Algorithm Design & Analysis IV

Recursive Algorithms

Fibonacci

Mergesort
binary search
Hanoi Tower
tree algorithms

- A recursive algorithm consists of one or more base cases and one or more calls to itself on a smaller set of data than the original input.
- Example: Binary Search
  - Base Case: If the size of the search array is 1, then you check that one element.
  - Recursive Step: If not, you check the median and if you find what you're looking for, you're done. Otherwise, you do the process again (recursive call) on one half of the array.

Analyzing Recursive Algorithms

$T(N)$ is $O(\log N)$

- **The runtime of a call to a recursive algorithm on an input size of N is going to equal**
  - **the runtime of** the recursive calls **+**
  - **the runtime of** any additional work
- We can model this with a recursive function called a recurrence relation.
- For example, binary search…
  - Let T(N) be the runtime of the algorithm on an array of size N.
  - In the worst case, we would have to do 1 recursive call(s) on N/2 of the data, which would have a runtime of $T(N/2)$.
  - Additionally, we also do $O(1)$, 1 amount of "other work."
  - So the total runtime would be $T(N) = T(N/2) + 1$ with a base case of $T(1) = 1$.

What does each part of this mean?

$$T(N) = \underset{\text{\# of recursive calls}}{a}\underbrace{T(\overbrace{f(N)}^{\text{size of recursive call}})}_{\text{recursive call(s)}} + \underbrace{g(N)}_{\text{extra work}};\ T(1) = \overbrace{h(n)}^{\text{base case work}}$$

Solving Recurrence Relations

- expansion & summation
- Master Theorem
- tree

Example. Determine what this function does. Then write a recurrence relation for the runtime in terms of N. Then determine what the big-Theta runtime is.

$\lfloor \log_2 n \rfloor$

```
function foo(int n)
    if n <= 1 return 0
    return 1 + foo(floor(n/2))
end foo
```

$T(N) = T(\lfloor N/2 \rfloor) + 1$

$T(1) = 1$

$\Theta(\log N)$

Example. Write a recurrence relation for the runtime in terms of N. Then determine what the big-Theta runtime is.

```
Algorithm X
A = an array of size N
doSomething(A, 0, N-1)


procedure doSomething(Array A, int i, int j)
    if (j - i <= 1) return
    m = (i+j)/2
    doSomething(A, i, m-1)
    doSomething(A, m, j)
    foo(A, i, j)//an O(N) operation where N is j-i+1
end doSomething
```

$T(N) = 2T(N/2) + N$

$T(1) = 1$

$\Theta(N \log N)$

Example. Write and analyze a recursive algorithm for calculating the n^{th} power of a constant b.

$O(N)$

```
power(int b, int n)
    if n=0 then 1
    b * power(b, n-1)
end
```

$T(n) = \underline{T(n-1) + 1}$

$T(1) = 1$

$\sum_{k=1}^{n} 1 = n \quad O(n)$

$O(\log N)$

```
power(int b, int n)
    if n=0 then 1
    if n=1 then b
    p = power(b, ⌊n/2⌋)
    if n is even then p*p
    else b*p*p
end
```

$T(n) = T(n/2) + 1 \quad \Theta(\log N)$

$T(1) = 1$

Example. Write and analyze a recursive algorithm for calculating the n^{th} fibonacci number.

$T(n) = T(n-1) + T(n-2) + 1$

$T(1) = 1$

```
fib(int n):
    if n = 0 then 0
    if n = 1 then 1
    fib(n-1) + fib(n-2)
end
```

$O(2^N)$ $\quad O(N)$