Divide and Conquer

Week 8

Required Activities

- Check Announcements regularly (every 2-3 days)
- Read Foundations of Algorithms (FA) book: Chapters 1 and 2
- Complete and submit **Assignment 3 due 12/21 (TOMORROW)**
- **Optional Extra Credit due 1/6 NO EXTENSIONS**
- **Next two weeks BREAK 12/22-1/2 (no class 12/27 & 1/3)**

Analysis of Algorithms

- measures the efficiency of an algorithm as the input size becomes large for the critical resources which are running time (or memory) and space
- **time complexity analysis** - how many times the basic operation is done for each value of the input size
- **memory complexity** is an analysis of how efficient the algorithm is in terms of memory
- **Time analysis** should consider:
  - Basic operations
  - Overhead instructions – for example initialization before a loop (does not increase with input size
  - Control instructions – for example incrementing loop counter (does increase with input size)
- **Analysis of correctness** - developing a proof that the algorithm actually does what it is supposed to do

Divide-and-Conquer (D&C)

- **Design algorithm** technique where problem instance is divided into smaller instances and then smaller instances until small enough where solution can be obtained. Then the solutions are combined to solve the original problem instance.
- Top-down approach
- Uses recursion
- **Steps:**
  - **Divide** problem into smaller instances
  - **Conquer** (solve) smaller instance or divide further (recursion)
  - **Combine** to obtain the solution for the original problem

Examples of efficient algorithms

- **Binary search** – divides data into two sub-lists, keeps searching in sub-list dividing each iteration, until find the item
- **Mergesort** – keeps dividing list to one unit, then merges in sorted order
- **Quicksort –** splits list based on pivot and keeps splitting and sorting one sub-list and then the other

Pros & Cons

- Powerful method for solving difficult problems (break down complex into simpler)
- Allows solving sub-problem independently so can be executed on different processors
- Recursion solution can be slow because of the overhead of repeated sub-problem calls
- For some problems the solution may be more complex than iterative approach (e.g. using simple loop)

Fibonacci Sequence

f0 = 0; f1 = 1; fn = f(n -1) + f(n-2) for n >= 2

Ex: f3 = f2+ f1 = f1 + f0 + f1= 1 + 0 + 1 = 2

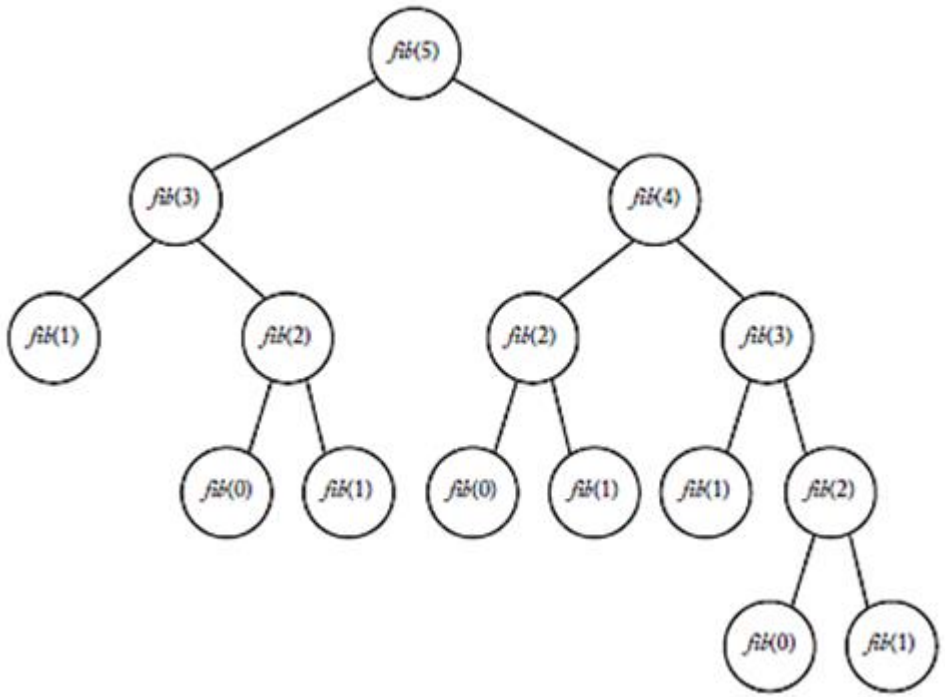
Recursive algorithm (divide and conquer)

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

fib(3): else: return fib(2)+ fib(1)
fib(2): else: return fib(1) + fib(0)
fib(1): return 1
fib(0): return 0
fib(1): return 1
1+0+1 = 2

- Extremely inefficient because computes same values over and over the greater the nth value
- Theorem 1.1 page 15: this algorithm calculates **$2^{n/2}$ terms**

Recursion Tree for the 5th Fibonacci Term



Fibonacci Improved

-> do not compute the same values repeatedly

Iterative algorithm (dynamic programming approach)

```
int fib2 (int n) {
   index i;
   f[0]=0
   int f[0..n];
   if (n > 0) {
       f[1]=1;
       for (i=2; i<=n; i++)
       f[i] = f[i-1] + f[i-2);
   }
   return f[n];
}
```

fib2(3): f[0] = 0
if => f[1] = 1
for: i=2; i<=n; f[2] = f[1] + f[0] => f[2] = 1 + 0 = 1; i++
i=3; i <=n; f[3] = f[2] + f[1] => f[3] = 1 + 1 = 2; i++
i=4; i <=n
return f[3] = 2

- Calculates each term only once and saves in the array so this algorithm calculates **n+1 terms**

Comparison of Recursive and Iterative Solutions

n	$n+1$	$2^{n/2}$	Execution Time Using Algorithm 1.7	Lower Bound on Execution Time Using Algorithm 1.6
40	41	1,048,576	41 ns*	1048 $\mu s^{\dagger}$
60	61	1.1×10^{9}	61 ns	1 s
80	81	1.1×10^{12}	81 ns	18 min
100	101	1.1×10^{15}	101 ns	13 days
120	121	1.2×10^{18}	121 ns	36 years
160	161	1.2×10^{24}	161 ns	3.8×10^{7} years
200	201	1.3×10^{30}	201 ns	4×10^{13} years

*1 ns $= 10^{-9}$ second.

$^{\dagger}1\ \mu s = 10^{-6}$ second.

Time Analysis of D&C

- Define problem in terms of recursive definition
- Solve the problems using Master theorem:
  - **For Subtract and Conquer recurrences:**

$$T(n) \leq \begin{cases} c, & \text{if } n \leq 1, \\ aT(n-b) + f(n), & n > 1, \end{cases},$$

for some constants c, a>0, b>0, k>=0 and function f(n). If f(n) is $O(n^k)$, then

1. If a<1 then $T(n) = O(n^k)$
2. If a=1 then $T(n) = O(n^{k+1})$
3. if a>1 then $T(n) = O(n^k a^{n/b})$

  - **For Divide and Conquer recurrences:**

$$T(n) = a\,T(n/b) + \Theta(n^k (\log n)^i).$$

1. If $f(n) = O(n^c)$ where $c < \text{Log}_b a$ then $T(n) = O(n^{\text{Log}_b a})$
2. If $f(n) = O(n^c)$ where $c = \text{Log}_b a$ then $T(n) = O(n^c \text{Log } n)$
3. If $f(n) = O(n^c)$ where $c > \text{Log}_b a$ then $T(n) = O(f(n))$

- Or solve using induction (guess at solution and substitute)

Example 1

Algorithm A which solves problem by dividing into 5 subprograms where each takes n-1 to solve recursively, and combines in linear time:

- 5 subprograms each takes n-1 to solve
- Time to solve 5* t(n-1)
- Linear time to combine means O(n)
- So recurrence T(n) = 5T(n-1)+O(n)
- So the recurrence is of format for Master theorem for subtract and conquer
  - $T(n) = aT(n-b) + f(n^k)$
  - a=5, b=1, k=1
  - $a > 1$ so use $T(n) = O(n^k a^{n/b})$ and plug in values
  - Solution is $O(n^1 5^{n/1}) = O(n5^n)$

Example 2

Algorithm B which solves problem by dividing into 5 subprograms of half size, recursively solves, and combines in linear time:

- 5 subprograms each of size n/2
- Time to solve 5* t(n/2)
- Linear time to combine means O(n)
- So recurrence is t(n) = 5t(n/2)+O(n)
- From recurrence we have a=5, b=2, k=1
- Using Master theorem for divide and conquer: $O(n\log_2{}^5) = O(n^{2.322}) = O(n^3)$

Master Theorem tool

- https://www.nayuki.io/page/master-theorem-solver-javascript
- Input: t(n) = 5t(n/2)+O(n)

Format: $T(n) = a\,T(n/b) + \Theta(n^k (\log n)^i)$.

a: 5
b: 2
k: 1
i: 0 (usually 0)

Calculate Clear

Output

Recurrence: $T(n) = 5\,T(n/2) + \Theta(n)$.

Solution: $T(n) \in \Theta(n^{\log_2 5}) \approx \Theta(n^{2.322})$.

- Textbook has induction answers : Appendix B pp. 603-642

More Examples

- Let's say we have problem where we determine the recurrence to be: T(n) = 3T(n-1) where n>0 so our constants are: a=3, b=1, f(n)=0 so k=0

That is the format for **master theorem for Subtract and Conquer recurrences**: **aT(n-b) + f(n) where f(n) = O(n^k)**

So we start substituting:

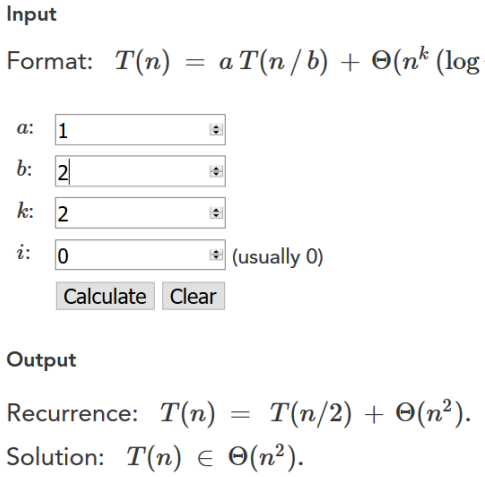
Using a=3, b=1, k=0 we determine that since a>1 then we use third rule: **T(n) = O($n^k a^{n/b}$)**

So substituting constants we have: t(n) = O($n^0 3^{n/1}$) = **O(3^n)**

- Let's say we have problem where we determine the recurrence to be: T(n) = T(n/2) + O(n^2) so our constants are: a=1, b=2, k=2

That is the format for **master theorem for Divide and Conquer recurrences**: **aT(n/b) + f(n) where f(n) = O(n^k)**

So using the tool we plug in values:



Algorithms revisited

- **Binary search** – worst case when item being compared is larger than all items in array **O(log n)**
  - t(n/2) number of comparisons in the recursive call (divide in half)
  - 1 comparison at top level
  - recurrence: **t(n)=t(n/2)+1** for n>1, n power of 2, t(1)=1 and using induction or master theorem: **O(log n)**

- **Mergesort** – worst case when every element of array will be compared at least once **O(n log n)**
  - Work is in merging two arrays: h=numOfItems in A1 and m=numOfItems in A2
  - h = time to sort; m = time to sort; (h + m - 1) = time to merge
  - h = (n/2) and m = (n/2) (divide in half)
  - h + m = n/2 + n/2 = n
  - t(n) = t(n/2) + t(n/2) + n – 1 = 2t(n/2) + n – 1
  - recurrence: **t(n) = 2t(n/2) + n – 1** for n>1, n power of 2, t(1) = 0 and using induction or master theorem: **O(n log n)**

- **Quicksort –** worst case when already sorted **O(n^2)** but remember average case is **O(n log n)**
  - Time to sort left sub-array t(0); time to sort right subarray t(n-1); time to partition (n-1)
  - t(n) = t(n-1) + n-1 for n > 0, t(0) = 0
  - Substituting yields recurrence: **t(n) = n(n-1)/2** and using induction or master theorem: **O(n^2)**

Questions ?

- Post in the discussions
- Send email to RMcFadden@HarrisburgU.edu
- Respond usually within 48hours