The Hong Kong University of Science & Technology
Department of Computer Science

COMP 171: Data Structures and Algorithms
Written Assignment 1
Out on October 5, 2005
Due on October 18, 2005 (at the beginning of class)

Your answers will be graded on clarity, correctness, efficiency, and precision.

1. For each pair of $f(n)$ and $g(n)$ below, decide if $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. Justify your answer using the definitions of these asymptotic notation. Note that more than one of these relations may hold for a given pair; list all correct ones.

   (a) $f(n) = \sqrt{n}$ and $g(n) = \log_2 n$.
   (b) $f(n) = \log_2^3 n$ and $g(n) = \log_2 n^3$.
   (c) $f(n) = 2^n$ and $g(n) = 2^{n/2}$.
   (d) $f(n) = \log_2(n!)$ and $g(n) = n \log_2 n$.

   Answer: (limit rule)

   (a) $\because \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{(1/2)n^{-1/2}}{1/(n \ln 2)} = \lim_{n\to\infty} \frac{1}{2} n^{1/2} \ln 2 = \infty$,
   $\therefore f(n) = \Omega(g(n))$.

   (b) $\because \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log_2^3 n}{3 \log_2 n} = \lim_{n\to\infty} (1/3) \log_2^2 n = \infty$,
   $\therefore f(n) = \Omega(g(n))$.

   (c) $\because \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{(2^{n/2})^2}{2^{n/2}} = \lim_{n\to\infty} 2^{n/2} = \infty$,
   $\therefore f(n) = \Omega(g(n))$.

   (d) **Solution 1** According to the Stirling's Approximation: $n! \approx \sqrt{2\pi n} n^n e^{-n}$,
   $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log_2 \sqrt{2\pi n} + n \log_2 n - n \log_2 e}{n \log_2 n} = 1$,
   $\therefore f(n) = \Theta(g(n))$ (which implies $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

   Solution 2 $\because n! < n^n$, $\therefore f(n) < g(n) \Rightarrow f(n) = O(g(n))$.
   On the other hand,

$$\begin{aligned} \log_2(n!) &= \log_2 n + \log_2(n-1) + \cdots + \log_2 3 + 1 \\ &\geq \log_2 n + \log_2(n-1) + \cdots + \log_2(n/2) \\ &\geq (n/2) \log_2(n/2) \\ &= \Omega(n \log_2 n). \end{aligned}$$

   $\therefore f(n) = \Omega(g(n))$.
   With $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, we get $f(n) = \Theta(g(n))$.

   Marking scheme:

   - Full mark is 10. We distribute it to four subproblems evenly (2.5 for each). For each subproblem, 1.5 corresponds to the explanation and 1.0 goes to the conclusion.

- The explanations based on either the limit rule or the definitions of the asymptotic notations are both correct. 0.5 will be deducted for each error. One possible error is: for 1(d), $\log_2 1 + \log_2 2 + \cdots + \log_2 n = (n/2)(\log_2 1 + \log_2 n)$.
- For the conclusions, 1 for giving all correct relations; otherwise, 0 mark. (Only the last subproblem has three correct relations.)

2. Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove each of the following conjectures.

   (a) $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.

   (b) $f(n) = \Theta(f(n/2))$.

   Answer:

   (a) FALSE statement. Counterexample: when $f(n) = n$ and $g(n) = n^2$, $f(n) = O(g(n))$ but $g(n) \neq O(f(n))$ ($\because \lim_{n\to\infty} g(n)/f(n) = \infty$).

   (b) FALSE statement. Counterexample: when $f(n) = 2^n$, $f(n) \neq \Theta(f(n/2))$ ($\because \lim_{n\to\infty} f(n)/f(n/2) = \infty$).

   Marking scheme:

   - Full mark is 10. We distribute it to two subproblems evenly (5.0 for each). For each subproblem, 3 corresponds to the explanation and 2 goes to the conclusion.
   - If the conclusions are wrong, no mark will be given.
   - 1 mark will be deducted for each error.

3. Solve the following recurrrence relation: $T(1) = 1$, $T(n) = T(\frac{n}{2}) + O(n)$, where $n > 1$.

   Answer:

$$\begin{aligned} T(n) &= T(\frac{n}{2}) + n \\ &= T(\frac{n}{2^2}) + \frac{n}{2} + n \\ &= T(\frac{n}{2^3}) + \frac{n}{2^2} + \frac{n}{2} + n \\ &\vdots \\ &= T(\frac{n}{2^k}) + \sum_{i=0}^{k-1} \frac{n}{2^i} \\ &= T(\frac{n}{2^k}) + 2n(1 - \frac{1}{2^k}) \end{aligned}$$

   $\therefore$ when $\frac{n}{2^k} = 1$, we get

$$T(n) = T(1) + 2n(1 - \frac{1}{n}) = O(n).$$

   Marking scheme:

   - Full mark is 10.

- Under the condition that $T(n) = O(n)$ is deduced, for each error, 1 mark will be deducted.
- If the deduced result is $O(n \log n)$ or $O(\log n)$, 5 marks are given.
- No mark will be given if conclusions other than $T(n) = O(n)$, $T(n) = O(\log n)$ or $T(n) = O(n \log n)$ are made.

4. Describe an algorithm to perform mergesort non-recursively. Use plain English text. Do not give us any C++ code or pseudo-code.

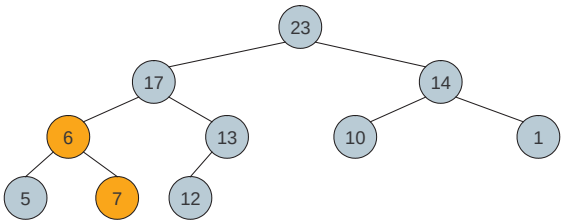
   Answer:
   Non-recursive mergesort is done by scanning through the array performing 1-by-1 merges to produce sorted sublists of size 2; then scan through the array performing 2-by-2 merges to produce sorted sublists of size 2, then do 4-by-4 merges to get sorted sublists of size 8, etc., until the whole list is sorted.

   Marking scheme:

   - Full mark is 10.
   - If the basic idea is correct, 1 mark will be deducted for each error.
   - If the basic idea is wrong, 0 mark.

5. (a) Is the array $\{23, 17, 14, 6, 13, 10, 1, 5, 7, 12\}$ a heap?
   (b) Given k sorted lists containing a total of n elements. Design an $O(n \log k)$ algorithm to merge these lists into a single sorted list.

   Answer:



   (a) No, it is not a heap. Elements 6, 7 violate the (max) heap definition (see the above figure).

   (b) Without loss of generality, we assume that the k given sorted lists are in increasing order. We use a min heap to solve this problem. First, build a min heap using the first element of each sorted list as key values. Each node of the heap is now a sorted list. Building such a min heap costs $O(k \log k)$. Second, remove the first element from the list corresponding to the root of the heap (and store it in a second array). If the resulting list is not empty, use the (new) first element to restore the min heap property by "bubbling down" (nodes consisting of sorted lists are swapped if the interchanging condition is satisfied); if the resulting list is empty, then delete the node by copying the last node (a sorted list) to the root and use its first element to heapify. Repeat Step 2 until the heap is empty. The resulting second array is now sorted. Since we perform Step 2 n times, each taking $O(\log k)$ time, the total cost is $O(n \log k)$.

Marking scheme:

- Full mark is 10. 3 for (a) and 7 for (b).
- For (a), no explanation is needed. 3 marks will be given if the correct conclusion is made.
- For (b), 5 for the algorithm designing and 2 for the running time analysis.
- Under the condition that the basic idea is correct, 1 mark will be deducted for each error.
- If the basic idea is wrong, 0 mark.

6. Let $A[0..n-1]$ be a (min) heap of size n. Let $A[j]$ be a specific entry given to you. You are to design algorithms to support the following operation on the heap. Given an input parameter k such that $k > A[j]$, describe an algorithm to increase the value of $A[j]$ to k using plain English text. (Do not give us any C++ code or pseudo-code.) Note that j might not be equal to 0 or $n-1$. Your algorithm should restore the heap order after increasing the value of $A[j]$. Analyze the worst-case running time of your algorithm. Your algorithm should be as efficient as possible. In particular, rebuilding a heap from scratch is not a satisfactory solution.

   Answer:

   When $A[j]$ is increased to k, the resulting tree may violate the heap property. Like DeleteMin, the solution is to restore the heap property by "bubbling down". Specifically, if the element is greater than the smallest of its two children, then interchange the parent and child.

   Worst Running Time = $O(height) = O(\log n)$.

   Marking scheme:

   - Full mark is 10.
   - 7 for the algorithm and 3 for the running time analysis.
   - Under the condition that the basic idea is correct, 1 mark will be deducted for each error.
   - If the basic idea is wrong, 0 mark.