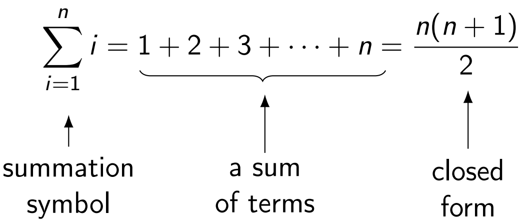
# Summations

Class 36

# Introduction

- Section 4.4 was about using proof by induction to prove that a closed form is correct
- this section is about techniques for finding closed forms

# Closed Forms



$$\sum_{i=1}^{n} i = \underbrace{1 + 2 + 3 + \cdots + n} = \frac{n(n+1)}{2}$$

- a closed form is an expression that is not a symbolic recipe
- that does not have ellipses
- that can be computed directly

(LaTeX: `\sum_{i=1}^n i`)

# Working With Summations

- box 5.2.1 on page 298 has some “facts”
- fact a: sum of a constant

$$\sum_{i=m}^{n} c = c(n - m + 1)$$

- $c$ is a constant
- $m \leq n,\ m, n \in \mathrm{N}$
- example: $\sum_{i=5}^{10} 2$ (notice how inline summation is formatted)

$$\sum_{i=5}^{10} 2 = 2(10 - 5 + 1)$$

$$= 12$$

# Constant Coefficient

- fact b: each term has an constant coefficient that can be factored out

$$\sum_{i=m}^{n} ca_i = c \sum_{i=m}^{n} a_i$$

- example

$$\begin{aligned} \sum_{i=1}^{10} 4i &= 4 \sum_{i=1}^{10} i \\ &= 4 \frac{10(10+1)}{2} \\ &= 220 \end{aligned}$$

# Collapsing Sums

- fact c: each element of the series is composed of two terms, adding a new term but subtracting the term from last time
- this happens more often than you might think

$$\sum_{i=1}^{n}(a_i - a_{i-1}) = a_n - a_0$$

$$\begin{aligned}
\sum_{i=1}^{8}(i - (i - 1)) &= (1 - 0) + (2 - 1) + \cdots + (8 - 7) \\
&= 8 - 7 + 7 - 6 + \cdots + 2 - 1 + 1 - 0 \\
&= 8 - 0 \\
&= 8
\end{aligned}$$

# Summation of Added Terms

- fact d: each element of the summation is a pair of terms, added together

$$\sum_{i=m}^{n}(a_i + b_i) = \sum_{i=m}^{n} a_i + \sum_{i=m}^{n} b_i$$

$$\begin{aligned}\sum_{i=1}^{10}(2i + 3i) &= \sum_{i=1}^{10} 2i + \sum_{i=1}^{10} 3i \\ &= 110 + 165 \\ &= 275\end{aligned}$$

- facts e – g are used less often

# Geometric Progression

- a common summation is

$$\sum_{i=0}^{n} a^i = a^0 + a^1 + a^2 + \cdots + a^n$$

- what is a closed form?

start with the following expression, and then factor

$$a^{i+1} - a^i = a^i(a-1)$$

# Geometric Progression

- a common summation is

$$\sum_{i=0}^{n} a^i = a^0 + a^1 + a^2 + \cdots + a^n$$

- what is a closed form?

start with the following expression, and then factor

$$a^{i+1} - a^i = a^i(a-1)$$

apply summation to both sides to get

$$\sum_{i=0}^{n}(a^{i+1} - a^i) = \sum_{i=0}^{n}(a^i(a-1))$$

# Derivation

$$\underbrace{\sum_{i=0}^{n}(a^{i+1}-a^{i})}_{\displaystyle\big\downarrow\\ a^{n+1}-1} = \sum_{i=0}^{n}(a^{i}(a-1))$$

by a direct application of fact c, the left side collapses

# Derivation

$$\sum_{i=0}^{n}(a^{i+1}-a^{i}) = \underbrace{\sum_{i=0}^{n}(a^{i}(a-1))}$$

$$\swarrow$$

$$(a-1)\sum_{i=0}^{n} a^{i}$$

by a direct application of fact b, the right side can have the common coefficient factored out

# Derivation

from the previous two slides we have

$$\sum_{i=0}^{n}(a^{i+1} - a^{i}) = \sum_{i=0}^{n}(a^{i}(a-1))$$

which gives

$$a^{n+1} - 1 = (a-1)\sum_{i=0}^{n} a^{i}$$

and if we ensure $a > 1$, we can rearrange to get

$$\sum_{i=0}^{n} a^{i} = \frac{a^{n+1}-1}{a-1}$$

and thus we have a closed form for the geometric summation

# Loops

- suppose we have a list of numbers and want to print them

$$[6, 16, 18, 16, 6, 9, 19, 0, 6, 5, 8, 0, 12, 19, 12, 3, 1, 9, 2, 16]$$

```
void print(const list<unsigned>& values)
{
  for (size_t index = 0; index < list.size(); index++)
  {
    cout << list.at(index) << endl;
  }
}
```

- how many times does the cout statement execute?

- there are two flavors of for loop headers:
  1: `for (index = lower; index < upper; index++)`
  2: `for (index = lower; index <= upper; index++)`
- the number of body iterations:
  - type 1: upper − lower
  - type 2: upper − lower + 1

# Nested Loops

- suppose we have a 2-dimensional matrix of values and need to print them

```
void print(const Matrix<unsigned>& values)
{
  for (size_t row = 0; row < values.numrows(); row++)
  {
    for (size_t col = 0; col < values.numcols(); col++)
    {
      cout << values.at(row, col) << ' ';
    }
    cout << endl;
  }
}
```

- if the number of rows is $n$ and the number of columns is $m$, so that the matrix is an $n \times m$ grid, and cout executes exactly once for each value in the grid, then cout must execute $n \times m$ times

# Nested Loops

- suppose we have a list of numbers and need to count how many times a value earlier in the list is larger than a value later in the list
- for example, looking at the first 6, it is larger than 0, 5, 0, 3, 1, and 2

$$[6, 16, 18, 16, 6, 9, 19, 0, 6, 5, 8, 0, 12, 19, 12, 3, 1, 9, 2, 16]$$

# Nested Loops

[6, 16, 18, 16, 6, 9, 19, 0, 6, 5, 8, 0, 12, 19, 12, 3, 1, 9, 2, 16]

```
 1  unsigned count_larger(const list<unsigned>& values)
 2  {
 3    unsigned count = 0;
 4    for (size_t outer = 0; outer < values.size() - 1; outer++)
 5    {
 6      for (size_t inner = outer + 1; inner < values.size(); inner++)
 7      {
 8        if (values.at(outer) > values.at(inner))
 9        {
10          count++;
11        }
12      }
13    }
14    return count;
15  }
```

- how many times does the comparison on line 8 execute?

# Nested Loops

```
unsigned count_larger(const list<unsigned>& values) // values.size() = 20
{
  unsigned count = 0;
  for (size_t outer = 0; outer < values.size() - 1; outer++)
  {
    for (size_t inner = outer + 1; inner < values.size(); inner++)
    {
      if (values.at(outer) > values.at(inner))
      {
        count++;
      }
```

- how many times does outer loop execute?

# Nested Loops

```
unsigned count_larger(const list<unsigned>& values) // values.size() = 20
{
  unsigned count = 0;
  for (size_t outer = 0; outer < values.size() - 1; outer++)
  {
    for (size_t inner = outer + 1; inner < values.size(); inner++)
    {
      if (values.at(outer) > values.at(inner))
      {
        count++;
      }
```

- how many times does outer loop execute? 19 times
- when outer loop is 0, inner loop is
  `for (i = 1; i < 20; i++)`
- how many times does the inner loop execute?

# Nested Loops

```
unsigned count_larger(const list<unsigned>& values) // values.size() = 20
{
  unsigned count = 0;
  for (size_t outer = 0; outer < values.size() - 1; outer++)
  {
    for (size_t inner = outer + 1; inner < values.size(); inner++)
    {
      if (values.at(outer) > values.at(inner))
      {
        count++;
      }
```

- how many times does outer loop execute? 19 times
- when outer loop is 0, inner loop is
  `for (i = 1; i < 20; i++)`
- how many times does the inner loop execute? $20 - 1 = 19$

# Nested Loops

```
unsigned count_larger(const list<unsigned>& values)
{
  unsigned count = 0;
  for (size_t outer = 0; outer < values.size() - 1; outer++)
  {
    for (size_t inner = outer + 1; inner < values.size(); inner++)
    {
      if (values.at(outer) > values.at(inner))
      {
        count++;
      }
```

- when outer is 1, what are the inner values and how many iterations?

# Nested Loops

```
unsigned count_larger(const list<unsigned>& values)
{
  unsigned count = 0;
  for (size_t outer = 0; outer < values.size() - 1; outer++)
  {
    for (size_t inner = outer + 1; inner < values.size(); inner++)
    {
      if (values.at(outer) > values.at(inner))
      {
        count++;
      }
```

- when outer is 1, what are the inner values and how many iterations?

  `for (i = 2; i < 20; i++)` runs 18 times

# Nested Loops

```
unsigned count_larger(const list<unsigned>& values)
{
  unsigned count = 0;
  for (size_t outer = 0; outer < values.size() - 1; outer++)
  {
    for (size_t inner = outer + 1; inner < values.size(); inner++)
    {
      if (values.at(outer) > values.at(inner))
      {
        count++;
      }
```

we can make a table

| outer | inner start | inner< | #iterations |
|---|---|---|---|
| 0 | 1 | 20 | 19 |
| 1 | 2 | 20 | 18 |
| 2 | 3 | 20 | 17 |
| 3 | 4 | 20 | 16 |
|  | ⋮ |  |  |
| 18 | 19 | 20 | 1 |

# Nested Loops

| **outer** | **inner start** | **inner<** | **#iterations** |
|---|---|---|---|
| 0 | 1 | 20 | 19 |
| 1 | 2 | 20 | 18 |
| 2 | 3 | 20 | 17 |
| 3 | 4 | 20 | 16 |
| | ⋮ | | |
| 18 | 19 | 20 | 1 |

- what is the total number of inner loop body executions?
- remember, $n = 20$

# Nested Loops

| outer | inner start | inner< | #iterations |
|---|---|---|---|
| 0 | 1 | 20 | 19 |
| 1 | 2 | 20 | 18 |
| 2 | 3 | 20 | 17 |
| 3 | 4 | 20 | 16 |
|  | $\vdots$ |  |  |
| 18 | 19 | 20 | 1 |

- what is the total number of inner loop body executions?
- remember, $n = 20$

$$\begin{aligned}\sum_{k=1}^{n-1} k &= \frac{(n-1)n}{2} \\ &= 190\end{aligned}$$

# Counting Operations

- the total number of inner loop body executions

$$\sum_{k=1}^{n-1} k = \frac{(n-1)n}{2}$$

- as we discussed in section 5.1, we use the number of operations as a measure of the time consumed when an algorithm executes
- we use the expression $T(n)$ to indicate the number of operations performed by an algorithm (the T stands for time)
- thus for the count_larger algorithm, we have

$$T(n) = \frac{(n-1)n}{2}$$

# Generalize

```
void foo(unsigned n)
{
  for (unsigned outer = 0; outer < n; outer++)
  {
    for (unsigned inner = outer + 1; inner < n; inner++)
    {
      bar();
    }
  }
}
```

- suppose every time `bar()` executes, three operations of a specific type are performed
- find a closed form for the total number of these operations performed for a given $n$
- note the bounds are slightly different from the example above

# Counting bar()

```
void foo(unsigned n)
{
  for (unsigned outer = 0; outer < n; outer++)
  {
    for (unsigned inner = outer + 1; inner < n; inner++)
    {
      bar();
    }
  }
}
```

| outer | inner start | inner< | #operations |
|---|---|---|---|
| 0 | 1 | $n$ | $3(n-1)$ |
| 1 | 2 | $n$ | $3(n-2)$ |
| 2 | 3 | $n$ | $3(n-3)$ |
| 3 | 4 | $n$ | $3(n-4)$ |
|  | ⋮ |  |  |
| $n-2$ | $n-1$ | $n$ | $3(n-(n-1))$ |
| $n-1$ | $n$ | $n$ | $3(n-n)$ |

# Counting bar()

| outer | inner start | inner< | #operations |
|---|---|---|---|
| 0 | 1 | $n$ | $3(n-1)$ |
| 1 | 2 | $n$ | $3(n-2)$ |
|  | $\vdots$ |  |  |
| $n-1$ | $n$ | $n$ | $3(n-n)$ |

$$\begin{aligned} T(n) &= \sum_{i=0}^{n-1} 3i \\ &= 3\frac{(n-1)n}{2} \end{aligned}$$

for $n = 20$, for example we have

$$\begin{aligned} T(20) &= 3\frac{(20-1)20}{2} \\ &= 570 \end{aligned}$$

# Count bar() version 2

```
void foo(unsigned n)
{
  for (unsigned outer = 0; outer < n; outer++)
  {
    for (unsigned inner = 0; inner < 2 * outer; inner++)
    {
      bar();
    }
  }
}
```

- again bar() executes three specific operations
- find a closed form for the number of these operations performed for a given *n*
- make a table
- if it's confusing for *n*, make it for an example specific example value, e.g., 10

# Count bar() version 2

```
void foo(unsigned n)
{
  for (unsigned outer = 0; outer < n; outer++)
  {
    for (unsigned inner = 0; inner < 2 * outer; inner++)
    {
      bar();
    }
  }
}
```

| outer | inner start | inner$<$ | #operations |
|---|---|---|---|

# Count bar() version 2

```
void foo(unsigned n)
{
  for (unsigned outer = 0; outer < n; outer++)
  {
    for (unsigned inner = 0; inner < 2 * outer; inner++)
    {
      bar();
    }
  }
}
```

| outer | inner start | inner< | #operations |
|---|---|---|---|
| 0 | 0 | 0 | $3(0) = 6(\text{outer})$ |
| 1 | 0 | 2 | $3(2) = 6(\text{outer})$ |
| 2 | 0 | 4 | $3(4) = 6(\text{outer})$ |
| 3 | 0 | 6 | $3(6) = 6(\text{outer})$ |
|  | $\vdots$ |  |  |
| $n-2$ | 0 | $2(n-2)$ | $6(n-2)$ |
| $n-1$ | 0 | $2(n-1)$ | $6(n-1)$ |

# Count `bar()` version 2

| outer | inner start | inner< | #operations |
|---|---|---|---|
| 0 | 0 | 0 | $3(0) = 6(\text{outer})$ |
| 1 | 0 | 2 | $3(2) = 6(\text{outer})$ |
| 2 | 0 | 4 | $3(4) = 6(\text{outer})$ |
| 3 | 0 | 6 | $3(6) = 6(\text{outer})$ |
|  | $\vdots$ |  |  |
| $n-2$ | 0 | $2(n-2)$ | $6(n-2)$ |
| $n-1$ | 0 | $2(n-1)$ | $6(n-1)$ |

# Count `bar()` version 2

| outer | inner start | inner< | #operations |
|---|---|---|---|
| 0 | 0 | 0 | $3(0) = 6(\text{outer})$ |
| 1 | 0 | 2 | $3(2) = 6(\text{outer})$ |
| 2 | 0 | 4 | $3(4) = 6(\text{outer})$ |
| 3 | 0 | 6 | $3(6) = 6(\text{outer})$ |
|  | ⋮ |  |  |
| $n-2$ | 0 | $2(n-2)$ | $6(n-2)$ |
| $n-1$ | 0 | $2(n-1)$ | $6(n-1)$ |

$$T(n) = \sum_{i=0}^{n-1} 6i$$

$$\begin{aligned} T(n) &= 6\frac{(n-1)n}{2} \\ &= 3(n^2 - n) \end{aligned}$$