Algorithm Analysis

Class 6

Two Nested Loops

```
uint64_t sum = 0;
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    sum++;
  }
}
```

n	sum
1	

Two Nested Loops

```
uint64_t sum = 0;
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    sum++;
  }
}
```

n	sum
1	1
2	

Two Nested Loops

```
uint64_t sum = 0;
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    sum++;
  }
}
```

n	sum
1	1
2	3
3	

Two Nested Loops

```
uint64_t sum = 0;
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    sum++;
  }
}
```

n	sum
1	1
2	3
3	6
4	

Two Nested Loops

```
uint64_t sum = 0;
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    sum++;
  }
}
```

n	sum
1	1
2	3
3	6
4	10
5	

Two Nested Loops

```
uint64_t sum = 0;
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    sum++;
  }
}
```

n	**sum**
1	1
2	3
3	6
4	10
5	15
n	

Two Nested Loops

```
uint64_t sum = 0;
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    sum++;
  }
}
```

n	sum
1	1
2	3
3	6
4	10
5	15
n	$\frac{n(n+1)}{2}$

Two Nested Loops

```
uint64_t sum = 0;
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    sum++;
  }
}
```

n	sum
1	1
2	3
3	6
4	10
5	15
n	$\frac{n(n+1)}{2}$

- the number of times the inner loop runs is

$$\frac{n(n+1)}{2}$$

Loop Operations

```
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    bar(); // assume this takes exactly 2 basic operations
  }
}
```

n	basic operations
1	

Loop Operations

```
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    bar(); // assume this takes exactly 2 basic operations
  }
}
```

n	basic operations
1	10
2	

Loop Operations

```
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    bar(); // assume this takes exactly 2 basic operations
  }
}
```

n	basic operations
1	10
2	22
3	

Loop Operations

```
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    bar(); // assume this takes exactly 2 basic operations
  }
}
```

n	basic operations
1	10
2	22
3	38
4	

Loop Operations

```
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    bar(); // assume this takes exactly 2 basic operations
  }
}
```

n	basic operations
1	10
2	22
3	38
4	58
5	

Loop Operations

```
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    bar(); // assume this takes exactly 2 basic operations
  }
}
```

n	basic operations
1	10
2	22
3	38
4	58
5	82

Counting Operations

number of operations due to outer for loop:

$$\underbrace{2}_{\text{two ops per run}}\underbrace{n}_{\text{number of runs}}+\underbrace{2}_{\text{last time}}$$

Counting Operations

number of operations due to inner for loop:

$$2\frac{n(n+1)}{2} + 2n = n^2 + 3n$$

two ops per run | number of runs | *n* last times, 2 ops each

Counting Operations

number of operations due to bar:

$$\underbrace{2}_{\text{two ops per run}}\underbrace{\frac{n(n+1)}{2}}_{\text{number of runs}} = n^2 + n$$

Counting Operations

total number of operations:

$$\begin{aligned} T(n) &= 2n + 2 + n^2 + 3n + n^2 + n \\ &= 2n^2 + 6n + 2 \\ &\in \Theta(n^2) \end{aligned}$$

Empirical Confirmation

- can we demonstrate this empirically?
  - add a counter
  - print input size and counter results
  - use bash to capture output of repeated runs
  - use a plotting program to display the results
  - when the plot is ok, export a graphic for use in LaTeXfor a report

Empirical Observation

original code:

```
for (unsigned outer = 0; outer < n; outer++)
{
  for (unsigned inner = outer; inner < n; inner++)
  {
    bar(); // assume this takes exactly 2 basic operations
  }
}
```

Empirical Observation

add a counter and output:

```
uint64_t count = 0;
for (unsigned outer = 0; outer < n; outer++)
{
  count += 2; // outer for loop header
  for (unsigned inner = outer; inner < n; inner++)
  {
    count += 2; // inner for loop header
    bar();
    count += 2; // bar's operations
  }
  count += 2; // inner loop header last time
}
count += 2; // outer loop header last time
cout << n << ' ' << count << endl;
```

Run the Program

- first confirm the theoretical analysis
- run the program for $n = 1, 2, 3, 4, 5$, compare to our table

Run the Program

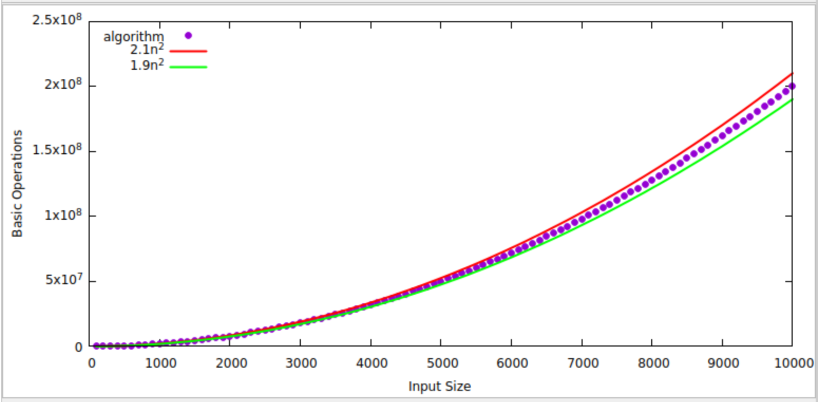
- first confirm the theoretical analysis
- run the program for $n = 1, 2, 3, 4, 5$, compare to our table

- now run the program many times, capture the output

```
for n in $(seq 100 100 10000)
do
  ./bar_count $n
done > bar_result.dat
```

Now Plot

```
$ gnuplot
set key top left
set xlabel "Input Size"
set ylabel "Basic Operations"
plot "bar_result.dat" using 1:2 pointtype 7 title "algorithm", \
     2.1*x**2 linewidth 2 linecolor "red" title "2.1n^2", \
     1.9*x**2 linewidth 2 linecolor "green" title "1.9n^2"
```


algorithm
2.1n2
1.9n2
Basic Operations
2.5x10^8
2x10^8
1.5x10^8
1x10^8
5x10^7
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Input Size

Big-Theta vs Big-Oh and Big-Omega

- so far, all our analyses have resulted in big-Theta
- what determines whether we have a big-Theta or not?
- here are some clues

Big-Theta

- controlled by for loops
- cannot end early
- only one case, or similar best and worst cases
- conditionals have little effect

Big-Oh and Big-Omega

- controlled by while loops
- can end early
- distinct best and worst cases
- conditionals make a big difference

Another Example

```
for (unsigned outer = 0; outer < n; outer++)
{
  if (foo()) // assume foo performs 3 operations
  {
    for (unsigned inner = 0; inner < n; inner++)
    {
      bar(); // assume bar performs 2 operations
    }
  }
  else
  {
    bam(); // assume bam performs 4 operations
  }
}
```

- just like before, what is the exact number of operations performed as a function of n?

Conditional

- now we're stuck, because this is non-deterministic
- the presence of the conditional means there is not an exact function

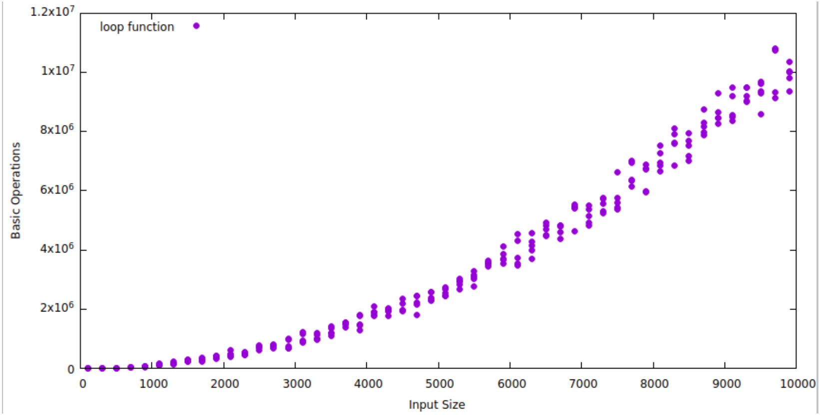
- there is a best case when the conditional is never true
- a worst case when the conditional is always true
- we must count each separately

Best and Worst Cases

```
for (unsigned outer = 0; outer < n; outer++)
{
  if (foo()) // assume foo performs 3 operations
  {
    for (unsigned inner = 0; inner < n; inner++)
    {
      bar(); // assume bar performs 2 operations
    }
  }
  else
  {
    bam(); // assume bam performs 4 operations
  }
}
```

n	ops best case	ops worst case
1	11	13
2	20	32
3	29	59
4	38	94

Visualize



loop function
Basic Operations
Input Size
1.2x10^7
1x10^7
8x10^6
6x10^6
4x10^6
2x10^6
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

Best Case

```
for (unsigned outer = 0; outer < n; outer++)
{
  if (foo()) // assume foo performs 3 operations
  {
    for (unsigned inner = 0; inner < n; inner++)
    {
      bar(); // assume bar performs 2 operations
    }
  }
  else
  {
    bam(); // assume bam performs 4 operations
  }
}
```

$$\begin{aligned} T(n) &\geq 2(n+1) + 3n + 4n \\ &\geq 9n + 2 \\ &\in \Omega(n) \end{aligned}$$

Worst Case

```
for (unsigned outer = 0; outer < n; outer++)
{
  if (foo()) // assume foo performs 3 operations
  {
    for (unsigned inner = 0; inner < n; inner++)
    {
      bar(); // assume bar performs 2 operations
    }
  }
  else
  {
    bam(); // assume bam performs 4 operations
  }
}
```

$$
\begin{aligned}
T(n) &\leq 2(n+1) + 3n + 2n(n+1) + 2n(n) \\
&\leq 4n^2 + 7n + 2 \\
&\in O(n^2)
\end{aligned}
$$

Visualize

- plot the actual program
- with best and worst cases
- with scaled standard functions to illustrate