Analyzing Algorithms

- Consider three solutions to weekly 1, each is also the foundation of a solution to anagram part 1 (class notes for January 16)
  - Sort, then scan looking for changes
  - Insert into StringSet, then count each unique string
  - Find unique elements without sorting, sort these, then count each unique string

- We want to discuss trade-offs of these solutions
  - Ease to develop, debug, verify
  - Runtime efficiency
  - Vocabulary for discussion

What is big-Oh about? (preview)

- Intuition: avoid details when they don’t matter, and they don’t matter when input size (N) is big enough
  - For polynomials, use only leading term, ignore coefficients
    - $y = 3x$
    - $y = 6x - 2$
    - $y = 15x + 44$
    - $y = x^2$
    - $y = x^2 - 6x + 9$
    - $y = 3x^2 + 4x$

- The first family is $O(n)$, the second is $O(n^2)$
  - Intuition: family of curves, generally the same shape
  - More formally: $O(f(n))$ is an upper-bound, when n is large enough the expression $cf(n)$ is larger
  - Intuition: linear function: double input, double time, quadratic function: double input, quadruple the time

More on O-notation, big-Oh

- Big-Oh hides/obscures some empirical analysis, but is good for general description of algorithm
  - Allows us to compare algorithms in the limit
    - 20N hours vs $N^2$ microseconds: which is better?
- O-notation is an upper-bound, this means that $N$ is $O(N)$, but it is also $O(N^2)$; we try to provide tight bounds.
  - Formally:
    - A function $g(N)$ is $O(f(N))$ if there exist constants $c$ and $n$ such that $g(N) < cf(N)$ for all $N > n$

Big-Oh calculations from code

- Search for element in vector:
  - What is complexity of code (using O-notation)?
  - What if array doubles, what happens to time?

```cpp
for(int k=0; k < a.size(); k++) {
    if (a[k] == target) return true;
}
return false;
```

- Complexity if we call N times on M-element vector?
  - What about best case? Average case? Worst case?
Big-Oh calculations again

- Weekly problem: first string to occur 3 times
  - What is complexity of code (using O-notation)?

```c
for(int k=0; k < a.size(); k++) {
    int count = 0;
    for(int j=0; j <= k; k++) {
        if (a[j] == a[k]) count++;
    }
    if (count >= 3) return a[k];
}
return ""; // nothing occurs three times
```

- What happens to time if array doubles in size?
  - $1 + 2 + 3 + \ldots + n-1$, why and what’s O-notation?

Amortization: Expanding Vectors

- Expand capacity of vector when `push_back` called
- Calling `push_back` N times, doubling capacity as needed

<table>
<thead>
<tr>
<th>Item #</th>
<th>Resizing cost</th>
<th>Cumulative cost</th>
<th>Resizing Cost per item</th>
<th>Capacity After push_back</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3-4</td>
<td>4</td>
<td>6</td>
<td>1.5</td>
<td>4</td>
</tr>
<tr>
<td>5-8</td>
<td>8</td>
<td>14</td>
<td>1.75</td>
<td>8</td>
</tr>
</tbody>
</table>

- $2^{n+1} - 2^{n-1}$
- $2^{n+1}$ around 2
- $2^n$

- What if we grow size by one each time?

Some helpful mathematics

- $1 + 2 + 3 + 4 + \ldots + N$
  - $N(N+1)/2$, exactly $N^2/2 + N/2$ which is $O(N^2)$ why?
- $N + N + N + \ldots + N$ (total of N times)
  - $N^2$ which is $O(N^2)$
- $N + N + N + \ldots + N + \ldots + N + \ldots + N$ (total of 3N times)
  - $3N^2$ which is $O(N^2)$
- $1 + 2 + 4 + \ldots + 2^N$
  - $2^{n+1} - 1 = 2 \times 2^n - 1$ which is $O(2^n)$

- Impact of last statement on adding $2^n+1$ elements to a vector
  - $1 + 2 + \ldots + 2^n + 2^{n+1} = 2^{n+2} - 1 = 4 \times 2^n - 1$ which is $O(2^n)$

Running times @ $10^6$ instructions/sec

<table>
<thead>
<tr>
<th>N</th>
<th>$O(log N)$</th>
<th>$O(N)$</th>
<th>$O(N \log N)$</th>
<th>$O(N^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.000003</td>
<td>0.00001</td>
<td>0.000033</td>
<td>0.0001</td>
</tr>
<tr>
<td>100</td>
<td>0.000007</td>
<td>0.00010</td>
<td>0.000664</td>
<td>0.1000</td>
</tr>
<tr>
<td>1,000</td>
<td>0.00010</td>
<td>0.01000</td>
<td>0.010000</td>
<td>1.0</td>
</tr>
<tr>
<td>10,000</td>
<td>0.000013</td>
<td>0.01000</td>
<td>0.132900</td>
<td>1.7 min</td>
</tr>
<tr>
<td>100,000</td>
<td>0.000017</td>
<td>0.10000</td>
<td>1.661000</td>
<td>2.78 hr</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.000020</td>
<td>1.0</td>
<td>19.9</td>
<td>11.6 day</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>0.000030</td>
<td>16.7 min</td>
<td>18.3 hr</td>
<td>318 centuries</td>
</tr>
</tbody>
</table>