COMP - 285

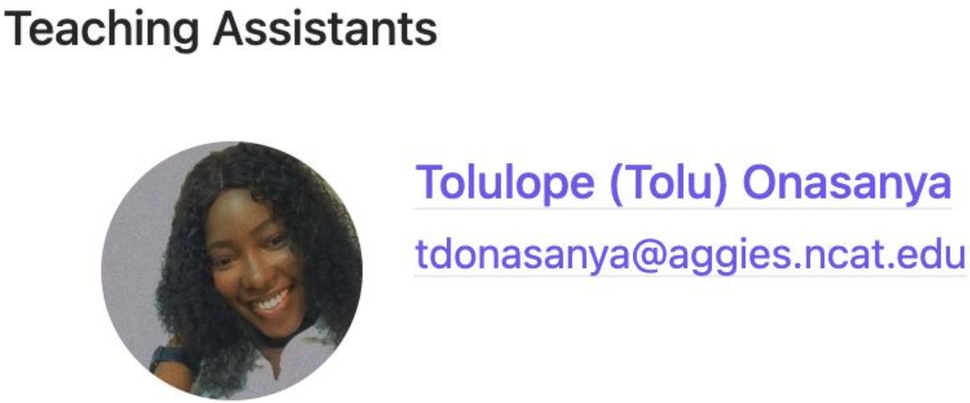
Advanced Analysis of Algorithms

Welcome to COMP 285

Lecture 3: Measuring Performance II

Chris Lucas (cflucas@ncat.edu)

Before that!



Teaching Assistants

Tolulope (Tolu) Onasanya

tdonasanya@aggies.ncat.edu

HW1 is out!

Due 09/06 @ 1:59pm

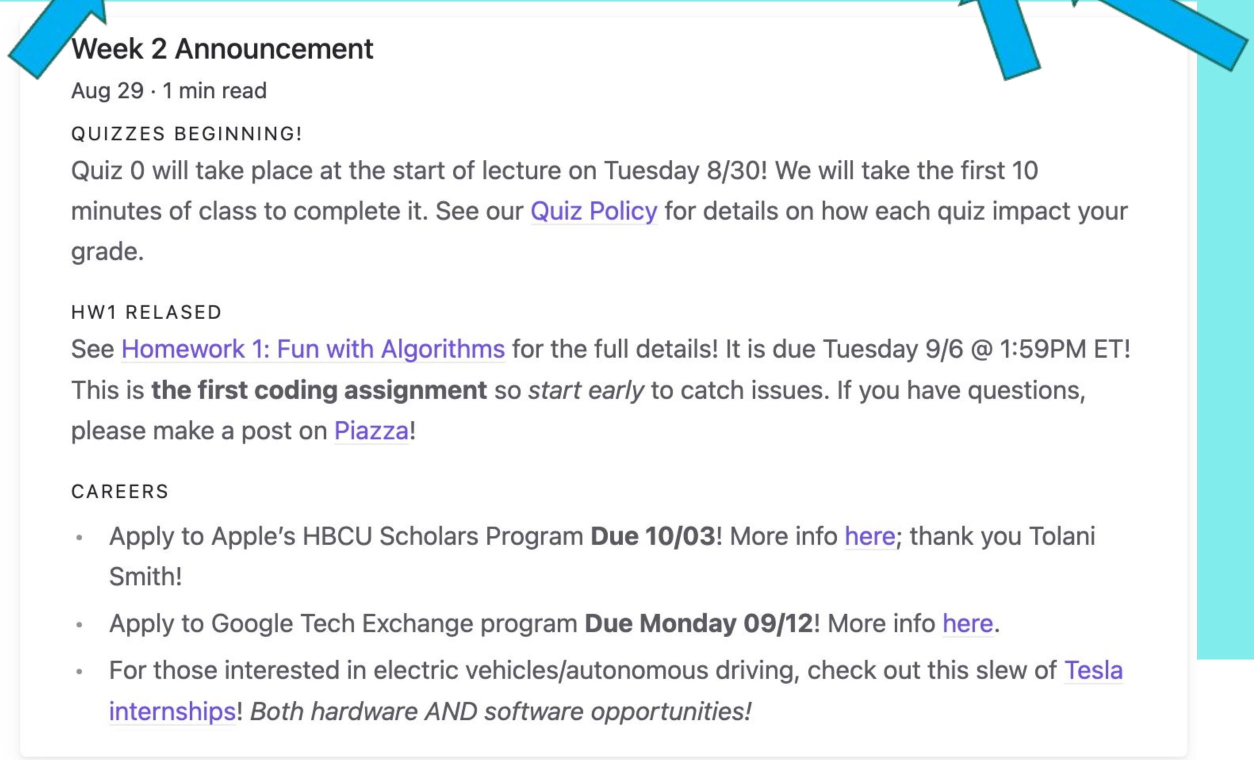
HW0 is being graded!

We’re aiming for O(1 week) runtime on grading :)

Thank you Priya and Tolu!

Piazza for Questions!

comp285-fall22.ml



Week 2 Announcement

Aug 29 · 1 min read

QUIZZES BEGINNING!

Quiz 0 will take place at the start of lecture on Tuesday 8/30! We will take the first 10 minutes of class to complete it. See our Quiz Policy for details on how each quiz impact your grade.

HW1 RELASED

See Homework 1: Fun with Algorithms for the full details! It is due Tuesday 9/6 @ 1:59PM ET! This is **the first coding assignment** so *start early* to catch issues. If you have questions, please make a post on Piazza!

CAREERS

- Apply to Apple's HBCU Scholars Program **Due 10/03**! More info here; thank you Tolani Smith!
- Apply to Google Tech Exchange program **Due Monday 09/12**! More info here.
- For those interested in electric vehicles/autonomous driving, check out this slew of Tesla internships! *Both hardware AND software opportunities!*

Quiz #0!

Lectures 0, 1 and 2

Quiz #0!

shorturl.at/DJLW5

Big Questions!

- How to Big-Oh? (pt. 2)
- How to Big-Oh? (space edition)
- How to Big-Oh? (recursion edition)
- Who really is Big-Oh?

Recall where we ended last lecture…

Big-O Process

1. Define the “input size”
    - What’s our “n”?
    - Is it the length of the vector? Is it the value of an integer?
    - The inputs to the function are a good place to look!
2. Count the number of operations
    - We’ve already practiced this!
3. Simplify
    - Some simplification rules we’ll get into. (n -> inf!)

Concrete Examples

Count the number of operations

```
void doThings(int number) {
 int x = 4;
 int y = x + y;
 std::cout << "hi" << std::endl;
 std::cout << number << std::endl;
}
```

1. Define the “input size”
2. Count the number of operations
3. Simplify

Count the number of operations

```
void doThings(int number) {
 int x = 4;
 int y = x + y;
 std::cout << "hi" << std::endl;
 std::cout << number << std::endl;
}
```

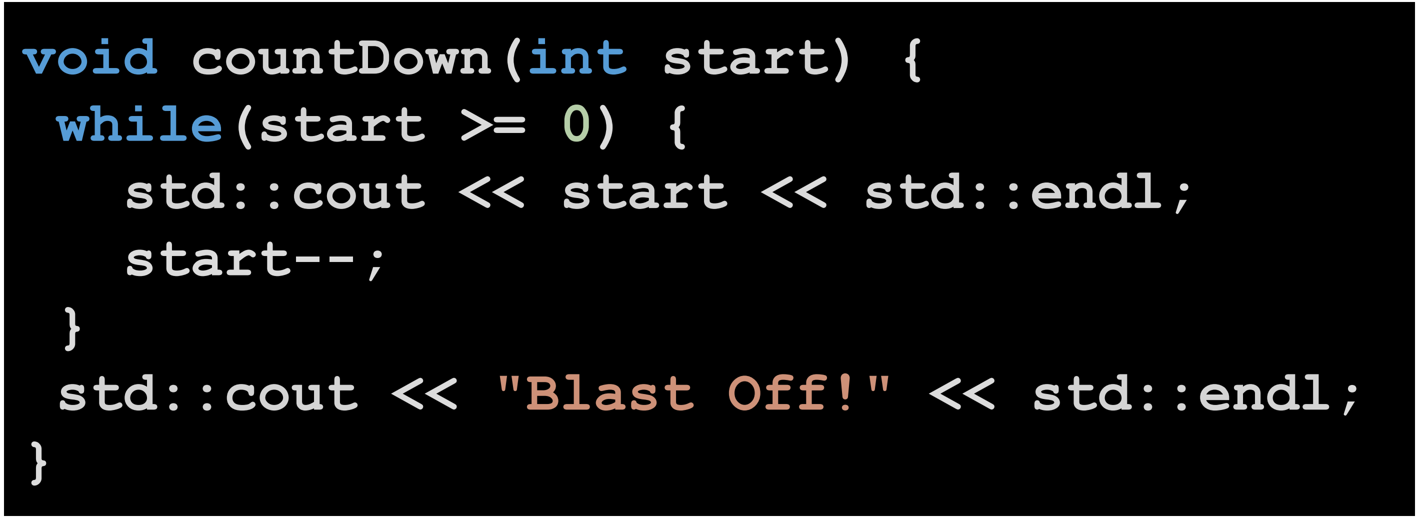
1. Define the “input size” **The value of “number” variable**
2. Count the number of operations **4**
3. Simplify **O(1)**

What’s the runtime?

```
void countDown(int start) {
 while(start >= 0) {
   std::cout << start << std::endl;
   start--;
 }
 std::cout << "Blast Off!" << std::endl;
}
```

1. Define the “input size” n
2. Count the number of operations
3. Simplify

What’s the runtime?



```
void countDown(int start) {
 while(start >= 0) {
   std::cout << start << std::endl;
   start--;
 }
 std::cout << "Blast Off!" << std::endl;
}
```

1. Define the “input size” n

The value of “start” variable

2. Count the number of operations

3N+4

3. Simplify

O(N)

What’s the runtime?

```
void printElements(const std::vector<int>& vec) {
 std::cout << "Printing..." << std::endl;
 for(int i = 0; i < vec.size(); i++) {
   for(int j = 0; j < vec.size(); j++) {
     std::cout << vec[i] << " " << vec[j] << " ";
   }
 }
 std::cout << std::endl;
}
```

1. Define the “input size”
2. Count the number of operations
3. Simplify

What’s the runtime?

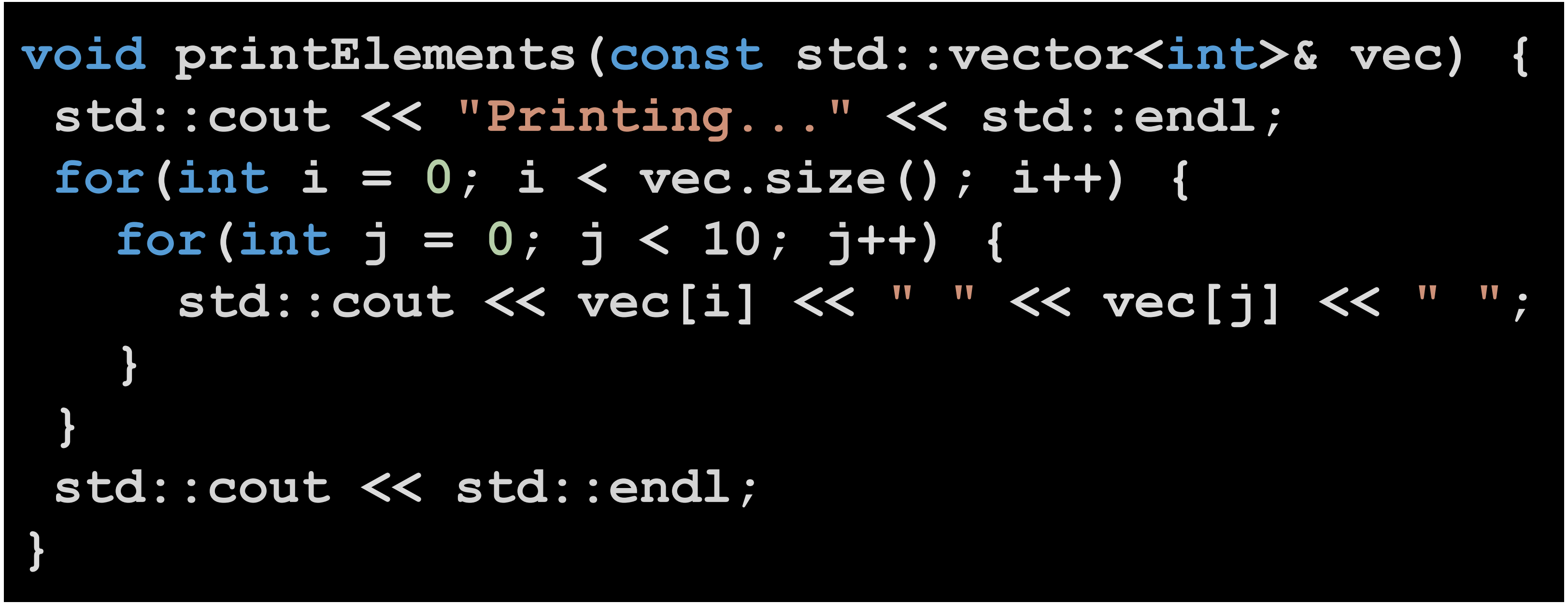
```cpp
void printElements(const std::vector<int>& vec) {
 std::cout << "Printing..." << std::endl;
 for(int i = 0; i < vec.size(); i++) {
   for(int j = 0; j < vec.size(); j++) {
     std::cout << vec[i] << " " << vec[j] << " ";
   }
 }
 std::cout << std::endl;
}
```

1. Define the “input size” **Number of elements in vec**
2. Count the number of operations **???**
3. Simplify **$O(N^2)$**

Simplification Rules

1. Simplify constant time:
    - 23 -> O(1)
2. Drop multiplicative constants
    - 7 * N -> O(N)
3. Drop all lower-order terms:
    - N + N^2 -> O(N^2)

What’s the runtime?



```cpp
void printElements(const std::vector<int>& vec) {
 std::cout << "Printing..." << std::endl;
 for(int i = 0; i < vec.size(); i++) {
   for(int j = 0; j < 10; j++) {
     std::cout << vec[i] << " " << vec[j] << " ";
   }
 }
 std::cout << std::endl;
}
```

Big Questions!

- How to Big-Oh? (pt. 2) ⬅
- How to Big-Oh? (space edition)
- How to Big-Oh? (recursion edition)
- Who really is Big-Oh?

What’s the runtime?

```
int doSomethingWithTwoVecs(const std::vector<int>& vecA,
                           const std::vector<int>& vecB) {
 int value = 0;
 for (int i = 0; i < vecA.size(); i++) {
   for (int j = 0; j < vecB.size(); j++) {
     if (vecA[i] == vecB[j]) {
       value += vecA[i];
     }
   }
 }
 return value;
}
```

What’s the runtime?

```
int doSomethingWithTwoVecs(const std::vector<int>& vecA,
                           const std::vector<int>& vecB) {
 int value = 0;
 for (int i = 0; i < vecA.size(); i++) {
   for (int j = 0; j < vecB.size(); j++) {
     if (vecA[i] == vecB[j]) {
       value += vecA[i];
     }
   }
 }
 return value;
}
```

A times

B times

O(A*B)

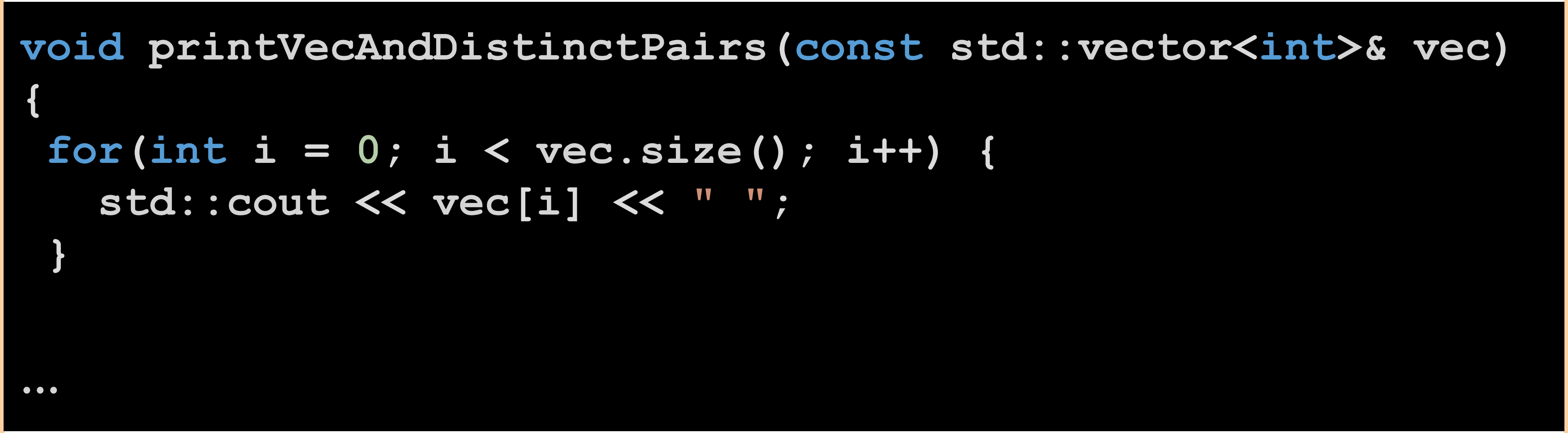
What’s the runtime?

```
void printVecAndDistinctPairs(const std::vector<int>& vec)
{
 for(int i = 0; i < vec.size(); i++) {
   std::cout << vec[i] << " ";
 }

 for(int i = 0; i < vec.size(); i++) {
   for(int j = i + 1; j < vec.size(); j++) {
     std::cout << vec[i] << " " << vec[j] << std::endl;
   }
 }
}
```

What’s the runtime?

```
void printVecAndDistinctPairs(const std::vector<int>& vec)
{
 for(int i = 0; i < vec.size(); i++) {
   std::cout << vec[i] << " ";
 }


…
```

What’s the runtime?



```
void printVecAndDistinctPairs(const std::vector<int>& vec)
{
 for(int i = 0; i < vec.size(); i++) {
   std::cout << vec[i] << " ";
 }


…
```

O(N), linear time

What’s the runtime?

```
void printVecAndDistinctPairs(const std::vector<int>& vec)

…
 for(int i = 0; i < vec.size(); i++) {
   for(int j = i + 1; j < vec.size(); j++) {
     std::cout << vec[i] << " " << vec[j] << std::endl;
   }
 }
}
```

What’s the runtime?

```
void printVecAndDistinctPairs(const std::vector<int>& vec)

…
 for(int i = 0; i < vec.size(); i++) {
   for(int j = i + 1; j < vec.size(); j++) {
     std::cout << vec[i] << " " << vec[j] << std::endl;
   }
 }
}
```

$O(N^2)$, quadratic time

What’s the runtime?

```cpp
void printVecAndDistinctPairs(const std::vector<int>& vec)
{
 for(int i = 0; i < vec.size(); i++) {
   std::cout << vec[i] << " ";
 }

 for(int i = 0; i < vec.size(); i++) {
   for(int j = i + 1; j < vec.size(); j++) {
     std::cout << vec[i] << " " << vec[j] << std::endl;
   }
 }
}
```

What’s the runtime?

```cpp
void printVecAndDistinctPairs(const std::vector<int>& vec)
{
 for(int i = 0; i < vec.size(); i++) {
   std::cout << vec[i] << " ";
 }

 for(int i = 0; i < vec.size(); i++) {
   for(int j = i + 1; j < vec.size(); j++) {
     std::cout << vec[i] << " " << vec[j] << std::endl;
   }
 }
}
```

O(N)

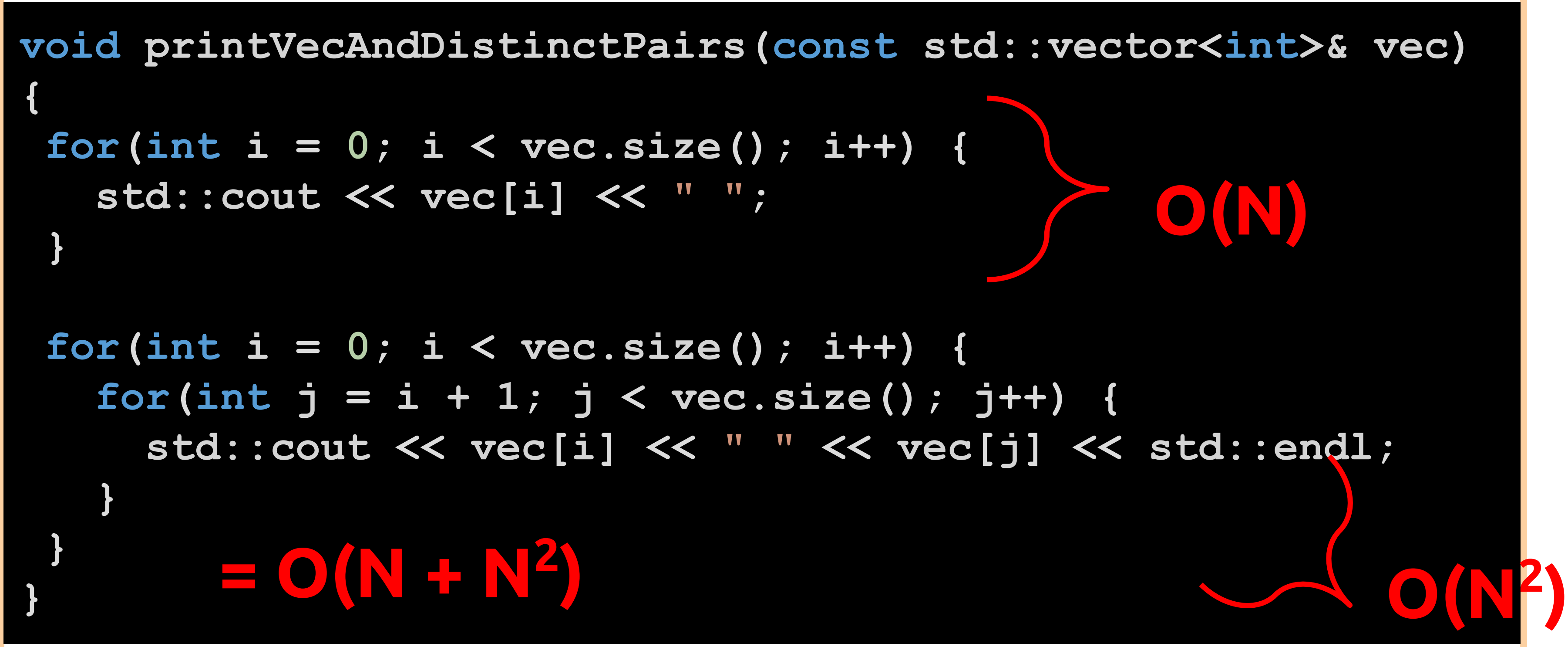
What’s the runtime?

```
void printVecAndDistinctPairs(const std::vector<int>& vec)
{
 for(int i = 0; i < vec.size(); i++) {
   std::cout << vec[i] << " ";
 }

 for(int i = 0; i < vec.size(); i++) {
   for(int j = i + 1; j < vec.size(); j++) {
     std::cout << vec[i] << " " << vec[j] << std::endl;
   }
 }
}
```

O(N)

$O(N^2)$

What’s the runtime?



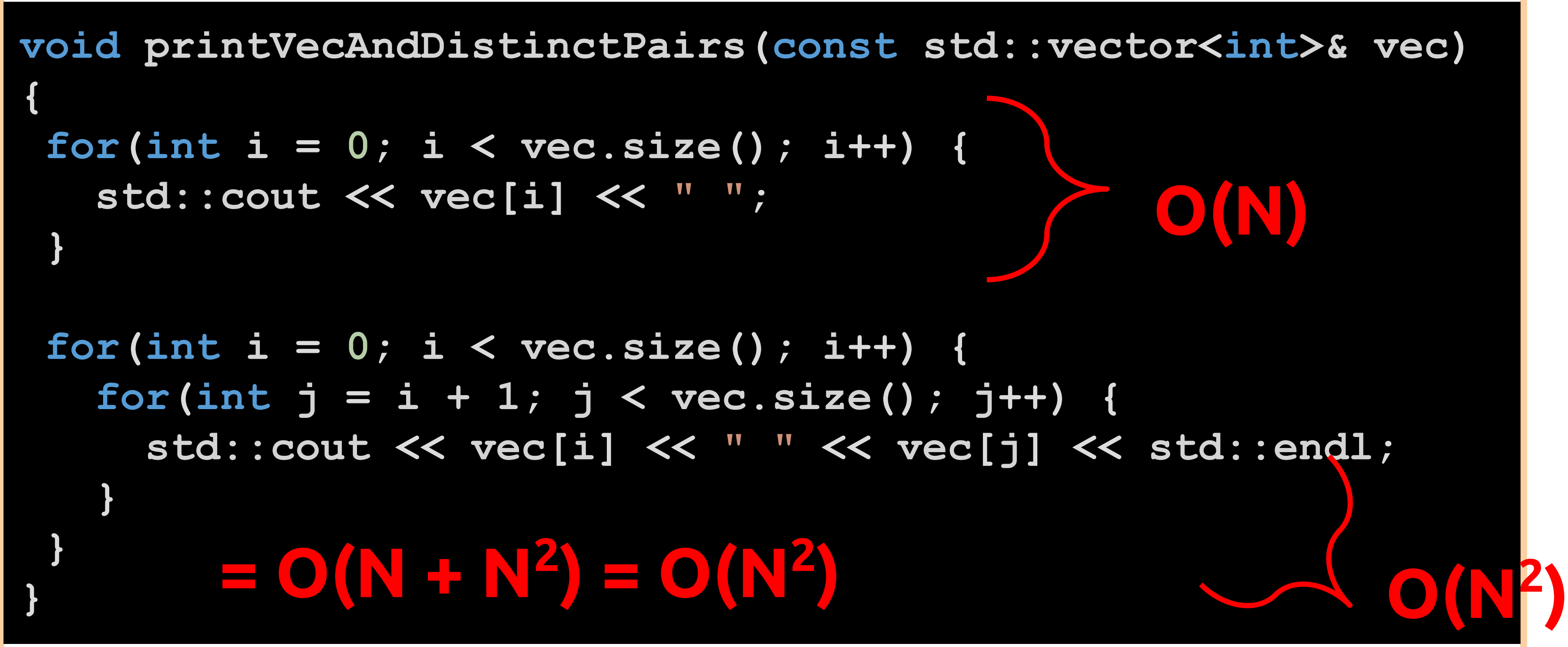
```
void printVecAndDistinctPairs(const std::vector<int>& vec)
{
 for(int i = 0; i < vec.size(); i++) {
   std::cout << vec[i] << " ";
 }

 for(int i = 0; i < vec.size(); i++) {
   for(int j = i + 1; j < vec.size(); j++) {
     std::cout << vec[i] << " " << vec[j] << std::endl;
   }
 }
}
```

$O(N)$

$O(N^2)$

$= O(N + N^2)$

What’s the runtime?



```
void printVecAndDistinctPairs(const std::vector<int>& vec)
{
 for(int i = 0; i < vec.size(); i++) {
   std::cout << vec[i] << " ";
 }

 for(int i = 0; i < vec.size(); i++) {
   for(int j = i + 1; j < vec.size(); j++) {
     std::cout << vec[i] << " " << vec[j] << std::endl;
   }
 }
}
```

$O(N)$

$O(N^2)$

$= O(N + N^2) = O(N^2)$

Poll - What’s the Big-O Runtime?

```
void doSomething(const std::vector<int>& vec) {

 for(int i = 0; i < vec.size(); i++) {
   for(int j = 0; j < 100; j++) {
     for(int k = 0; k < vec.size(); k++) {
       std::cout << vec[i] * vec[j] * vec[k] << std::endl;
     }
   }
 }
}
```

1. O(1) 2. O(N) 3. O(N log N) 4. $O(N^2)$ 5. $O(N^3)$

Poll - What’s the Big-O Runtime?

```
void doSomething(const std::vector<int>& vec) {

 for(int i = 0; i < vec.size(); i++) {
   for(int j = 0; j < 100; j++) {
     for(int k = 0; k < vec.size(); k++) {
       std::cout << vec[i] * vec[j] * vec[k] << std::endl;
     }
   }
 }
}
```

1. $O(1)$ 2. $O(N)$ 3. $O(N \log N)$ 4. [✓ $O(N^2)$] 5. $O(N^3)$

Poll - What’s the Big-O Runtime?

An algorithm prints every other element in a vector of size N. (The for-loop increments by += 2) What is its runtime?

1. O(1/2)
2. O(1)
3. O(N)
4. O(N/2)
5. O(N^2)

Poll - What’s the Big-O Runtime?

An algorithm prints every other element in a vector of size N. (The for-loop increments by += 2) What is its runtime?

1. O(1/2)
2. O(1)
3. [✓ O(N)]
4. O(N/2)
5. $O(N^2)$

Poll - What's the Big-O Runtime?

An algorithm that takes in a distance in miles, prints out the numbers from 1 to 1,000,000,000, then converts the miles to kilometers. What is its runtime?

1. O(1)
2. O(1,000,000,000)
3. O(N)
4. $O(N^2)$
5. $O(N^3)$

Poll - What’s the Big-O Runtime?

An algorithm that takes in a distance in miles, prints out the numbers from 1 to 1,000,000,000, then converts the miles to kilometers. What is its runtime?

1. [✓ $O(1)$]
2. $O(1{,}000{,}000{,}000)$
3. $O(N)$
4. $O(N^2)$
5. $O(N^3)$

Wait!

STOP!

Big Questions!

- How to Big-Oh? (pt. 2)
- How to Big-Oh? (space edition)
- How to Big-Oh? (recursion edition)
- Who really is Big-Oh?

Space Complexity

- We can use Big-O to describe the amount of additional space “units” we use.

Space Complexity

- We can use Big-O to describe the amount of additional space "units" we use.

- When we declare primitive types, that takes constant space (i.e. int x = 4 is O(1)).

Space Complexity

- We can use Big-O to describe the amount of additional space "units" we use.

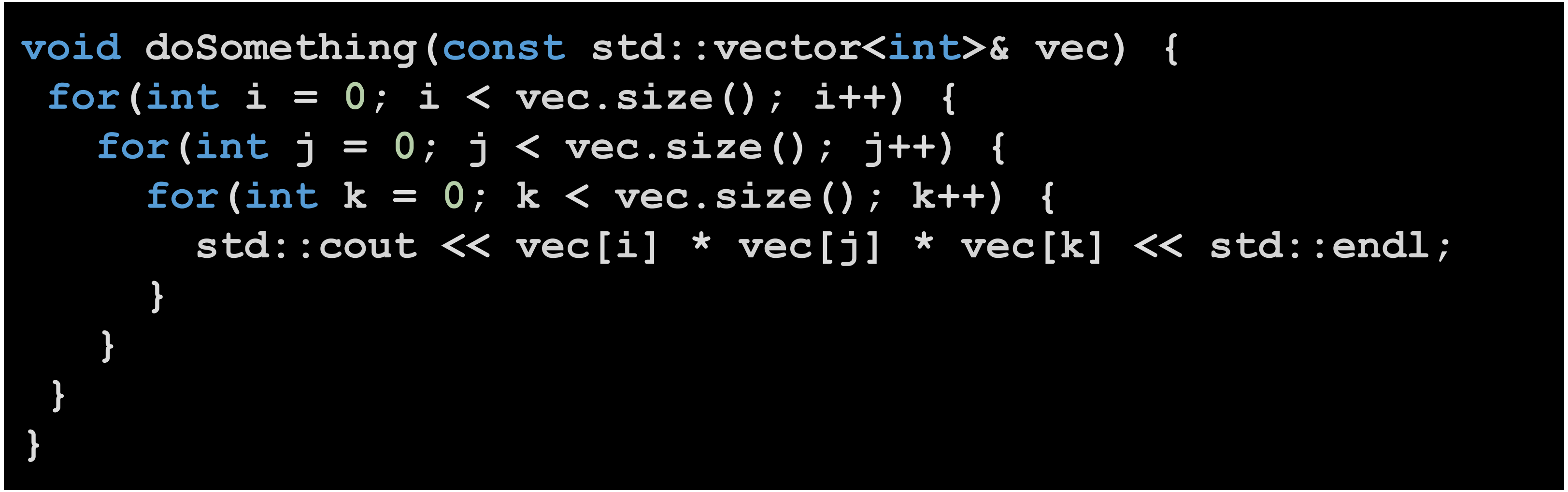
- When we declare primitive types, that takes constant space (i.e. int x = 4 is O(1)).

- In general, whenever you create data structures that depend on the size of your input, you'll have to keep track of usage.

Space Complexity

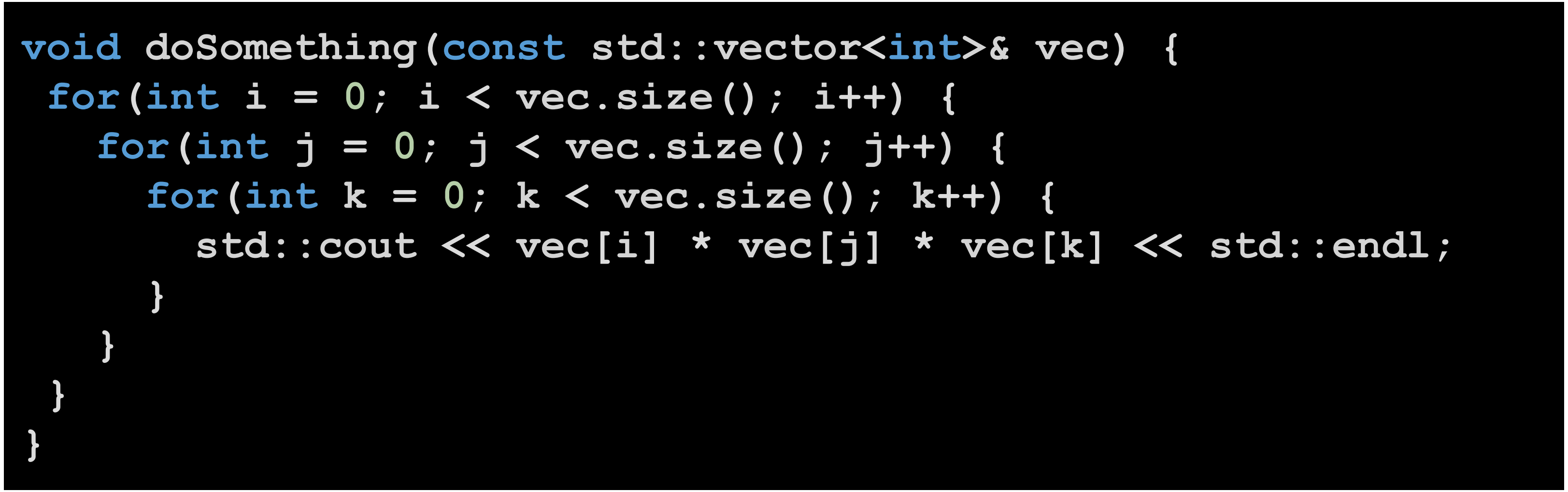
- We can use Big-O to describe the amount of additional space “units” we use.
- When we declare primitive types, that takes constant space (i.e. int x = 4 is O(1)).
- In general, whenever you create data structures that depend on the size of your input, you’ll have to keep track of usage.
- Recursive function + stack frame considerations

What’s the space complexity?



```
void doSomething(const std::vector<int>& vec) {
 for(int i = 0; i < vec.size(); i++) {
   for(int j = 0; j < vec.size(); j++) {
     for(int k = 0; k < vec.size(); k++) {
       std::cout << vec[i] * vec[j] * vec[k] << std::endl;
     }
   }
 }
}
```

What’s the space complexity?



```
void doSomething(const std::vector<int>& vec) {
 for(int i = 0; i < vec.size(); i++) {
   for(int j = 0; j < vec.size(); j++) {
     for(int k = 0; k < vec.size(); k++) {
       std::cout << vec[i] * vec[j] * vec[k] << std::endl;
     }
   }
 }
}
```

O(1) space complexity,
O(N^3) time complexity

What’s the space complexity?

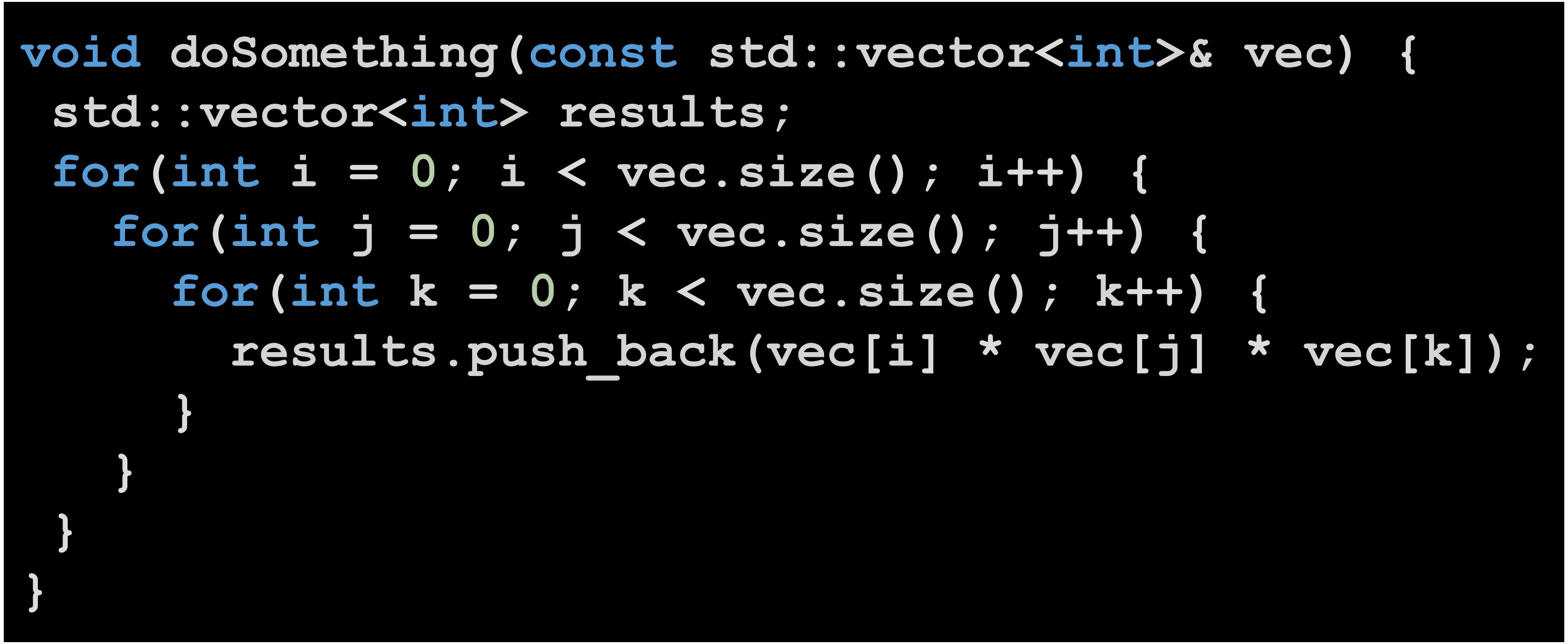
```
void doSomething(const std::vector<int>& vec) {
 std::vector<int> results;
 for(int i = 0; i < vec.size(); i++) {
   results.push_back(vec[i]);
 }
}
```

What's the space complexity?

```
void doSomething(const std::vector<int>& vec) {
 std::vector<int> results;
 for(int i = 0; i < vec.size(); i++) {
   results.push_back(vec[i]);
 }
}
```

O(N) space complexity,
O(N) time complexity

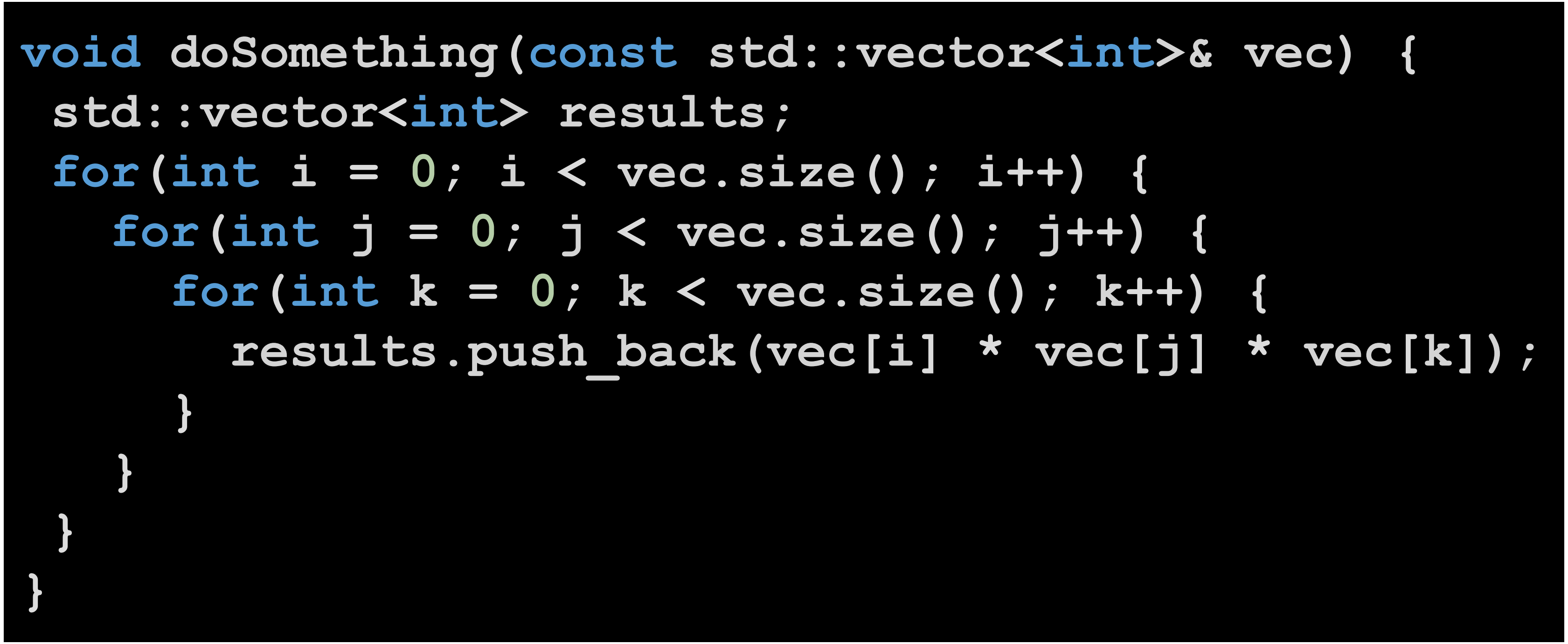
Poll - What’s the space complexity?



```
void doSomething(const std::vector<int>& vec) {
 std::vector<int> results;
 for(int i = 0; i < vec.size(); i++) {
   for(int j = 0; j < vec.size(); j++) {
     for(int k = 0; k < vec.size(); k++) {
       results.push_back(vec[i] * vec[j] * vec[k]);
     }
   }
 }
}
```

1. O(1) 2. O(N) 3. O(N log N) 4. O(N^2) 5. O(N^3)

Poll - What’s the space complexity?



```
void doSomething(const std::vector<int>& vec) {
 std::vector<int> results;
 for(int i = 0; i < vec.size(); i++) {
   for(int j = 0; j < vec.size(); j++) {
     for(int k = 0; k < vec.size(); k++) {
       results.push_back(vec[i] * vec[j] * vec[k]);
     }
   }
 }
}
```

1. $O(1)$ 2. $O(N)$ 3. $O(N \log N)$ 4. $O(N^2)$ 5. [✓ $O(N^3)$]

What’s the space complexity?

```
void doSomething(const std::vector<int>& vec) {
 for(int i = 0; i < vec.size(); i++) {
   for(int j = 0; j < vec.size(); j++) {
     std::vector<int> results;
     for(int k = 0; k < vec.size(); k++) {
       results.push_back(vec[i] * vec[j] * vec[k]);
     }
   }
 }
}
```

1. O(1) 2. O(N) 3. O(N log N) 4. $O(N^2)$ 5. $O(N^3)$

What’s the space complexity?

```
void doSomething(const std::vector<int>& vec) {
 for(int i = 0; i < vec.size(); i++) {
   for(int j = 0; j < vec.size(); j++) {
     std::vector<int> results;
     for(int k = 0; k < vec.size(); k++) {
       results.push_back(vec[i] * vec[j] * vec[k]);
     }
   }
 }
}
```

1. O(1) 2. [✓ O(N)] 3. O(N log N) 4. O(N^2) 5. O(N^3)

COMP - 285

Advanced Analysis of Algorithms

Welcome to COMP 285

Lecture 3: Measuring Performance II

Chris Lucas (cflucas@ncat.edu)