### Analysis: Algorithms and Data Structures

- We need a vocabulary to discuss performance and to reason about alternative algorithms and implementations
  - It’s faster! It’s more elegant! It’s safer! It’s cooler!
- We need empirical tests and analytical/mathematical tools
  - Given two methods, which is better? Run them to check.
    - 30 seconds vs. 3 seconds, easy. 5 hours vs. 2 minutes, harder
    - What if it takes two weeks to implement the methods?
  - Use mathematics to analyze the algorithm,
  - The implementation is another matter, cache, compiler optimizations, OS, memory,...

---

### Jaron Lanier

Jaron Lanier is a computer scientist, composer, visual artist, and author. He coined the term “Virtual Reality”... he co-developed the first implementations of virtual reality applications in surgical simulation, vehicle interior prototyping, virtual sets for television production, and assorted other areas

“What’s the difference between a bug and a variation or an imperfection? If you think about it, if you make a small change to a program, it can result in an enormous change in what the program does. If nature worked that way, the universe would crash all the time.”

Lanier has no academic degrees

---

### Recursion and recurrences

- Why are some functions written recursively?
  - Simpler to understand, but ...
  - Mt. Everest reasons
- Are there reasons to prefer iteration?
  - Better optimizer: programmer/scientist v. compiler
  - Six of one? Or serious differences
    - “One person’s meat is another person’s poison”
    - “To each his own”, “Chacun a son gout”, ...
- Complexity (big-Oh) for iterative and recursive functions
  - How to determine, estimate, intuit

---

### What’s the complexity of this code?

```java
// first and last are integer indexes, list is List
int lastIndex = first;
Comparable pivot = list.get(first);
for(int k=first+1; k <= last; k++){
  Comparable ko = list.get(k);
  if (ko.compareTo(pivot) <= 0){
    lastIndex++;
    Collections.swap(list, lastIndex, k);
  }
}
```

- What is big-Oh cost of a loop that visits n elements of a vector?
  - Depends on loop body, if body O(1) then ...
  - If body is O(n) then ...
  - If body is O(k) for k in [0..n) then ...
**FastFinder.findHelper**

```java
class FastFinder {
    private Object findHelper(ArrayList<Comparable> list, int first, int last, int kindex) {
        for (int k=first+1; k <= last; k++) {
            Comparable ko = list.get(k);
            Collections.swap(list, lastIndex, k);
        }
        Collections.swap(list, lastIndex, first);
        return findHelper(list, last, kindex);
    }
}
```

**Different measures of complexity**

- **Worst case**
  - Gives a good upper-bound on behavior
  - Never get worse than this
  - Drawbacks?

- **Average case**
  - What does average mean?
  - Averaged over all inputs? Assuming uniformly distributed random data?
  - Drawbacks?

- **Best case**
  - Linear search, useful?

**Multiplying and adding big-Oh**

- Suppose we do a linear search then we do another one
  - What is the complexity?
  - If we do 100 linear searches?
  - If we do n searches on a vector of size n?

- What if we do binary search followed by linear search?
  - What are big-Oh complexities? Sum?
  - What about 50 binary searches? What about n searches?

- What is the number of elements in the list (1,2,2,3,3,3)?
  - What about (1,2,2, ..., n,n,...,n)?
  - How can we reason about this?

**Helpful formulae**

- We always mean base 2 unless otherwise stated
  - What is log(1024)?
  - $\log(xy) = \log(x) + \log(y)$
  - $n\log(2) = n$
  - $2^{(\log n)} = n$

- Sums (also, use sigma notation when possible)
  - $1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$
  - $1 + 2 + 4 + 8 + ... + 2^k = 2^{k+1} - 1 = \sum_{i=0}^{k} 2^i$
  - $a + ar + ar^2 + ... + ar^{n-1} = a\left(r^n - 1\right) / \left(r-1\right)$
Recursion Review

- Recursive functions have two key attributes
  - There is a base case, sometimes called the exit case, which does not make a recursive call
  - All other cases make recursive call(s), the results of these calls are used to return a value when necessary
    - Ensure that every sequence of calls reaches base case
    - Some measure decreases/moves towards base case
    - “Measure” can be tricky, but usually it’s straightforward

- Example: sequential search in an ArrayList
  - If first element is search key, done and return
  - Otherwise look in the “rest of the list”
  - How can we recurse on “rest of list”?

Sequential search revisited

- What is complexity of sequential search? Of code below?
  ```java
  boolean search(ArrayList<Object> list, int first, Object target) {
    if (first >= list.size()) return false;
    else if (list.get(first).equals(target))
      return true;
    else return search(list, first+1, target);
  }
  ```
- Why are there three parameters? Same name good idea?
  ```java
  boolean search(ArrayList list, Object target){
    return search(list,0,target);
  }
  ```

Why we study recurrences/complexity?

- Tools to analyze algorithms
- Machine-independent measuring methods
- Familiarity with good data structures/algorithms

- What is CS person: programmer, scientist, engineer? 
  *scientists build to learn, engineers learn to build*

- Mathematics is a notation that helps in thinking, discussion, programming

Recurrences

- Summing Numbers
  ```java
  int sum(int n) {
    if (0 == n) return 0;
    else return n + sum(n-1);
  }
  ```
- What is complexity? justification?
- T(n) = time to compute sum for n
  ```latex
  T(n) = T(n-1) + 1 \\
  T(0) = 1
  ```
- instead of 1, use O(1) for constant time
  - independent of n, the measure of problem size
Solving recurrence relations

- plug, simplify, reduce, guess, verify?

\[ T(n) = T(n-1) + 1 \]
\[ T(0) = 1 \]
\[ T(n-1) = T(n-1) + 1 \]
\[ T(n) = [T(n-2) + 1] + 1 = T(n-2) + 2 \]
\[ T(n-2) = T(n-2) + 1 \]
\[ T(n) = [ (T(n-3) + 1) + 1] + 1 = T(n-3) + 3 \]

\[ T(n) = T(n-k) + k \quad \text{find the pattern!} \]

Now, let \( k = n \), then \( T(n) = T(0) + n = 1 + n \)

- get to base case, solve the recurrence: \( O(n) \)

Complexity Practice

- What is complexity of \texttt{Build}? (what does it do?)

```java
ArrayList<Integer> build(int n) {
    if (0 == n) return new ArrayList<Integer>(); // empty
    ArrayList<Integer> list = build(n-1);
    for(int k=0;k < n; k++)
        list.add(n);
    return list;
}
```

- Write an expression for \( T(n) \) and for \( T(0) \), solve.

Recognizing Recurrences

- Solve once, re-use in new contexts
  - \( T \) must be explicitly identified
  - \( n \) must be some measure of size of input/parameter
    - \( T(n) \) is the time for quicksort to run on an \( n \)-element vector

\[ T(n) = T(n/2) + O(1) \quad \text{binary search} \quad O(\log n) \]
\[ T(n) = T(n-1) + O(1) \quad \text{sequential search} \quad O(n) \]
\[ T(n) = 2T(n/2) + O(1) \quad \text{tree traversal} \quad O(n) \]
\[ T(n) = 2T(n/2) + O(n) \quad \text{quicksort} \quad O(n \log n) \]
\[ T(n) = T(n-1) + O(n) \quad \text{selection sort} \quad O(n^2) \]

- Remember the algorithm, re-derive complexity

Eugene (Gene) Myers

- Lead computer scientist/software engineer at Celera Genomics (now at Berkeley, now at ... ?)

  - “What really astounds me is the architecture of life. The system is extremely complex. It's like it was designed.” ... “There's a huge intelligence there.”