    // Pre: a is sorted
    // Post: returns position of x in a, or -1 if it isn't present
    int lo = 0;
    int hi = a.length - 1;
    while (lo <= hi) {
        // if x is in a, it is in a[lo..hi]
        int mid = (lo + hi) / 2;
        if (x == a[mid])
            return mid;
        else if (x < a[mid])
            hi = mid - 1;
        else
            lo = mid + 1;
    }
    return -1;
}
```
Correctness?

Partial correctness:
are answers correct?

Termination?

Running time?

Key observation:
each call is half the size of previous one.

Alternatively, what range can be explored in k iterations:

\[
\begin{array}{cc}
\frac{k}{n} \\
1 & 1 \\
2 & 3 \\
3 & 7 \\
4 & 15 \\
\end{array}
\]

\[n = 2^k - 1,\]

\[k = \lceil \log (n+1) \rceil\]

\[\log n \leq \lceil \log (n+1) \rceil \leq (\log n) + 2\]

\[\ell \in \Theta (\log n)\]