Imperative programming - tutorial week 6

Arnold Meijster

Dept. computer science (university of Groningen)

October 8, 2017

7.4.1 Complexity

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $f(n) \in O(g(n))$

7.4.1 Complexity

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $f(n) \in O(g(n))$

$f(n) \in O(g(n))$ means: "the complexity of $f(n)$ is at most $g(n)$". The big-O notation yields therefore an upperbound of the 'real' complexity.

7.4.1 Complexity

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $f(n) \in O(g(n))$

$f(n) \in O(g(n))$ means: "the complexity of $f(n)$ is at most $g(n)$". The big-O notation yields therefore an upperbound of the 'real' complexity.

Formally: $f(n) \in O(g(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $f(n) \leq c \cdot g(n)$.

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $f(n) \in O(g(n))$

$f(n) \in O(g(n))$ means: "the complexity of $f(n)$ is at most $g(n)$". The big-O notation yields therefore an upperbound of the 'real' complexity.

Formally: $f(n) \in O(g(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $f(n) \leq c \cdot g(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$2 \cdot n^3 \quad \leq \quad c \cdot 19 \cdot n^2$$

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $f(n) \in O(g(n))$

$f(n) \in O(g(n))$ means: "the complexity of $f(n)$ is at most $g(n)$". The big-O notation yields therefore an upperbound of the 'real' complexity.

Formally: $f(n) \in O(g(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $f(n) \leq c \cdot g(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$2 \cdot n^3 \quad \leq \quad c \cdot 19 \cdot n^2$$

Clearly, this cannot be satisfied. Divide both sides by n^2 and we find:
$2n \leq 19c$. This cannot be true for a fixed c and all $n > N$.

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $f(n) \in O(h(n))$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $f(n) \in O(h(n))$

Formally: $f(n) \in O(h(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $f(n) \leq c \cdot h(n)$.

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $f(n) \in O(h(n))$

Formally: $f(n) \in O(h(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $f(n) \leq c \cdot h(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$2 \cdot n^3 \leq c \cdot 3 \cdot (n+3)^3$$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $f(n) \in O(h(n))$

Formally: $f(n) \in O(h(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $f(n) \leq c \cdot h(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$2 \cdot n^3 \leq c \cdot 3 \cdot (n+3)^3$$

Choose $c = \frac{2}{3}$

$$\begin{aligned} 2 \cdot n^3 &\leq \frac{2}{3} \cdot 3 \cdot (n+3)^3 \\ 2 \cdot n^3 &\leq 2 \cdot (n+3)^3 \\ n^3 &\leq (n+3)^3 \end{aligned}$$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $f(n) \in O(h(n))$

Formally: $f(n) \in O(h(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $f(n) \leq c \cdot h(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$2 \cdot n^3 \leq c \cdot 3 \cdot (n+3)^3$$

Choose $c = \frac{2}{3}$

$$\begin{aligned} 2 \cdot n^3 &\leq \frac{2}{3} \cdot 3 \cdot (n+3)^3 \\ 2 \cdot n^3 &\leq 2 \cdot (n+3)^3 \\ n^3 &\leq (n+3)^3 \end{aligned}$$

Choose $N = 0$, such that this is true for all $n > N$, so indeed $f(n)$ is $O(h(n))$.

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $g(n) \in O(f(n))$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $g(n) \in O(f(n))$

Formally: $g(n) \in O(f(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $g(n) \leq c \cdot f(n)$.

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $g(n) \in O(f(n))$

Formally: $g(n) \in O(f(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $g(n) \leq c \cdot f(n)$.

In this concrete example, this means that there $\exists c > 0$, N such that $\forall n > N$:

$$\begin{aligned} 19 \cdot n^2 &\leq c \cdot 2 \cdot n^3 \\ 19 &\leq c \cdot 2 \cdot n \end{aligned}$$

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $g(n) \in O(f(n))$

Formally: $g(n) \in O(f(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $g(n) \leq c \cdot f(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$\begin{aligned} 19 \cdot n^2 &\leq c \cdot 2 \cdot n^3 \\ 19 &\leq c \cdot 2 \cdot n \end{aligned}$$

Choose $c = 9\frac{1}{2}$

$$19 \leq 19 \cdot n$$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $g(n) \in O(f(n))$

Formally: $g(n) \in O(f(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $g(n) \leq c \cdot f(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$\begin{array}{rcl} 19 \cdot n^2 & \leq & c \cdot 2 \cdot n^3 \\ 19 & \leq & c \cdot 2 \cdot n \end{array}$$

Choose $c = 9\frac{1}{2}$

$$19 \leq 19 \cdot n$$

This is true for all $n > N = 1$, so indeed $g(n)$ is $O(f(n))$.

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $g(n) \in O(h(n))$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $g(n) \in O(h(n))$

Formally: $g(n) \in O(h(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $g(n) \leq c \cdot h(n)$.

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $g(n) \in O(h(n))$

Formally: $g(n) \in O(h(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $g(n) \leq c \cdot h(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$19 \cdot n^2 \quad \leq \quad c \cdot 3 \cdot (n+3)^3$$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $g(n) \in O(h(n))$

Formally: $g(n) \in O(h(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $g(n) \leq c \cdot h(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$19 \cdot n^2 \quad \leq \quad c \cdot 3 \cdot (n+3)^3$$

Choose $c = \frac{19}{3}$

$$\begin{aligned} 19 \cdot n^2 &\leq \frac{19}{3} \cdot 3 \cdot (n+3)^3 \\ 19 \cdot n^2 &\leq 19 \cdot (n+3)^3 \\ n^2 &\leq (n+3)^3 \end{aligned}$$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $g(n) \in O(h(n))$

Formally: $g(n) \in O(h(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $g(n) \leq c \cdot h(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$19 \cdot n^2 \quad \leq \quad c \cdot 3 \cdot (n+3)^3$$

Choose $c = \frac{19}{3}$

$$\begin{aligned} 19 \cdot n^2 \quad &\leq \quad \frac{19}{3} \cdot 3 \cdot (n+3)^3 \\ 19 \cdot n^2 \quad &\leq \quad 19 \cdot (n+3)^3 \\ n^2 \quad &\leq \quad (n+3)^3 \end{aligned}$$

This is true for all $n > N = 0$, so indeed $g(n)$ is $O(h(n))$.

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $h(n) \in O(f(n))$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $h(n) \in O(f(n))$

Formally: $h(n) \in O(f(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $h(n) \leq c \cdot f(n)$.

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $h(n) \in O(f(n))$

Formally: $h(n) \in O(f(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $h(n) \leq c \cdot f(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$\begin{aligned} 3 \cdot (n+3)^3 &\leq c \cdot 2 \cdot n^3 \\ 3n^3 + 27n^2 + 81n + 81 &\leq c \cdot 2n^3 \end{aligned}$$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $h(n) \in O(f(n))$

Formally: $h(n) \in O(f(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $h(n) \leq c \cdot f(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$\begin{array}{rcl} 3 \cdot (n+3)^3 & \leq & c \cdot 2 \cdot n^3 \\ 3n^3 + 27n^2 + 81n + 81 & \leq & c \cdot 2n^3 \end{array}$$

Choose $c = 96$

$$3n^3 + 27n^2 + 81n + 81 \quad \leq \quad 96 \cdot 2n^3 = 192 \cdot n^3 = 3n^3 + 27n^3 + 81n^3 + 81n^3$$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $h(n) \in O(f(n))$

Formally: $h(n) \in O(f(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $h(n) \leq c \cdot f(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$\begin{aligned} 3 \cdot (n+3)^3 &\leq c \cdot 2 \cdot n^3 \\ 3n^3 + 27n^2 + 81n + 81 &\leq c \cdot 2n^3 \end{aligned}$$

Choose $c = 96$

$$3n^3 + 27n^2 + 81n + 81 \quad \leq \quad 96 \cdot 2n^3 = 192 \cdot n^3 = 3n^3 + 27n^3 + 81n^3 + 81n^3$$

This is true for all $n > N = 1$, so indeed $h(n)$ is $O(f(n))$.

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $h(n) \in O(g(n))$

7.4.1 Complexity (continued)

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n + 3)^3$.
Prove or refute the following claim: $h(n) \in O(g(n))$

Formally: $h(n) \in O(g(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $h(n) \leq c \cdot g(n)$.

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $h(n) \in O(g(n))$

Formally: $h(n) \in O(g(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $h(n) \leq c \cdot g(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$
\begin{aligned}
3 \cdot (n+3)^3 = 3n^3 + 27n^2 + 81n + 81 \quad &\leq \quad c \cdot 19 \cdot n^2 \\
\frac{3}{19}n^3 + \frac{27}{19}n^2 + \frac{81}{19}n + \frac{81}{19} \quad &\leq \quad c \cdot n^2 \\
\frac{3}{19}n + \frac{27}{19} + \frac{81}{19n} + \frac{81}{19n^2} \quad &\leq \quad c
\end{aligned}
$$

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $h(n) \in O(g(n))$

Formally: $h(n) \in O(g(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $h(n) \leq c \cdot g(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$\begin{aligned}
3 \cdot (n+3)^3 = 3n^3 + 27n^2 + 81n + 81 &\leq c \cdot 19 \cdot n^2 \\
\frac{3}{19}n^3 + \frac{27}{19}n^2 + \frac{81}{19}n + \frac{81}{19} &\leq c \cdot n^2 \\
\frac{3}{19}n + \frac{27}{19} + \frac{81}{19n} + \frac{81}{19n^2} &\leq c
\end{aligned}$$

Clearly, this cannot be true. For example $n = \frac{19c}{3}$:

$$\begin{aligned}
\frac{3}{19} \cdot \frac{19c}{3} + \frac{27}{19} + \frac{81}{19\frac{19c}{3}} + \frac{81}{19(\frac{19c}{3})^2} &\leq c \\
c + \frac{27}{19} + \frac{81}{19\frac{19c}{3}} + \frac{81}{19(\frac{19c}{3})^2} &\leq c
\end{aligned}$$

Let $f(n) = 2 \cdot n^3$, $g(n) = 19 \cdot n^2$, and $h(n) = 3 \cdot (n+3)^3$.
Prove or refute the following claim: $h(n) \in O(g(n))$

Formally: $h(n) \in O(g(n))$ if there exist constants $c > 0$ and N, such that for each $n > N$ we have $h(n) \leq c \cdot g(n)$.

In this concrete example, this means that there $\exists c > 0,\ N$ such that $\forall n > N$:

$$\begin{aligned}
3 \cdot (n+3)^3 = 3n^3 + 27n^2 + 81n + 81 &\leq c \cdot 19 \cdot n^2 \\
\frac{3}{19}n^3 + \frac{27}{19}n^2 + \frac{81}{19}n + \frac{81}{19} &\leq c \cdot n^2 \\
\frac{3}{19}n + \frac{27}{19} + \frac{81}{19n} + \frac{81}{19n^2} &\leq c
\end{aligned}$$

Clearly, this cannot be true. For example $n = \frac{19c}{3}$:

$$\begin{aligned}
\frac{3}{19} \cdot \frac{19c}{3} + \frac{27}{19} + \frac{81}{19\frac{19c}{3}} + \frac{81}{19(\frac{19c}{3})^2} &\leq c \\
c + \frac{27}{19} + \frac{81}{19\frac{19c}{3}} + \frac{81}{19(\frac{19c}{3})^2} &\leq c
\end{aligned}$$

We now have a contradiction, because $\frac{27}{19} + \frac{81}{19\frac{19c}{3}} + \frac{81}{19(\frac{19c}{3})^2} \geq 0$.

We are located at the origin of a grid with integer coordinates and want to walk to (i, j) (where $i \geq 0$ and $j \geq 0$). At each grid point, we are allowed to move one step to the north or to the east.

1. Design a recursive function $F(i, j)$ that computes the number of possible walks from $(0, 0)$ to (i, j).
2. Find a closed expression (i.e. a formula without recursion) for $F(i, j)$ and prove its correctness by means of an inductive proof.

We find the following recurrence $F(i, j)$:

- $F(i, 0) =$

7.4.2 Manhattan walks (continued)

We find the following recurrence $F(i,j)$:

- $F(i,0) = 1$
- $F(i,0) =$

7.4.2 Manhattan walks (continued)

We find the following recurrence $F(i,j)$:

- $F(i,0) = 1$
- $F(i,0) = 1$
- $F(i,j) =$

7.4.2 Manhattan walks (continued)

We find the following recurrence $F(i, j)$:

- $F(i, 0) = 1$
- $F(i, 0) = 1$
- $F(i, j) = F(i - 1, j) + F(i, j - 1)$

We find the following recurrence $F(i,j)$:

- $F(i,0) = 1$
- $F(i,0) = 1$
- $F(i,j) = F(i-1,j) + F(i,j-1)$

A recursive implementation would be:

```
int numPaths(int i, int j) {
  if ((i==0) || (j==0)) {
    return 1;
  }
  return numPaths(i-1, j) + numPaths(i, j-1);
}
```

7.4.2 Manhattan walks (continued)

Every path consists of i steps to the right and j steps upwards.

7.4.2 Manhattan walks (continued)

Every path consists of i steps to the right and j steps upwards.

Define $R =$ "step right" and $U =$ "step upwards". Every sequence in which i times an R and j times a U occurs is therefore a valid path from $(0,0)$ to (i,j).

Every path consists of i steps to the right and j steps upwards.

Define $R =$ "step right" and $U =$ "step upwards". Every sequence in which i times an R and j times a U occurs is therefore a valid path from $(0,0)$ to (i,j).

So, you might me tempted to say that the total number of paths is $(i+j)!$.

7.4.2 Manhattan walks (continued)

Every path consists of i steps to the right and j steps upwards.

Define $R =$ "step right" and $U =$ "step upwards". Every sequence in which i times an R and j times a U occurs is therefore a valid path from $(0, 0)$ to (i, j).

So, you might me tempted to say that the total number of paths is $(i + j)!$.

However, we have to correct for counting paths multiple times (you cannot distinguish R_0R_1 from R_1R_0).

Every path consists of i steps to the right and j steps upwards.

Define $R =$ "step right" and $U =$ "step upwards". Every sequence in which i times an R and j times a U occurs is therefore a valid path from $(0,0)$ to (i,j).

So, you might me tempted to say that the total number of paths is $(i+j)!$.

However, we have to correct for counting paths multiple times (you cannot distinguish R_0R_1 from R_1R_0).

All occurrences of R can be permuted in $i!$ ways:

$$\text{for example } R_1R_2R_3,\ R_1R_3R_2,\ R_2R_1R_3,\ R_2R_3R_1,\ R_3R_1R_2,\ R_3R_2R_1$$

7.4.2 Manhattan walks (continued)

Every path consists of i steps to the right and j steps upwards.

Define $R =$ "step right" and $U =$ "step upwards". Every sequence in which i times an R and j times a U occurs is therefore a valid path from $(0,0)$ to (i,j).

So, you might me tempted to say that the total number of paths is $(i+j)!$.

However, we have to correct for counting paths multiple times (you cannot distinguish R_0R_1 from R_1R_0).

All occurrences of R can be permuted in $i!$ ways:

for example $R_1R_2R_3$, $R_1R_3R_2$, $R_2R_1R_3$, $R_2R_3R_1$, $R_3R_1R_2$, $R_3R_2R_1$

The same holds for U which can be permuted in $j!$ ways. So, we find:

$$\texttt{F(i,j)} = \frac{\texttt{(i+j)!}}{\texttt{i!} \cdot \texttt{j!}}.$$

7.4.2 Manhattan walks (continued)

Every path consists of i steps to the right and j steps upwards.

Define $R =$ "step right" and $U =$ "step upwards". Every sequence in which i times an R and j times a U occurs is therefore a valid path from $(0, 0)$ to (i, j).

So, you might me tempted to say that the total number of paths is $(i + j)!$.

However, we have to correct for counting paths multiple times (you cannot distinguish R_0R_1 from R_1R_0).

All occurrences of R can be permuted in $i!$ ways:

for example $R_1R_2R_3$, $R_1R_3R_2$, $R_2R_1R_3$, $R_2R_3R_1$, $R_3R_1R_2$, $R_3R_2R_1$

The same holds for U which can be permuted in $j!$ ways. So, we find:

$$\texttt{F(i,j)} = \frac{\texttt{(i+j)!}}{\texttt{i!} \cdot \texttt{j!}}.$$

We prove it using mathematical induction:

7.4.2 Manhattan walks (continued)

Every path consists of i steps to the right and j steps upwards.

Define $R =$ "step right" and $U =$ "step upwards". Every sequence in which i times an R and j times a U occurs is therefore a valid path from $(0,0)$ to (i,j).

So, you might me tempted to say that the total number of paths is $(i+j)!$.

However, we have to correct for counting paths multiple times (you cannot distinguish R_0R_1 from R_1R_0).

All occurrences of R can be permuted in $i!$ ways:

$$\text{for example } R_1R_2R_3, R_1R_3R_2, R_2R_1R_3, R_2R_3R_1, R_3R_1R_2, R_3R_2R_1$$

The same holds for U which can be permuted in $j!$ ways. So, we find:

$$\texttt{F(i,j)} = \frac{\texttt{(i+j)!}}{\texttt{i!}\cdot\texttt{j!}}.$$

We prove it using mathematical induction: base cases are $i = 0$ and $j = 0$:

$$\texttt{F(0,j)} = 1 = \frac{\texttt{j!}}{\texttt{j!}} = \frac{\texttt{(0+j)!}}{\texttt{0!}\cdot\texttt{j!}}.$$

$$\texttt{F(i,0)} = 1 = \frac{\texttt{i!}}{\texttt{i!}} = \frac{\texttt{(i+0)!}}{\texttt{i!}\cdot\texttt{i!}}.$$

The induction step goes as follows:

$$\texttt{F(i,j)} = \texttt{F(i-1, j) + F(i, j-1)}$$

The induction step goes as follows:

$$\begin{aligned}\texttt{F(i,j)} &= \texttt{F(i-1, j) + F(i, j-1)} \\ &= \frac{\texttt{(i-1+j)!}}{\texttt{(i-1)!}\cdot\texttt{j!}} + \frac{\texttt{(i+j-1)!}}{\texttt{i!}\cdot\texttt{(j-1)!}}\end{aligned}$$

The induction step goes as follows:

$$
\begin{aligned}
\texttt{F(i,j)} &= \texttt{F(i-1, j) + F(i, j-1)} \\
&= \frac{\texttt{(i-1+j)!}}{\texttt{(i-1)!}\cdot\texttt{j!}} + \frac{\texttt{(i+j-1)!}}{\texttt{i!}\cdot\texttt{(j-1)!}} \\
&= \frac{\texttt{i(i-1+j)!}}{\texttt{i(i-1)!}\cdot\texttt{j!}} + \frac{\texttt{j(i+j-1)!}}{\texttt{i!}\cdot\texttt{j(j-1)!}}
\end{aligned}
$$

The induction step goes as follows:

$$
\begin{aligned}
\texttt{F(i,j)} &= \texttt{F(i-1, j) + F(i, j-1)} \\
&= \frac{\texttt{(i-1+j)!}}{\texttt{(i-1)!}\cdot\texttt{j!}} + \frac{\texttt{(i+j-1)!}}{\texttt{i!}\cdot\texttt{(j-1)!}} \\
&= \frac{\texttt{i(i-1+j)!}}{\texttt{i(i-1)!}\cdot\texttt{j!}} + \frac{\texttt{j(i+j-1)!}}{\texttt{i!}\cdot\texttt{j(j-1)!}} \\
&= \frac{\texttt{i(i-1+j)! + j(i+j-1)!}}{\texttt{i!}\cdot\texttt{j!}}
\end{aligned}
$$

The induction step goes as follows:

$$
\begin{aligned}
\texttt{F(i,j)} &= \texttt{F(i-1, j) + F(i, j-1)} \\
&= \frac{\texttt{(i-1+j)!}}{\texttt{(i-1)!}\cdot\texttt{j!}} + \frac{\texttt{(i+j-1)!}}{\texttt{i!}\cdot\texttt{(j-1)!}} \\
&= \frac{\texttt{i(i-1+j)!}}{\texttt{i(i-1)!}\cdot\texttt{j!}} + \frac{\texttt{j(i+j-1)!}}{\texttt{i!}\cdot\texttt{j(j-1)!}} \\
&= \frac{\texttt{i(i-1+j)! + j(i+j-1)!}}{\texttt{i!}\cdot\texttt{j!}} \\
&= \frac{\texttt{(i+j)(i+j-1)!}}{\texttt{i!}\cdot\texttt{j!}}
\end{aligned}
$$

The induction step goes as follows:

$$
\begin{aligned}
\mathtt{F(i,j)} &= \mathtt{F(i-1,\ j)\ +\ F(i,\ j-1)} \\
&= \frac{\mathtt{(i-1+j)!}}{\mathtt{(i-1)!}\cdot\mathtt{j!}} + \frac{\mathtt{(i+j-1)!}}{\mathtt{i!}\cdot\mathtt{(j-1)!}} \\
&= \frac{\mathtt{i(i-1+j)!}}{\mathtt{i(i-1)!}\cdot\mathtt{j!}} + \frac{\mathtt{j(i+j-1)!}}{\mathtt{i!}\cdot\mathtt{j(j-1)!}} \\
&= \frac{\mathtt{i(i-1+j)!\ \ +\ j(i+j-1)!}}{\mathtt{i!}\cdot\mathtt{j!}} \\
&= \frac{\mathtt{(i+j)(i+j-1)!}}{\mathtt{i!}\cdot\mathtt{j!}} = \frac{\mathtt{(i+j)!}}{\mathtt{i!}\cdot\mathtt{j!}} \quad \text{QED.}
\end{aligned}
$$

7.4.3 Square roots

The *integer root* of a natural number `x` is the greatest integer `r` such that `r*r <= x`. In this exercise we will implement two different versions of the function `isqrt`:

```
int isqrt(int x);    // greatest r such that r*r<=x
```

(a) Linear search: use a linear search algorithm to implement the body of the function `isqrt`.

(b) Binary search: use a binary search algorithm to implement the body of the function `isqrt`. Introduce two variables `p` and `q` and maintain the invariant `p*p <= x < q*q`.

(c) Determine the time complexity of both algorithms. Which version is more efficient?

Linear search:

```
int isqrt(int x) {
  int w = 0 ;
  while ((w+1)*(w+1) <= x) {
    w++;
  }
  return w;
}
```

7.4.3 Square roots

Binary search: invariant `p*p <= x < q*q`

```
int isqrt(int x) {
  int p = 0;
  int q = x+1;
  while (p != q - 1) {
    int m = (p + q)/2;
    if (m*m <= x) {
      p = m;
    } else {
      q = m;
    }
  }
  return p;
}
```

Binary search: invariant `p*p <= x < q*q`

```
int isqrt(int x) {
  int p = 0;
  int q = x+1;
  while (p != q - 1) {
    int m = (p + q)/2;
    if (m*m <= x) {
      p = m;
    } else {
      q = m;
    }
  }
  return p;
}
```

Clearly, binary search is faster than linear search: $O(\log n)$ vs. $O(n)$.

7.4.4. Bisection method for finding roots

In this exercise, we consider the function $f(x) = a \cdot x^3 + b \cdot x^2 + c \cdot x + d$.

The values a, b, c and d are natural numbers.

Write a function that, using binary search, returns a `double` value `x` such that the absolute value of $f(x)$ is less than 0.0001.

```
double f(int a, int b, int c, int d, double x) {
  return a*x*x*x + b*x*x + c*x + d;
}

double absval(double n) {
  return (n < 0 ? -n : n);
}
```

```
double bisection(int a, int b, int c, int d) {
  double left=-1, right=1, mid;
  while (f(a, b, c, d, left) > 0) {
    left *= 2;
  }
  while (f(a, b, c, d, right) <= 0) {
    right *= 2;
  }
  // invariant: f(a,b,c,d,left)<=0<f(a,b,c,d,right)
  while (absval(f(a,b,c,d,left)) >= 0.0001) {
    mid = (left + right)/2;
    if (f(a,b,c,d,mid) <= 0) {
      left = mid;
    } else {
      right = mid;
    }
  }
  return left;
}
```

7.4.5. A two-dimensional landscape

Given is the declaration:

```
int alt[N][N];
```

Think of `alt` as a landscape, where `alt[x][y]` denotes the altitude at `(x,y)`.

(a) [Easy] Write a code fragment that counts the number of points that have an altitude below a given altitude `w`. The time complexity of your algorithm should be quadratic in `N`. For example, if `w==20`, we count the number of **bold face** figures in the following matrix:

1	**16**	25	22	**0**	**1**	**17**	20	**19**	29
9	22	**7**	**1**	**5**	**16**	**13**	**3**	**14**	24
12	**6**	**13**	**16**	**14**	20	**9**	**14**	**11**	**6**
16	**0**	**2**	**13**	**8**	**2**	**16**	**14**	**3**	**16**
25	**16**	20	27	**7**	**3**	**5**	27	24	22
23	23	**2**	29	**14**	26	26	**14**	**8**	**19**
25	**19**	**9**	**18**	29	20	27	**15**	**8**	**18**
27	20	27	**12**	21	**1**	**14**	**12**	**6**	26
16	**7**	**8**	**12**	**3**	**16**	**15**	**15**	**18**	**0**
13	**2**	**11**	29	**9**	23	**15**	24	**7**	**12**

7.4.5. A two-dimensional landscape

```c
int alt[N][N] = {
  {7, 13, 14, 25, 25, 27, 29, 29, 32, 33},
  {6, 11, 12, 23, 24, 25, 27, 29, 32, 32},
  {6,  9, 12, 22, 22, 23, 27, 29, 30, 30},
  {6,  9, 10, 20, 20, 23, 25, 25, 27, 28},
  {6,  9, 10, 18, 20, 21, 21, 23, 25, 25},
  {6,  7, 10, 16, 16, 19, 21, 22, 23, 23},
  {5,  5,  8, 14, 15, 17, 19, 21, 21, 23},
  {5,  5,  6, 12, 12, 15, 16, 17, 18, 19},
  {5,  5,  6, 10, 12, 14, 15, 16, 17, 19},
  {3,  5,  6,  8,  9,  9,  9, 10, 11, 13}
};

int easy(int w) {
  int i, j, cnt = 0;
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      cnt += (alt[i][j] < w);
    }
  }
  return cnt;
}
```

7.4.5. A two-dimensional landscape

(b) [Intermediate] Again, consider the array `arr`, with some extra restrictions:

- If `x0 <= x1` then `alt[x0][y] <= alt[x1][y]` for all `y`.
- If `y0 <= y1` then `alt[x][y0] <= alt[x][y0]` for all `x`.

Think of `alt` as a slope of which the altitude does not decrease if one moves to the east or north (or north-east).

7	**13**	**14**	25	25	27	29	29	32	33
6	**11**	**12**	23	24	25	27	29	32	32
6	**9**	**12**	22	22	23	27	29	30	30
6	**9**	**10**	20	20	23	25	25	27	28
6	**9**	**10**	**18**	20	21	21	23	25	25
6	**7**	**10**	**16**	**16**	**19**	21	22	23	23
5	**5**	**8**	**14**	**15**	**17**	**19**	21	21	23
5	**5**	**6**	**12**	**12**	**15**	**16**	**17**	**18**	**19**
5	**5**	**6**	**10**	**12**	**14**	**15**	**16**	**17**	**19**
3	**5**	**6**	**8**	**9**	**9**	**9**	**10**	**11**	**13**

Write a code fragment that counts the number of points that have an altitude below a given altitude `w`. The time complexity of your algorithm should be of the order `N*log N`. [*Hint: Use a binary search per row.*]

7.4.5. A two-dimensional landscape

```
  int binarySearch(int length, int arr[], int value) {
  int left = 0, right = length;
  while (left + 1 < right) {
    int mid = (left + right)/2;
    if (value < arr[mid]) {
      right = mid; /* right is lowered */
    } else {
      left = mid; /* left is raised */
    }
  }
  return left;
}
int intermediate(int w) {
  int cnt = 0;
  for (int i = 0; i < N; i++) {
    int j = binarySearch(N, alt[i], w-1);
    if (alt[i][j] < w) {
      cnt += j + 1;
    }
  }
  return cnt;
}
```

(c) [Hard] The same exercise as in (b), but now with a linear time complexity (i.e. of the order `N`).

(c) [Hard] The same exercise as in (b), but now with a linear time complexity (i.e. of the order N).

```
int hard(int w) {
  int i=0, j=0, cnt=0;
  while ((i < N) && (j < N)) {
    if (alt[i][j] >= w) {
      i++;
    } else {
      cnt += N-i;
      j++;
    }
  }
  return cnt;
}
```