CS 106B, Lecture 5
Stacks and Big O

reading:
*Programming Abstractions in C++, Chapter 4-5*
Plan for Today

• Analyzing algorithms using **Big O** analysis
  – Understand what makes an algorithm "good" and how to compare algorithms
• Another type of collection: the **Stack**
Big O
Big O Intuition

• Lots of different ways to solve a problem; which is best?
• Measure algorithmic **efficiency**
  – how many resources (time? memory? etc.) does the program use
  – We'll focus on time
• Idea: algorithms are better if they take less time
• Problem: amount of time a program takes is variable
  – Depends on what computer you're using, what other programs are running, if your laptop is plugged in…
Big O

- Idea: assume each statement of code takes some unit of time
  - for the purposes of this class, that unit doesn't matter
- We can count the number of units of time and get the runtime
- Sometimes, the number of statements depends on the input – we'll say the input size is N
statement1;  // runtime = 1

for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
        statement2;
    }
}

for (int i = 1; i <= N; i++) {
    statement3;
    statement4;
    statement5;
}  // total = N^2 + 3N + 1
The actual constant doesn't matter (remember that we haven't even specified how much a unit of time is) – so we get rid of the constants: $N^2 + 3N + 1 \rightarrow N^2 + N + 1$

Only the biggest power of $N$ matters: $N^2 + N + 1 \rightarrow N^2$

– The biggest term grows so much faster than the other terms that the runtime of that term "dominates"

– Another way to think about it: $N^2 + N + 1 < 2N^2$ when $N$ is big, and we already said we don't care about constants

We would then say the code snippet has $O(N^2)$ runtime
Finding Big O

- Work from the innermost indented code out
- Realize that some code statements are more costly than others
  - It takes $O(N^2)$ time to call a function with runtime $O(N^2)$, even though calling that function is only one line of code
- Nested code multiplies
- Code at the same indentation level adds
What is the Big O?

```cpp
int sum = 0;
for (int i = 1; i < 100000; i++) {
    for (int j = 1; j <= i; j++) {
        for (int k = 1; k <= N; k++) {
            sum++;
        }
    }
}

Vector<int> v;
for (int x = 1; x <= N; x += 2) {
    v.insert(0, x);
}

cout << v << endl;
```
Complexity Classes

- complexity class: A category of algorithmic efficiency based on the algorithm's relationship to the input size "N".

<table>
<thead>
<tr>
<th>Class</th>
<th>Big-Oh</th>
<th>If you double N, ...</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>$O(1)$</td>
<td>unchanged</td>
<td>10ms</td>
</tr>
<tr>
<td>logarithmic</td>
<td>$O(\log_2 N)$</td>
<td>increases slightly</td>
<td>175ms</td>
</tr>
<tr>
<td>linear</td>
<td>$O(N)$</td>
<td>doubles</td>
<td>3.2 sec</td>
</tr>
<tr>
<td>log-linear</td>
<td>$O(N \log_2 N)$</td>
<td>slightly more than doubles</td>
<td>11 sec</td>
</tr>
<tr>
<td>quadratic</td>
<td>$O(N^2)$</td>
<td>quadruples</td>
<td>1 min 42 sec</td>
</tr>
<tr>
<td>quad-linear</td>
<td>$O(N^2 \log_2 N)$</td>
<td>slightly more than quadruple</td>
<td>8 min</td>
</tr>
<tr>
<td>cubic</td>
<td>$O(N^3)$</td>
<td>multiplies by 8</td>
<td>55 min</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>exponential</td>
<td>$O(2^N)$</td>
<td>multiplies drastically</td>
<td>$5 \times 10^{61}$ years</td>
</tr>
<tr>
<td>factorial</td>
<td>$O(N!)$</td>
<td>multiplies drastically</td>
<td>$10^{200}$ years</td>
</tr>
</tbody>
</table>
Announcements

• Assignment 1 due **Thursday at 5PM**
• Shreya will be guest-presenting tomorrow!
• No class on July 4\(^{th}\) – if you have section, either attend a Thursday or Friday section or watch the videoed section and email your SL a summary
  – No LaIR either

• **SCPD Exam Form Typo:** final exam is on August 18 NOT August 25; please check your Stanford email/Piazza for details (only SCPD students who did not indicate they'd take the exam on August 18 were emailed)
ADTs – the Story so Far

Start → How many dimensions of data do I have?

- Two → Grid
- One → Vector
A new ADT: the Stack

• A specialized data structure that only allows a user to add, access, and remove the last element
  – "Last In, First Out"
  – Super fast (O(1)) for these operations
    • Built directly into the hardware
• Main operations:
  – **push(value)**: add an element to the end of the stack
  – **pop()**: remove and return the last element in the stack
  – **peek()**: return (but do not remove) the last element in the stack
"Stacked" examples

- Real life
  - Pancakes
  - Clothes
  - Plates in the dining hall

- In computer science
  - Function calls
  - Keeping track of edits to undo or pages visited on a website to go back to (you'll implement this in assignment 5)

source: https://c2.staticflickr.com/8/7583/15638298618_104af94267_b.jpg
Stack Syntax

```cpp
#include "stack.h"

Stack<int> nums;
nums.push(1);
nums.push(3);
nums.push(5);
cout << nums.peek() << endl; // 5
cout << nums << endl; // {1, 3, 5}
nums.pop(); // nums = {1, 3}
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.isEmpty()</td>
<td>O(1)</td>
<td>returns true if stack has no elements</td>
</tr>
<tr>
<td>s.peek()</td>
<td>O(1)</td>
<td>returns <strong>top</strong> value without removing it; throws an error if stack is empty</td>
</tr>
<tr>
<td>s.pop()</td>
<td>O(1)</td>
<td>removes <strong>top</strong> value and returns it; throws an error if stack is empty</td>
</tr>
<tr>
<td>s.push(value);</td>
<td>O(1)</td>
<td>places given value on <strong>top</strong> of stack</td>
</tr>
<tr>
<td>s.size()</td>
<td>O(1)</td>
<td>returns number of elements in stack</td>
</tr>
</tbody>
</table>
Stack limitations/idioms

- You cannot access a stack's elements by index.

```java
Stack<int> s;
...
for (int i = 0; i < s.size(); i++) {
  do something with s[i]; // does not compile
}
```

- Instead, you pull elements out of the stack one at a time.

- **common idiom: Pop each element until the stack is empty.**

  ```java
  // process (and empty!) an entire stack
  while (!s.isEmpty()) {
    do something with s.pop();
  }
  ```
Sentence Reversal

• Goal: print the words of a sentence in reverse order
  – "Hello my name is Inigo Montoya" -> "Montoya Inigo is name my Hello"
  – "Inconceivable" -> "Inconceivable"
• Assume characters are only letters and spaces
• How could we use a Stack?
void printSentenceReverse(const string &sentence) {
    Stack<string> wordStack;
    for (char c : sentence) {
        if (c == SPACE) {
            wordStack.push(word);
            word = ""; // reset
        } else {
            word += c;
        }
    }
    if (word != "") {
        wordStack.push(word);
    }
    cout << " New sentence: ";
    while (!wordStack.isEmpty()) {
        word = wordStack.pop();
        cout << word << SPACE;
    }
    cout << endl;
}
ADTs – the Story so Far

Start

How many dimensions of data do I have?

Two

Grid

One

Which elements do I need to access?

Frequent looping or middle elements

Vector

Last element

Stack