Lecture 3: Recursion
CSCI 700 - Algorithms I

Andrew Rosenberg
Last Time

- Asymptotic Notation
- Review of Proofs by Induction
Today

- Recursion
  - Binary Search
  - MergeSort
  - QuickSort
A recursive function calls itself.
Recursion

- A **recursive** function calls itself.
- Requirements to avoid infinite loops
  - A stopping condition
  - Modification of input
Fibonacci Series

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

- Related to the Golden Ratio $\phi$
Formula for Calculating Fibonacci numbers

- $Fib(1) = 1$
- $Fib(2) = 1$
- $Fib(n) = Fib(n - 1) + Fib(n - 2)$ for $n > 2$
- $1, 1, 2, 3, 5, 8, 12, 20, 32, 52, \ldots$
Fib(n) in pseudocode

Fib(n)

1: if n = 1 then
2: return 1
3: end if
4: if n = 2 then
5: return 1
6: end if
7: return Fib(n-1) + Fib(n-2)
Runtime of $Fib(n)$

- How many times is $Fib(n)$ called?
Fib(3)

Fib(2)  Fib(1)

1 1 2
Fib(5)

\[
\begin{array}{c}
\text{Fib(5)} \\
\downarrow \\
\text{Fib(4)} \\
\downarrow \\
\text{Fib(3)} \\
\downarrow \\
\text{Fib(2)} & \text{Fib(1)} \\
\downarrow & \downarrow \\
1 & 1 \\
\end{array}
\]
Fib(n)

Fib(n)
  /     \
Fib(n-1)  Fib(n-2)
  /     \
Fib(n-2)  Fib(n-3)  Fib(n-4)
  /     \
Fib(n-3)  Fib(n-4)
  /     \

1 = 2^0
2 = 2^1
4 = 2^2
8 = 2^3
\[ \text{Fib}(n) = O(2^n) \]

We will return to \( \text{Fib}(n) \) when we talk about Dynamic Programming.
Fib(n)

- $Fib(n) = O(2^n)$
  - To think about. $Fib(n) = \Theta(2^{\frac{n}{2}})$

- We will return to $Fib(n)$ when we talk about Dynamic Programming.
Find the index $i$ of an item $x$ in a sorted array $A$ of size $n$.

```c
int Find(A, x)
1: ... 
2: return i
```
### Sequential Search

```plaintext
int Find(A, x)

1:   for i ← 0..N-1 do
2:       if A[i] = x then
3:           return i
4:       end if
5:   end for
6:   return Not Found
```
Runtime of Sequential Search

- The for loop (line 1-5) runs N times.
- Best Case $A[0] = x \Theta(1)$
- Worst Case $A[n] = x$ or $x$ is not in $A \Theta(n)$
- Expected Case if $x$ is in $A$, $\Theta(\frac{n}{2}) = \Theta(n)$, if not, $\Theta(n)$

```c
int Find(A, x)
    for i ← 0..N-1 do
        if A[i] = x then
            return i
        end if
    end for
    return Not Found
```
A = [10, 13, 14, 29, 37]

Find 29 in A

Check 14. \( A_1 = [10, 13] \) \( A_2 = [29, 37] \)

Find 29 in \( A_2 \)

Check 29. Success.
**Binary Search**

```plaintext
int Find(A, x)

1: return BinarySearch(A, x, 1, N+1)

int BinarySearch(A, x, low, high)

1: if low > high then
2: return Not Found
3: end if
4: mid ← \left\lfloor \frac{low+high}{2} \right\rfloor
5: if x = A[mid] then
6: return mid
7: end if
8: if x < [mid] then
9: return BinarySearch(A, x, low, mid-1)
10: else
11: return BinarySearch(A, x, mid+1, high)
12: end if
```
Logarithmic and Exponential Growth

- Note: These are crude rules of thumb.
- Multiplication at each recursive step leads to exponential growth
- Division at each recursive step leads to logarithmic growth
Each recursive call to BinarySearch cuts the examined size of $A$ in half.

Therefore suspect that BinarySearch($A$, $x$, low, high) is $\Theta(\log n)$

**Best Case** $A[mid] = x$ $\Theta(1)$

**Worst Case** $x$ is not in $A$ $\Theta(\log n)$

**Expected Case** if $x$ is in $A$, $\Theta(\log n - 1) = \Theta(\log n)$. $x$ is not in $A$ $\Theta(\log n)$
Binary vs. Sequential Search

- Binary Search - $\Theta(\log n)$
- Sequential Search - $\Theta(n)$
- ...but Binary search requires A to be sorted.
- The best sorting we’ve looked at is $\Theta(n^2)$. Can we do better?
Divide and Conquer

- A recursive strategy.
- **Divide** the problem into subproblems
- **Conquer** the subproblems recursively
- **Combine** the subproblem solutions into a solution of the initial problem.
To construct a sorted list from two sorted lists is $\Theta(n)$.

- $A_1 = [1, 4, 7, 9, 11]$
- $A_2 = [2, 3, 8, 10]$
- $A = []$
Merge Sort

- To construct a sorted list from two sorted lists is $\Theta(n)$.
- $A_1 = [4, 7, 9, 11]$
- $A_2 = [2, 3, 8, 10]$
- $A = [1]$
To construct a sorted list from two sorted lists is $\Theta(n)$.

- $A_1 = [4, 7, 9, 11]$
- $A_2 = [3, 8, 10]$
- $A = [1, 2]$
To construct a sorted list from two sorted lists is $\Theta(n)$.

- $A_1 = [4, 7, 9, 11]
- A_2 = [8, 10]
- A = [1, 2, 3]
To construct a sorted list from two sorted lists is $\Theta(n)$.

- $A_1 = [7, 9, 11]$
- $A_2 = [8, 10]$
- $A = [1, 2, 3, 4]$
Merge Sort

To construct a sorted list from two sorted lists is $\Theta(n)$.

- $A_1 = [9, 11]$
- $A_2 = [8, 10]$
- $A = [1, 2, 3, 4, 7]$
To construct a sorted list from two sorted lists is $\Theta(n)$.

- $A_1 = [9, 11]$
- $A_2 = [10]$
- $A = [1, 2, 3, 4, 7, 8]$
To construct a sorted list from two sorted lists is $\Theta(n)$.

- $A_1 = [11]$
- $A_2 = [10]$
- $A = [1, 2, 3, 4, 7, 8, 9]$
Merge Sort

• To construct a sorted list from two sorted lists is $\Theta(n)$.
• $A_1 = [11]$
• $A_2 = []$
• $A = [1, 2, 3, 4, 7, 8, 9, 10]$
To construct a sorted list from two sorted lists is $\Theta(n)$. 

- $A_1 = []$
- $A_2 = []$
- $A = [1, 2, 3, 4, 7, 8, 9, 10, 11]$
Apply Divide and Conquer to Merge Sort

Divide the $n$ element array into two subarrays with $\frac{n}{2}$ elements each to be sorted.

Conquer by sorting the subsequences using Merge Sort

Combine the two subsequences (as above) to construct the sorted array
MergeSort(A)

\[
\text{if } \text{size}(A) = 1 \text{ then} \\
\quad \text{return } A \\
\text{end if} \\
\text{mid } \leftarrow \lfloor \frac{\text{size}(A)}{2} \rfloor \\
A_1 = \text{MergeSort}(A[1..\text{mid}]) \\
A_2 = \text{MergeSort}(A[\text{mid}+1..N]) \\
\text{return } \text{Merge}(A_1, A_2)
\]
Merge Sort Example

- Divide
- A = [10, 2, 6, 4]
Merge Sort Example

- Divide
- \( A = [10, 2, 6, 4] \)
- \( A = [10, 2][6, 4] \)
Merge Sort Example

- **Divide**
  - $A = [10, 2, 6, 4]$
  - $A = [10, 2][6, 4]$
  - $A = [10][2][6][4]$
Merge Sort Example

- **Conquer** (or Merge)
- \( A = [10, 2, 6, 4] \)
- \( A = [2, 10][4, 6] \)
- \( A = [10][2][6][4] \)
Merge Sort Example

- Conquer (or Merge)
- $A = [2, 4, 6, 10]$
- $A = [2, 10][4, 6]$
- $A = [10][2][6][4]$
How long does MergeSort take? $\Theta(\cdot)$
Runtime analysis of MergeSort

- How long does MergeSort take? $\Theta(?)$
- How many levels of recursion are there?
Runtime analysis of MergeSort

- How long does MergeSort take? $\Theta(?)$
- How many levels of recursion are there?
- $\log n$
Runtime analysis of MergeSort

- How long does MergeSort take? $\Theta(?)$
- How many levels of recursion are there?
- $\log n$
- How many operations are performed at each level of the recursion?
Runtime analysis of MergeSort

- How long does MergeSort take? $\Theta(\cdot)$
- How many levels of recursion are there?
  - $\log n$
- How many operations are performed at each level of the recursion?
  - $n$
Runtime analysis of MergeSort

- How long does MergeSort take? $\Theta(\cdot)$
- How many levels of recursion are there? $\log n$
- How many operations are performed at each level of the recursion? $n$
- Merge Sort is $\Theta(n \log n)$
Quick Sort

- Another recursive sorting Algorithm
- Concatenating arrays is $\Theta(1)$
- Identifying those points in an array that are higher than a point is $\Theta(n)$
Quick Sort

- Identify a point $A[i]$ from the $A$ called the “pivot”
- Construct lists $A_1$ and $A_2$ such that $A_1$ contains all the elements of $A$ less than $A[i]$, and $A_2$ those greater than $A[i]$
- Recursively sort $A_1$ and $A_2$
- Concatenate $A_1$, $A[i]$ and $A_2$
Quick Sort Example

- Divide
- \( A = [10, 2, 6, 4] \)
Quick Sort Example

- Divide
- $A = [10, 2, 6, 4]$
- Select pivot 6
Quick Sort Example

- Divide
- $A = [10, 2, 6, 4]$
- Select pivot 6
- $A = [2, 4][6][10]$
Quick Sort Example

- Divide
- $A = [10, 2, 6, 4]$  
- Select pivot 6
- $A = [2, 4][6][10]$
- $A_1 = [2,4]$
Quick Sort Example

- Divide
- $A = [10, 2, 6, 4]$
- Select pivot 6
- $A = [2, 4][6][10]$
- $A_1 = [2, 4]$
- Select pivot 4
Quick Sort Example

- **Divide**
- $A = [10, 2, 6, 4]$
- Select pivot 6
- $A = [2, 4][6][10]$
- $A_1 = [2, 4]$
- Select pivot 4
- $A = [2][4]$
Quick Sort Example

- Combine
- \( A_1 = [2][4] \) return \([2, 4]\)
Quick Sort Example

- Combine
  - $A = [2, 4][6][10]$ return $[2, 4, 6, 10]$
How long does QuickSort take? $\Theta(\cdot)$
Runtime analysis of QuickSort

- How long does QuickSort take? $\Theta(\cdot)$
- How many levels of recursion are there?
Runtime analysis of QuickSort

- How long does QuickSort take? $\Theta (?)$
- How many levels of recursion are there?
- $\log n$
Runtime analysis of QuickSort

- How long does QuickSort take? $\Theta(?)$
- How many levels of recursion are there? $\log n$
- How many operations are performed at each level of the recursion?
Runtime analysis of QuickSort

- How long does QuickSort take? $\Theta(\cdot)$
- How many levels of recursion are there?
  - $\log n$
- How many operations are performed at each level of the recursion?
  - $n$
Runtime analysis of QuickSort

- How long does QuickSort take? $\Theta(\cdot)$
- How many levels of recursion are there? $\log n$
- How many operations are performed at each level of the recursion? $n$
- Quick Sort is $\Theta(n \log n)$
<table>
<thead>
<tr>
<th></th>
<th>MergeSort</th>
<th>QuickSort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Conquer</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>Combine</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>
Expected vs. Worst Case runtime

- Quicksort has an expected runtime of $\Theta(n \log n)$
- QuickSort could be as slow as $\Theta(n^2)$
Expected vs. Worst Case runtime

- Quicksort has an expected runtime of $\Theta(n \log n)$
- QuickSort could be as slow as $\Theta(n^2)$
- How?
Bye

Next time (9/16)
- Analyzing the runtime of general recursive algorithms.
- Read Chapter 8.