Analysis of Algorithms - CS 323/700

Lecture #3 - February 17, 2016

Taken by Kan Xue

Lecture notes:

- **Sorting terms**

  (1) **Inversion:** two elements out of order with respect to one another.

  Example:

  i < j

  a[i] > a[j]

  (2) Time complexity:

  a) Minimum (sorting time) = 0 (everything is sorted)

  b) Average (sorting time) = $\frac{n(n-1)}{4}$ or ($\frac{Maximum}{2}$)

  c) Maximum (sorting time) = $C(n,2) = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$

  (3) Stable sort: It is a sort that preserves the good order of non-converted elements.

  (4) Exchange: Swapping elements.

  (5) Internal sort V.S. External Sort

  Note: external sort means to take out a trunk of data and only sort that amount of data first. Our class will be focus on internal sort

- **Reminder:**

O	Big Oh	iff:	if and only if	$\mathbb{R}^+$:	positive real numbers
Ω	Big Omega	∃:	there exist		
θ	(Big) Theta	ϵ:	an element of		
o	Little Oh	∀:	for all		
ω	Little Omega	$\mathbb{N}$:	natural numbers		

- **$f(n) = O(g(n))$ iff $\exists\ n_0 \in \mathbb{N}$ and $C \in \mathbb{R}^+$ such that $\forall\ n \geq n_0$ $f(n) \leq Cg(n)$**

  Given $f(n) = 1000n^2 + 3n + 10$ and $g(n) = 10n^2 + 400n + 40$.

  <u>Example:</u>
  Find n_0 and C for: $1000n^2 + 3n + 10 = O(10n^2 + 400n + 40)$
  *<u>Note</u>: The value of n_0 and C **does not** have to be the smallest.*

  So we use C = 100 and we get:

  $1000n^2 + 3n + 10 \leq C(10n^2 + 400n + 40)$
  $1000n^2 + 3n + 10 \leq (1000n^2 + 40000n + 4000)$
  And it works for $n_0 \geq 0$

- **$f(n) = \Omega(g(n))$ iff $\exists\ n_0 \in \mathbb{N}$ and $C \in \mathbb{R}^+$ such that $\forall\ n \geq n_0$ $f(n) \geq Cg(n)$**

- **$f(n) = \theta(g(n))$ iff $\exists\ n_0 \in \mathbb{N}$ and $C_1 \in \mathbb{R}^+$ and $C_2 \in \mathbb{R}^+$ such that $\forall\ n \geq n_0$ $C_2g(n) \leq f(n) \leq C_1g(n)$**

  <u>Note</u>: $f(n) = \theta(g(n))$ means: $f(n) = O(g(n))$ and $g(n) = O(f(n))$
  also the same as: $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

- **<u>Sorting:</u>**
  - **Assuming we are sorting an array of integer, we first find the Min and Max.**

    Alghorithm to find Min:

```
tempMin = a[0]
for i = 1 to n-1
      if a[i] < tempMin
            tempMin = a[i]
```

    Alghorithm to find Max:

```
tempMax = a[0]
for i = 1 to n-1
      if a[i] > tempMin
            tempMax = a[i]
```

  Sorting time for finding Min or Max = n

- **Method:**

1. <u>Linear Search:</u> go through everything and compare (decrease-by-one and conquer)

	Comparisons
Best	1
Average	n/2
Worst	n

<u>Note</u>: if the key is in the array, average case will be n/2. Otherwise, average case will also be n.

Find the time complexity T(n) with recurrence:

$T(1) = 1$

$T(n) = T(n-1) + 1 = [T(n-2)+1] + 1 = [T(n-3)+1]+1+1$ (if we keep expanding, there will be (n-1) of 1's for $T(1)^{th}$ term.)

Therefore,

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

Or, we can rewrite the equation as the following:

$$\left.\begin{array}{ll} T(n) - T(n-1) & = 1 \\ T(n-1) - T(n-2) & = 1 \\ . \qquad . & \\ . \qquad . & \\ T(2) - T(1) & = 1 \end{array}\right\} \quad \boxed{\begin{array}{l}\text{total (n-1) equations}\\ \text{so there will be (n-1)}\\ \text{of 1's}\end{array}}$$

Add up all expressions on the left(as we did for geometric series) we get:

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

$T(n) - 1 \quad = n-1$

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

2. <u>Binary Search:</u> take the ceiling value of $\frac{Low+high}{2}$ to find the mid-point(decrease by half and conquer)

<u>Note</u>: For each iteration, there will be key comparisons and index comparisons.

Find the time complexity T(n) with recurrence:

$T(1) = 1$
$T(n) = T(\frac{n}{2}) + 1$
$T(n) = T(\frac{n}{4}) + 1 + 1$
$T(n) = T(\frac{n}{8}) + 1 + 1 + 1$
...

there will be total $\log_2 n$ levels so it will be ($\log_2 n$ + 1) of 1's in total

Hence, the time complexity for binary search is $\log_2 n + 1$

Or, we can do it using Domain Transformation(substitution)

Example:

Note:
$K=0 \rightarrow T(2^0) = T(1) = 1 \Rightarrow S(0) = 1$
$T(\frac{n}{2}) = T(2^{k-1}) = S(K-1)$

Assume $n = 2^k$,
$T(n) = T(2^k) = S(k)$
$S(0) = 1$
$S(k) = S(k-1) + 1$
$S(k) = S(k-2) + 1 + 1$
...
$S(k) = k + 1$

total k terms plus S(0) gives k+1 terms

Then substitute back into the original equation:

$T(2^k) = k + 1$
$T(n) = k + 1 = \log_2 n + 1$ (take the floor value of $\log_2 n$ here)

Application: In case we have an array with size *m* and *m* is not power of 2 where $2^k < m < 2^{k+1}$. We then can add artificial elements(none key values) to the array to make the size of the array to be power of 2.

Example:

If there are 65 elements in the array, we can increase them to 128 elements. It will only need 1 extra comparison to get rid of 64 elements in the array which makes the problem to be easier to deal with.

- **Bubble Sort(integer):**

  Algorithm:

```
for(i = n-1; i > 0; i--)
   for(j = 0; j < i; j++)
       if(a[j] > a[j+1])
          temp = a[j]       }
          a[j] = a[j+1]     }   swap
          a[j+1] = temp     }
```

  To count the number of operations, we will include both swaps and comparisons. (*Note: swap using pointer is less expensive)*

  Time Complexity:

	Comparison	**Swap**
Best	$\frac{n(n-1)}{2}$	**0**
Average	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{4}$
Worst	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$

Homework #2 solution

The first three questions pertain to the recurrenceg(0) = 2, g(1) = 1, g(n) = g(n-1) + 2g(n-2)

1. Using the method of characteristic equations presented in class, derive the closed-form formula g(n) = 2^n + $(-1)^n$.

1) g(n) - g(n-1) - 2g(n-2) = 0
2) substitute: $x^n - x^{n-1} - 2x^{n-2} = 0$
3) factor: $x^{n-2}(x^2-x-2) = 0$
4) characteristic: $x^2-x-2 = 0$
5) solve for x: x = 2 or x = -1

6) put values in for the equation:

$g(n) = p(2)^n + q(-1)^n$

$g(0) = p(2)^0 + q(-1)^0 = 2 \Rightarrow p + q = 2$

$g(1) = p(2)^1 + q(-1)^1 = 1 \Rightarrow 2p - q = 1$

7)solve the system:

$$\begin{cases} p + q = 2 \\ 2p - q = 1 \end{cases}$$

$$\begin{cases} p = 1 \\ q = 1 \end{cases}$$

8)put back the coefficients and we will find the formula:

$g(n) = 2^n + (-1)^n$

2. **Using mathematical induction, prove that the formula $g(n) = 2^n + (-1)^n$ works for all cases $n \geq 0$.**

Base case: $n = 0$

$g(0) = 2^0 + (-1)^0 = 2$

Inductive step:

Inductive hypothesis, assume $g(k) = 2^k + (-1)^k$ is true for $0 \leq k \leq n$.

$g(k+1) = 2^{k+1} + (-1)^{k+1}$

$g(k+1) = g(k) + 2g(k-1) = 2^k + (-1)^k + 2[2^{k-1} + (-1)^{k-1}]$

$= 2^k + 2^k + (-1)^k - 2(-1)^k = 2^{k+1} - (-1)^k = 2^{k+1} + (-1)^{k+1}$

3. **Using the logarithm-based approximation method discussed in class, estimate how many digits are in g(1000). Then use an on-line calculator to find out the actual number of digits in g(n).**

$2^n \Rightarrow (2^x)^{n/x}$

$\Rightarrow (10^{3.32})^{1000/3.32} \approx 10^{301}$

calculator result: 1.07^{301}

$$\boxed{\log_2 10 = \frac{\log 10}{\log 2} \approx 3.32}$$

4. **Suppose n unsorted elements are in a stack. Describe how to sort the data, using auxiliary stacks as necessary. What is the time complexity of your algorithm?**

1) iterate through stack
2) define a tempMax
3) push tempMax into original stack and assign 2nd largest value to be new tempMax
4) repeat 1-3.

Time Complexity: $O(n^2)$

5. **Suppose n unsorted elements are in a queue. Describe how to sort the data, using auxiliary queues as necessary. What is the time complexity of your algorithm?**

    1) iterate through the queue, dequeue the first one and enqueue it back to the queue to find Min.

    2)repeat the procedure and enqueue the smallest key to the other queue until it is finished.

    Time Complexity: O(n^2)

6. **Suppose n unsorted elements are in a "binary search tree", defined as a binary tree in which the keys in the left and right children are respectively smaller and larger than the key in their parent node. Describe how to sort the data, using auxiliary BSTs as necessary. What is the time complexity of your algorithm?**

    perform an inorder traversal with following order:

    inorder(Left-Sub-Tree), Root, inorder(Right-Sub-Tree)

    Time Complexity: O(n)

7. **Suppose n unsorted elements are in a "heap", defined as a binary tree in which the keys in the left and right children are *both* larger than the key in their parent node. Describe how to sort the data, using auxiliary heaps as necessary. What is the time complexity of your algorithm?**

    Heap is a complete tree, every levels from root is larger than its previous level.

    Use deleteMin to promote the next smallest one and push entire tree up.

    Time Complexity: O(nlogn)

8. **Given the various possible shapes of a binary tree, what are the best, average and worst case number of operations for "preorder" traversal? Why?**

    It does not matter what shape the tree is because the preorder traversal travels through every node once.

    Time Complexity: O(n)