University Software

- Data:
  - Student records
    - Name, address, college, major, GPA, …
  - Accounts
    - Number, description, current balance, …
  - Schedules
    - Course, rooms, exams, …
- Operations:
  - Mailing list creation
  - Tuition billing
  - Finance charges
  - Balance update
  - Room scheduling
  - Exam scheduling

Computational Biology

- Data:
  - Genomes:
    - Sequences of A,C,G,T
  - Protiens:
    - Sequences of base triples
    - Geometric structure
  - Taxonomies:
    - Hierarchies of species
- Operations:
  - Similarity of genomes
  - Determining geometry from base sequences
  - Determining evolutionary hierarchies from genomes

Data Structures

- Schemes for laying out data in memory and on disk, and algorithms for retrieving and updating that data.
- Quality:
  - Space: how much space is needed to store n items.
  - Time: how much time is needed to retrieve/update if there are n items.

Why would you care?

- Fundamental part of many software systems is storage and manipulation of data.
- Many computer science problems are impossible with poor datastructure choice.

CS410 Outline

- Tools:
  - Asymptotic notation, recurence relations, amortisation, ADTs.
- Structures:
  - Stacks, queues, linked lists, trees, heaps, balanced trees, red-black trees, splay trees, random treaps, tries, hashing, quad trees, suffix trees.
- Algorithms:
  - Sorting/searching, graphs, compression.

Running Times

- Database of n items.
- Operations: find, insert, delete.
- Interested in running time of the operations as a function of n.
- $\Theta(\log n)$, $\Theta(n)$, $\Theta(n \log n)$, $\Theta(n^2)$

Recurrence Relations (CLR 4)

- Recursive algorithms are easy to analyse as recurrence relations.
- Factorial:

```
int fact(int n) {
  if (n<=1)
    return 0
  else
    return n*fact(n-1);
}
```

- Running time:
  - $T(n) = c_1$ if $n \leq 1$
  - $T(n) = T(n\text{-}1)+c_2$ otherwise
- Solution: $T(n) \in \Theta(n)$

Fibonacci

```
int fib(int n) {
  if (n<=2) return 1
  else return fib(n-1)+fib(n-2);
}
```

- Recurrence:
  - $T(n) = c_1$ if $n \leq 2$
  - $T(n) = T(n\text{-}1)+T(n\text{-}2)+c_2$ otherwise

Binary Search

```
int bin_search(Object a[], Object key,int low,int high) {
   if (high<=low) return low;
   int mid = (low+high)/2;
   if (a[mid]<=key)
        return bin_search(a,key,mid,high);
   else
        return bin_search(a,key,low,mid);
}
```

- Recurrence:
  - $T(n) = c_1$ if $n \leq 0$ $T(n)=T(n/2)+c_2$ otherwise

Recurrence Relations

- A function $T(n)$ defined in terms of T for smaller values of n.
- Solution to a recurence relation is a definition of $T(n)$ not defined in terms of T.
- Method turns running time analysis into a clean mathematical problem.

Solution Techniques

- Guess and verify (Substitution)
- Iteration
- Master Theorem

Guess and Verify

- Guess a solution and then prove that its correct by induction on n.
- Example: factorial is $O(n)$ guess that $T(n) \leq cn$ for some c.
- Base case (n=1): $T(n) = c_1 \leq cn$ if $c_1 \leq c$
- Inductive case ($n=k+1$): $T(k+1)=T(k)+c_2 \leq ck+c_2 \leq c(k+1)$ if $c_2 \leq c$

Fibonacci

- Fibonacci is O(2^n), guess $T(n) \le c2^n$
- Base case (n=0): $T(n) = c_1 \le c2^n$ if $c_1 \le c$
- Base case (n=1): $T(n) = c_1 \le c2^n$ if $c_1 \le 2c$
- Base case (n=2): $T(n) = c_1 \le c2^n$ if $c_1 \le 4c$
- Inductive case ($n=k+2, k \ge 1$):
  $T(k+2)=T(k+1)+T(k)+c_2 \le c2^{k+1}+c2^k+c_2$
  $\le c(2^{k+1}+2^k+1)$ if $c_2 \le c$
  $\le c2^{k+2}$

Binary Search

- Binary search is $\Omega(\log n)$ guess $c \log n \le T(n)$
- Base case ($n=1$): $c \log n = 0 \le c_1 + c_2 = T(n)$
- Inductive case ($n>1$):
  $T(n)=T(n/2)+c_2 \ge c \log n/2 + c_2 =$
  $(c \log n) - c + c_2 \ge c \log n$ if $c_2 \le c$

Change of Variable

- $T(n)=2T(n^{1/2})+\log n$
- Rename $m=\log n$
- $T(2^m)=2T(2^{m/2})+m$
- $S(m)=T(2^m)$, $S(m)=2S(m/2)+m$
- $S(m) \in O(m \log m)$
- $T(n) \in O(\log n \log \log n)$

Iteration

- $S(m)=m+2S(m/2)$
  $=m+2m/2+4S(m/4)$
  $=m+2m/2+4m/4+8S(m/8)$
  $=m+2m/2+\ldots+mS(m/m)$
  $=m \log_2 m$
- Two differculties:
  - How many terms in sum
  - What does each sum look like

More Iteration

- Consider binary search again:
- $T(n)=c_2+T(n/2)$
  $=c_2+c_2+T(n/4)$
  $=c_2+c_2+\ldots+c_2+T(n/n)$
  $=c_2+c_2+\ldots+c_2+c_1$
  $=c_2 \log n + c_1$

Master Theorem

- If $a \ge 1$ and $b > 1$ are constants, $f(n)$ a function, and $T(n)=aT(n/b)+f(n)$ then $T(n)$ has the following solution:
  - If $f(n) \in O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$ then $T(n) \in \Theta(n^{\log_b a})$
  - If $f(n) \in \Theta(n^{\log_b a})$ then $T(n) \in \Theta(n^{\log_b a} \log n)$
  - If $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for some $c<1$ and all sufficiently large n then $T(n) \in \Theta(f(n))$

Master Theorem Intuition

- Two parts $aT(n/b)$ and $f(n)$, the three cases correspond to which part is growing faster.
- $aT(n/b)$ grows as $n^{\log_b a}$
- case 1 is where $aT(n/b)$ grows faster
- case 2 is where they are same, this adds a log n factor
- case 3 is where $f(n)$ grows faster

Binary Search

- Recurrence:
  - $T(n) = c_1$ if $n \leq 0$ $\quad T(n)=T(n/2)+c_2$ otherwise
- $a=1$, $b=2$, $f(n)=c_2$
- $n^{\log_b a}=n^0=1$
- $f(n) \in \Theta(n^{\log_b a})$
- case 2
- $T(n) \in \Theta(n^{\log_b a} \log n) = \Theta(\log n)$

Merge Sort

- Divide array into two, sort each part, merge results.
- Merge takes $\Theta(n)$ running time.
- $T(n)=2T(n/2)+\Theta(n)$
- $a=2$, $b=2$, $f(n) \in \Theta(n)$
- $n^{\log_b a}=n^1=n$
- case 2, $T(n) \in \Theta(n^{\log_b a} \log n) = \Theta(n \log n)$

Case 1

- $T(n)=8T(n/2)+n$
- $a=8$, $b=2$, $f(n)=n$
- $n^{\log_b a}=n^3$
- $f(n) \in O(n^{\log_b a-\varepsilon})$ for $\varepsilon=2$
- $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^3)$

Case 3

- $T(n)=4T(n/4)+n^2$
- $a=4$, $b=4$, $f(n)=n^2$
- $n^{\log_b a}=n$
- $f(n) \in \Omega(n^{\log_b a+\varepsilon})$ for $\varepsilon=1$
- Regularity: $af(n/b)=an^2/b^2=a/b^2 f(n)$
- $T(n) \in \Theta(f(n)) = \Theta(n^2)$