# CIIC4025/ICOM4038

Design and Analysis of Algorithms

Solution Exam 1 (Prof. Wilson Rivera)

## Instructions

- Write your solutions in the space provided. Extra scratch paper may be provided if you need more room, although your answer should fit in the given space.
- The exam is closed book, closed notes and closed phone.
- Good luck!

| | | |
|---|---|---|
| 1A | 5 points | |
| 1B | 5 points | |
| 2A | 10 points | |
| 2B | 10 points | |
| 3A | 10 points | |
| 3B | 5 points | |
| 4 | 5 points | |
| 5A | 10 points | |
| 5B | 10 points | |
| 6A | 10 points | |
| 6B | 10 points | |
| 7 | 10 points | |
| TOTAL | 100 Points | |

1. Prove or disprove the following statements

   (a) $2^{2n} = O(2^n)$

   FALSE.
   Suppose it is true. Then, there exist constants $c$ and $n_0$ such that $2^{2n} \leq c2^n$, for any $n \geq n_0$. Let be $n_0 = 1$, then $2^{2n} = 2^n 2^n \leq c2^n$. Thus, $2^n \leq c$ for any $n \geq 1$. which is clearly impossible since $c$ is a constant. Contradiction !!

   (b) $f(n) + O(f(n)) = \Theta(f(n))$

   TRUE.
   suppose that $g(n) \leq cf(n)$, $\forall n \geq n_0$.
   Let be $c = 1$. Then $g(n) \leq f(n)$.
   Thus, $f(n) + g(n) \leq f(n) + f(n) = 2f(n)$
   As a result, $f(n) + O(f(n)) = O(f(n))$

   Now choosing $n > n_0$ such that $g(n) > 0$,
   we have that $f(n) \leq f(n) + g(n)$
   Hence, $f(n) + O(f(n)) = \Omega(f(n))$

   Then, we can conclude that $f(n) + O(f(n)) = \Theta(f(n))$

2. Find the order of execution time for the following recurrent forms

   (a) $T(n) = 2T(n/2) + \frac{n}{\log n}$
   Let be $n = 2^k$ then

$$\begin{aligned}
T(2^k) &= 2T(2^{k-1}) + \frac{2^k}{k} \\
&= 2[2T(2^{k-2}) + \frac{2^{k-1}}{k-1}] + \frac{2^k}{k} \\
&= 2^2T(2^{k-2}) + \frac{2^k}{k-1} + \frac{2^k}{k} \\
&= 2^jT(2^{k-j}) + \frac{2^k}{k-j} + \cdots + \frac{2^k}{k-1} + \frac{2^k}{k} \\
&= 2^kT(2^{k-k}) + \sum_{j=0}^{k-1} \frac{2^k}{k-j} + 2^k \\
&= 2^kT(1) + \sum_{j=0}^{k-1} \frac{2^k}{k-j} + 2^k \\
&= 2^k[\sum_{j=1}^{n} \frac{1}{j} + 2] \\
&= O(2^k \log k) \\
&= O(n \log(\log n))
\end{aligned}$$

   (b) $T(n) = T(n-2) + 7n$

$$\begin{aligned}
T(n) &= T(n-2) + 7n \\
&= T(n-4) + 7(n-2) + 7n \\
&= T(n-6) + 7(n-4) + 7(n-2) + 7n \\
&= T(0) + 7(2) + \cdots + 7(n-4) + 7(n-2) + 7n \\
&= O(7n\frac{n}{2}) = O(n^2)
\end{aligned}$$

3. Given an arbitrary red-black tree. let $b(x)$ be the number of black nodes on a path from node $x$ to $NIL$.

   (a) Prove that the number of non-NIL descendents of $x$ is at least $2^{b(x)} - 1$

   (b) Prove that the height of the tree is less than $2\log(n+1)$

   SOLUTION.

   (1) NIL node has $b(x) = 0$ and $2^0 - 1 = 0$ non NIL descendents
   (2) assume that the number of descendents in a tree of order $k < n$ is given by $2^{b(x)-1} - 1$
   (3) Let $d(x)$ be the number of descendents of $x$ then

$$\begin{aligned} d(x) &= 1 + d(left(x)) + d(right(x)) \\ &\geq 1 + (2^{b(x)-1} - 1) + (2^{b(x)-1} - 1) \\ &= 2 * 2^{b(x)-1} - 1 \\ &= 2^{b(x)} - 1 \end{aligned}$$

   Let $h = h(r)$ the height from the root. Then, $b(r) \geq \frac{h(r)}{2}$
   Thus, $n \geq 2^{b(r)} - 1 > 2^{h/2} - 1$
   Hence, $n + 1 \geq 2^{h/2}$ then $\log(n+1) \geq h/2$.
   As a result, $h \leq 2\log(n+1)$

4. Consider the family of hash functions $h_b(x) = 5x + b \bmod 11$, where $b \in [0, 10]$ and the argument $x$ is from the universe $\{0, 1, 2, \ldots 21\}$. Is this family universal? Explain.

   SOLUTION.

   The provided family is not universal. To show that a family of hash functions $h_b(x)$ is not universal, it is sufficient to show that for a hash function $h$ randomly chosen from $h_b$,
   $P(h(x_i) = h(x_j)) > \frac{1}{n} = \frac{1}{11}$ for some $x_i \neq x_j$

   If we take $x_i = 0$ and $x_j = 11$, then for any hash function $h$ (any value of $b$ ) that we select from the family, $h(0) = h(11)$ because $5 * 0 + b \equiv 5 * 11 + b \pmod{11}$.
   Therefore, $P(h(0) = h(11)) = 1 > \frac{1}{11}$

5. Consider a modification to merge sort in which $n/k$ sublists of length $k$ are sorted using insertion sort and then merged using the standard merging mechanism, where $k$ is a value to be determined.

   (a) Show that insertion sort can sort the $n/k$ sublists, each of length $k$, in $\Theta(nk)$ worst-case time.

   (b) Show how to merge the sublists in $\Theta(n \log(n/k))$ worst-case time.

   SOLUTION

   Insertion sort takes $\Theta(k^2)$ time per $k$-element list in the worst case. Therefore, sorting $n/k$ lists of k elements each takes $\Theta(k^2 n/k) = \Theta(nk)$ worst-case time. Extending the 2-list merge to merge all the lists at once would take $\Theta(n*(n/k)) = \Theta(n^2/k)$ time ($n$ from copying each element once into the result list, $n/k$ from examining $n/k$ lists at each step to select next item for result list).

   To achieve $\Theta(n \log(n/k))$-time merging, we merge the lists pairwise, then merge the resulting lists pairwise, and so on, until there is just one list. The pairwise merging requires $\Theta(n)$ work at each level, since we are still working on $n$ elements, even if they are partitioned among sublists. The number of levels, starting with $n/k$ lists (with $k$ elements each) and finishing with 1 list (with $n$ elements), is $\log(n/k)$. Therefore, the total running time for the merging is $\Theta(n \log(n/k))$.

6. Given two sets $A$ and $B$ of n integers

   (a) Give an efficient deterministic algorithm to find $A \bigcap B$.

   (b) Can you do better with randomization? Design a randomized algorithm and compare its run time order to the run time order of the deterministic algorithm in (a).
   SOLUTION

   For the deterministic algorithm, sort each list. Then, iterate through the elements looking for elements common to both arrays. This takes $O(n \log n)$ time.
   Using randomization, hash the elements of A. Then iterate through $B$, looking up elements in the hash of $A$. The expected running time is $O(n)$.

7. Consider two disjoint sorted arrays $A[1...m]$ and $B[1...n]$. Find a $O(\log k)$ time algorithm for computing the $k^{th}$ smallest element in the union of the two arrays. Prove the order using the recurrent form of the algorithm.

   SOLUTION

   Consider $A[k/2]$ and $B[k/2]$. Without loss of generality, assume $A[k/2] < B[k/2]$. Then $A[k/2]$ is greater than at most k elements. Furthermore the elements $A[1...\frac{k}{2}-1]$ are all less than the $k^{th}$ element, so we can eliminate them. Similarly, $B[k/2]$ is greater than at least $k$ elements, so the elements $B[\frac{k}{2}+1...n]$ are all larger than the $k^{th}$ element. We can therefore eliminate them too. We are therefore left with two subarrays, and we now want to find the $k/2^{th}$ element (since we eliminated $k/2$ elements that were guaranteed to be less than the $k^{th}$ element). This divide-and-conquer algorithm follows the recursion $T(k) = T(k/2) + 1$, which is $O(\log k)$.