Week 1

Wednesday, May 17, 2017
6:18 PM

No Tutorial Tomorrow

Learning Goals

- Design - Problem Solving
- Correct - Proofs
- "Efficient" Algorithm - "Better"
    - Faster - Better time
    - Less resources - Better than the current solution

Given: A sorted array A, and an element x
Goal: Find whether x is in A
Solution 1:

```
for i=1 ... n
    if A[i] == x
        return True
return False
```

Different Proof Techniques

1. Proof by contradiction
2.

a. Proof by mathematical induction
   $P(n)$: Prove something for every n
   Base Case: $P(0)$ (could be other numbers)
   Inductive Hypothesis: Assume the result holds for n
   Inductive Step: Prove the result holds for $n+1$
b. Strong induction
   Inductive Hypothesis: Assume this result holds for all $k = 0, \dots, n$ (up to n)
c. Structural Induction

3. Proof by Elimination and Cases
4. Proof by contrapositive ($\neg q \Rightarrow \neg p$)
5. Direct Proof ($p \Rightarrow q$)
6. Proof by image/picture
7. Trivial Proof/Obvious
8. Proof by equivalence
   ($p \equiv q \equiv r$)
   $p \Rightarrow q, q \Rightarrow r, r \Rightarrow p$

Binary Search:

$$A = [a_1,\ a_2,\ \dots,\ a_n]$$
$$a_1 \leq a_2 \leq a_3 \dots \leq a_{\frac{n}{2}} < \cdots < a_n$$

If not: 1. $a_{\lfloor a \rfloor} < x$
2. 1. $a_{\lfloor a \rfloor} > x$

Solution 2:

```
find(A, left, right, x) // like find(A, 0, n-1, x)
    if left > right
        return False
    mid = (left + right)/2
    if a[mid] == x
        return True
    if a[mid] < x
        return find(A, mid+1, right, x)
```

Time: $O(\log n)$

- Better than brute force
- Polynomial in the number of inputs

Sorting: (by comparison)
Lower bound: $\Omega(n\log_2 n)$
(numbers are distributed in a small range)
Counting/Bucket Sort - $O(kn)$
(number are distributed in a small range)

E.g.
1 billion people
Age - [0, 200]
Sort by age

Buckets

$O(n)$

Complexity : $c, \lg n, n, n \lg n, n^2, n^3, n^4, 2^n, e^n, 3^n$
--->
polynomial exponential

1st Technique: Divide-and-conquer

1. Binary Search
   Divide: Break A into $A\left[0, \frac{n}{2}\right]$ or $A\left[\frac{n}{2}, n\right]$
   Conquer: $O(1)$ (if $x == A\left[\frac{n}{2}\right]$)
   $$
   \begin{aligned}
   T(n) &= T\left(\frac{n}{2}\right) + O(1) \\
   &= T\left(\frac{n}{4}\right) + O(1) + O(1) \\
   &= T\left(\frac{n}{8}\right) + O(1) + O(1) + O(1) \\
   &\vdots \\
   &= O(1) + \log_2 n
   \end{aligned}
   $$

Merge-sort
Divide: $A[n] \rightarrow A\left[0, \frac{n}{2}\right]$ $A\left[\frac{n}{2}, n\right]$
Conquer: merge Initialize $C[]$

```
MergeSort(A, left, right)
    if (left < right)
        mid = (left + right) / 2
    mergeSort(A, left, mid)
    mergeSort(A, right, mid)
    merge(A, left, right, mid)

merge(A, left, right, mid)
    initialize C
    index = 0, rIndex = mid+1
    while (left < mid and rIndex < rig
        if A[left] < A[rIndex]
            C[index+1] = A[left+1]
        else
            C[index+1] = A[rIndex+1]
    if (lIndex < mid)
        copy rest of ALeft to C
    return C
```

$$T(n) = \underset{\substack{\downarrow \\ \text{\# of sub-problems}}}{2}T\left(\underset{\substack{\downarrow \\ \text{size of sub-problems}}}{\frac{n}{2}}\right) + O(n)$$

$$
\begin{aligned}
T(n) &= 2\left[2T\left(\frac{n}{4}\right) + O\left(\frac{n}{2}\right)\right] + O(n) \\
&= 2\left[2\left[2T\left(\frac{n}{8}\right) + O\left(\frac{n}{4}\right)\right] + O\left(\frac{n}{2}\right)\right] + O(n) \\
&= 2^3T\left(\frac{n}{2^3}\right) + 2^2O\left(\frac{n}{2^2}\right) + 2O\left(\frac{n}{2}\right) + O(n) \\
&\vdots \\
&= 2^k\underbrace{T\left(\frac{n}{2^k}\right)}_{\substack{= T(1) \\ = O(1)}} + 2^{k-1}O\left(\frac{n}{2^{k-1}}\right) + \cdots + O(n)
\end{aligned}
$$

n

/ \

$\frac{n}{2}$ $\frac{n}{2}$

/ \ / \

$\frac{n}{4}$ $\frac{n}{4}$ $\frac{n}{4}$ $\frac{n}{4}$

Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Case I: $f(n) = O\left(n^{\log_b a-\epsilon}\right)$ $T(n) = \theta\left(n^{\log_b a}\right)$

Case II: $f(n) = O\left(n^{\log_b a}\right)$ $T(n) = \theta\left(n^{\log_b a} \lg n\right)$

Case III: $f(n) = O(n^{\log_b a+\epsilon})$

& additional restriction $T(n) = \theta\big(f(n)\big)$

2. Merge Sort

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$a = 2, b = 2, f(n) = O(n)$

$n^{\log_b a} = n^{\log_2 2} = n = O\big(f(n)\big)$

By case II, $T(n) = \theta\left(n^{\log_a b} \log_2 n\right) = \theta(n \log_2 n)$

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n)$$

$a = 4, b = 2, f(n) = O(n)$

$n^{\log_b a} = n^{\log_2 2^2} = n^2 > O\big(f(n)\big)$ $(\epsilon = 1)$

By Case I, $T(n) = O(n^2)$

3. Computing power

```
Power(a, n)  -> return a^n
//Brute Force
  for i = 1 to n
        t = t * a  -> O(1)
  return t

Recursive_Power(a,n)
  if (n == 1)
        return a
  t = Recursive_Power(a, n/2)
  if n is even:
        t = Recursive_Power(a, n/2)
        return t
  return t * t * a
```

$T(n) = T\left(\frac{n}{2}\right) + O(1)$

$T(n) = O(\log_2 n)$

4. Multiply two integers

```
Product(a, b) -> return a*b
  t = 0
  for i = 1 : a
        t = t + b
  return b
```

$O(b)$

$a = 11011001$

$b = 100101110$

$a + b = 1011011111$
$n = \max(\log_2 n, \log_2 b)$

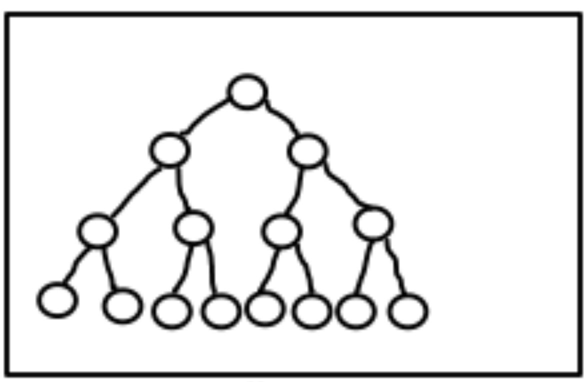
RecursiveProduct(a, b)

$$\text{Let } a = a_1 \cdot 2^{\frac{n}{2}} + a_0$$
$$b = b_1 \cdot 2^{\frac{n}{2}} + b_0$$
$$ab = a_1 b_1 \cdot 2^n + (a_0 b_1 + a_1 b_0) \cdot 2^{\frac{n}{2}} + a_0 b_0$$

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n)$$
Case I: $T(n) = \theta\left(n^{\log_2 4}\right)$
$= \theta(n^2)$
$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$
$$T(n) = \theta\left(n^{\log_2 3}\right) \approx \theta(n^{1.59})$$

5. Embedding a complete binary tree in a circuit

$\log_2 n$



Total area $O(n log_2 n)$

$n \quad nlglgn \quad n^2$

$2 \log_2 n = n$

$\left\lceil \frac{n}{2} \right\rceil$
$O(n)$

$$L\times W = A(n) =$$

$$A(n) = \theta(n)$$

$$\sqrt{n}\times\sqrt{n} = n$$

$$L(n) = \theta(\sqrt{n})$$

$$W(n) = \theta(\sqrt{n})$$

$$L(n) = 2L\left(\frac{n}{4}\right) + O(\sqrt{n} - \epsilon)$$

$$\sqrt{n} = n^{\frac{1}{2}} = n^{\log_b a}$$

$$a = 2 \quad b = 4$$

$$\log n^2 = \frac{1}{\log_2 4} = \frac{1}{2}$$

6. Multiply two matrices

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}\begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} \overset{\overset{O(n)}{\downarrow}}{a_1 b_1 + a_2 b_3} & - \\ - & - \end{bmatrix}$$

$$A = []_{n\times n} \qquad C = AB = \begin{bmatrix} O(n) & - \\ - & - \end{bmatrix}_{n^2}$$

$$D = A + B = T(n) = \theta(n^2)$$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$4 - \text{size}\,\frac{n}{2}\times\frac{n}{2}$$

$$AB = \begin{pmatrix} A_{11}B_{11} + A_{21}B_{21} & - \\ - & - \end{pmatrix}$$

$$(A_{11})_{\frac{n}{2}\times\frac{n}{2}}\,(B_{11})_{\frac{n}{2}\times\frac{n}{2}} \qquad\qquad x_0 y_1 + x_1 y_0$$

$$= \theta\left(\frac{n^3}{8}\right)\times 4 = \theta(n^3)$$

$$T(n) = 7T\left(\frac{v}{4}\right) + O(n)$$

$$T(n) = \theta\left(n^{\log_2 7}\right)$$