12-2 Lecture:

Lecture Notes: Key Topics and Concepts

---

1. Administrivia

- Lab12 is last lab, next week is just for questions
- Final Exam is December 20th from 10-12 in MB AUD
- QOTD24 will be released december 6th, QOTD25 will be released December 9th, and QOTD26(Survey) will be released December 11th
- 2 weeks left and bunch of topics we have not covered that he has in past:

  Regular expressions, GUIs, Client/server systems, Cryptography, Tractability/intractability and computability

- In addition, he often gives one general lecture about Computational Epidemiology, the focus of his research for the past 15 years
- Since we have done enough coding, we will spend a few lectures on tractability/intractability and computability

---

2. XYZ Sequence

- **Problem Description**:
  - Assume you would like to compute a completely made up sequence called xyz(n).
  - This sequence is defined for positive n where xyz(0) = 0 and xyz(1)=1 abd xyz(n)= 3 * xyz(n-1)-xyz(n-2)
  - Q: How would you approach this problem?
  - A: recursively
- **Problem Solution**:

```
def xyz(n):
    if n in {0,1}:
        return(1)
    return(3*xyz(n-1) - xyz(n-2))
```

  - Q: how many times does xyz(n) invoke itself?(runtime)
  - A: It is complicated.Clearly many times. Solving for how many times, will wait till CS:3330
  - But we can get an operational value by implementing an “invocation counting” of xyz(n)

```
def xyz_count(n, c=0):
    c = c+1
    if n in {0,1}:
        return(c,1)
    c,r1 = xyz_count(n-1, c)
    c,r2 = xyz_count(n-2, c)
    return(c,3*r1 - r2)
```

```
>>>xyz_count(1)
(1, 1)
>>>xyz_count(2)
(3, 2)
>>>xyz_count(10)
(177, 4181)
>>>xyz_count(20)
(21891, 63245986)
>>>xyz_count(30)
(2692537, 956722026041)
```

- Not linear! It is exponential, O(2^N)

---

3. Tractable and Intractable Problems

- **Primary Goals**:
- Computer science theory tells us that some problems are **Tractable**, while others are **intractable**, that is, they are difficult to solve outright
- A problem is **intractable** in solving a larger version of the problem that takes exponentially more time **regardless of the algorithm you use.**
- Sorting is an example of a *Tractable* problem; as the number of records you want to sort increases, the cost of sorting grows polynomially (e.g. O(N^2) or NlogN) with the number of records,N.
- The traveling salesperson problem is an example of an intractable program; as the number of cities increases the cost of finding the optimal solution grows exponentially, **regardless of the algorithm used.**
    - Is our xyz problem truly intractable?
        1. .
- **Key Points**:
    - Truly intractable problems have no known better algorithmic solutions
    - In some cases, it has been shown or proven mathematically that no such better solution can exist
    - In other cases, no such proof has been found

---

4. Non deterministic Polynomial Time

- **Definition**:
  - For many of these seemingly intractable problems, the cost of checking a solution for correctness is polynomial and not exponential time(these problems are fundamental in cryptography)
- **More**:
  - These problems are said to be solvable in nondeterministic polynomial time, meaning no better than exponential time solution algorithms, but polynomial time checkable.:
  - Thus the great unsolved problem in theoretical Computer Science: does P=NP?
  - If it is, then we just have not found good solutions for these problems
  - If it isn't, then we have not found a proof that they cannot exist
- **Output**:
  - Returns a set of hard constraint patterns.

---

5. Is XYZ Intractable

- **It is exponential but it is not, and there is a better algorithm**

```
def xyzmemo(n,M={0:1, 1:1}):
    try:
        return(M[n])
    except:
        M[n]=3*xyzmemo(n-1, M) - xyzmemo(n-2,M)
        return(M[n])
```

Dictionary gets bigger and bigger.
