Sample Solution to Problem Set 2

Problem 5 of PS1. (3 + 3 + 4 = 10 points) Properties of asymptotic notation

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

(a) $f(n) + g(n) = \Theta(\min\{f(n), g(n)\})$.

Answer: False. Consider $f(n) = n$ and $g(n) = n^2$. The left-hand side of the above equation is $f(n) + g(n) = n^2 + n$, while the right side is $\min\{f(n), g(n)\} = n$, since $n \leq n^2$ for all $n \geq 1$. So we need to compare $n^2 + n$ with n. Taking their ratios and then the limit as n tends to infinity, we obtain:

$$\lim_{n\to\infty} \frac{n^2 + n}{n} = \lim_{n\to\infty} n + 1 = \infty.$$

Obviously, $f(n) + g(n) = n^2 + n = \Theta(n^2) = \Theta(g(n)) = \Theta(max\{f(n), g(n)\})$.

(b) If $f(n) = O(h(n))$ and $g(n) = \Omega(h(n))$, then $f(n) = O(g(n))$.

Answer: True. Since $f(n) = O(h(n))$, there exist positive constants c_1 and n_1 such that $f(n) \leq c_1 h(n)$ for all $n \geq n_1$. Similarly, since $g(n) = \Omega(h(n))$, there exist positive constants c_2 and n_2 such that $g(n) \geq c_2 h(n)$ for all $n \geq n_2$. This implies that for all $n \geq \max\{n_1, n_2\}$, $f(n) \leq c_1 h(n) \leq c_1 g(n)/c_2$. Therefore, $f(n) = O(g(n))$.

(c) $f(n^2) = \Omega((f(n))^2)$.

Answer: False. Consider $f(n) = \lg n$. Then, $f(n^2) = 2 \lg n$, while $(f(n))^2 = \lg^2 n$. Clearly, $f(n^2) \neq \Omega((f(n))^2)$, since

$$\lim_{n\to\infty} \frac{2 \lg n}{\lg^2 n} = \lim_{n\to\infty} \frac{2}{\lg n} = 0.$$

1. (10 points) Towers of Hanoi

The Towers of Hanoi problem is a classic example of recursion. In the Towers of Hanoi problem, n disks of different sizes are piled on a peg in order of their size, with the largest at the bottom. There are two empty pegs. The problem is to move all the disks to the third peg by moving only one disk at a time and never placing a disk on top of a smaller one. The second peg may be used for intermediate moves.

The usual solution recursively moves all but the last disk from the starting peg to the spare peg, then moves the remaining disk (the largest one) from the start peg to the destination peg, and then recursively moves all the others from the spare peg to the destinstion peg. This recursive solution is described in the following recursive procedure.

HANOI(n, *start*, *destination*, *spare*)
1. **if** $n > 0$
2. HANOI($n-1$, *start*, *spare*, *destination*)
3. **print** "move top disk from peg ", *start*, "to peg ", *destination*
4. HANOI($n-1$, *spare*, *destination*, *start*)

Write a recurrence for the number of moves for n disks. Solve this recurrence.

Answer: Let $H(n)$ denote the number of moves for n disks. From the recursive solution, we obtain

$$H(n) = \begin{cases} 2H(n-1)+1 & n > 0, \\ 0 & n = 0 \end{cases}$$

We use the iteration method to solve the recurrence. Upon unraveling the recurrence i times, we obtain

$$H(n) = 2^i H(n-i) + 2^{i-1} + 2^{i-2} + \ldots + 2^1 + 1.$$

Setting $i = n$, we get

$$\begin{aligned} H(n) &= 2^n H(0) + 2^{n-1} + 2^{n-2} + \ldots + 2^1 + 1 \\ &= 2^{n-1} + 2^{n-2} + \ldots + 2^1 + 1 \\ &= 2^n - 1. \end{aligned}$$

2. (20 points) Recurrences Find the asymptotic solution (using the Θ-notation) for each of the following recurrences. Assume that $T(n)$ is constant for n sufficiently small.

(a) $T(n) = T(n/2) + 2n^2$.

Answer: We use the Master method. We have $a = 1, b = 2$, and $f(n) = 2n^2$. We obtain $\log_b a = 0$ and $n^{\log_b a} = n^0 = 1$. Since $2n^2 = \Omega(n^{0+2})$, there exists an $\varepsilon > 0$ (for example, $\varepsilon = 2$) such that $f(n) = \Omega(n^{\log_b a + \varepsilon})$. This suggests case 3 of the Master Theorem. We need to check for another condition, though. This is whether $af(n/b) \leq cf(n)$ for some $c < 1$. Thus, we need to compare $f(n/2)$ with $f(n)$. Since $f(n/2) = n^2/2$ and $f(n) = 2n^2$, we obtain that $c = 1/4$ ensures that $af(n/b) \leq cf(n)$. Thus, we invoke case 3 of the Master theorem to obtain $T(n) = \Theta(n^2)$.

(b) $T(n) = 81T(n/3) + n^4/3$.

Answer: We use the Master method. We have $b = 81, c = 3$, and $f(n) = n^4/3$. We obtain $E = \lg 81/\lg 3 = 4$ and $n^E = n^4$. Since $n^4/4 = \Theta(n^4)$, we have $f(n) = \Theta(n^E)$. Thus, we invoke case 2 of the Master theorem to obtain $T(n) = \Theta(n^4 \lg n)$.

(c) $T(n) = T(\sqrt{n}) + 1$.

Answer: We cannot use the master method. We therefore use the recursion tree approach, shown here as an iteration (without the picture).

$$\begin{aligned} T(n) &= 1 + T(n^{1/2}) \\ &= 1 + 1 + T(n^{1/4}) \\ &= i + T(n^{1/2^i}) \\ &= \Theta(\lg\lg n) + T(2) \text{ (since } n^{1/2^{\lg\lg n}} = n^{1/\lg n} = 2^{(\lg n)/(\lg n)} = 2) \\ &= \Theta(\lg\lg n). \end{aligned}$$

We note that we iterate until n comes down to 2, rather than 1 as we have been usually doing. The reason we do this is that if we repeatedly take the square root of a number bigger than 1, we will never obtain 1 in a finite number of steps. Therefore, the recurrence is undefined below $n = 2$. Since we know that $T(2)$ is constant, we stop the recurrence at 2 and derive the final result.

(d) $T(n) = T(n/5) + T(n/3) + 2n$.

Answer: We use the recursion tree approach. The contribution of the first level is $2n$. The contribution of the next level is $2n(1/3 + 1/5) = 2n \cdot (8/15)$. The contribution of the third level can be shown to be $2n \cdot (8/15)^2$. Thus, the total contribution is at most

$$2n \sum_{i=0}^{\infty} \left(\frac{8}{15}\right)^i = \frac{30n}{7}.$$

Thus, we have $T(n) = O(n)$. Clearly $T(n) \geq 2n$; therefore, $T(n) = \Omega(n)$. Thus, $T(n) = \Theta(n)$.

3. (10 points) FIFO queues and stacks

Exercise 6.5-6, page 142.

Answer: Whenever we add a record x to our dynamic set S, we augment x by a field, which we call its *sequence number*. Let $s[x]$ denote the sequence number of x. We also maintain a variable *Max* that stores the largest sequence number assigned thus far.

We represent a first-in first-out queue or a stack by a priority queue in where the key is the sequence number.

- First-in first-out queue: We use a priority queue that provides the INSERT and EXTRACT-MIN operations. We assume that if S is empty then EXTRACT-MIN returns NIL. The operations ENQUEUE and DEQUEUE for a FIFO queue Q are defined as follows:

  ENQUEUE(Q, x)
  1. $Max \leftarrow Max + 1$
  2. $s[x] \leftarrow Max$
  3. INSERT(S, x)

  DEQUEUE(Q, x)
  1. $x \leftarrow$ EXTRACT-MIN(S)
  2. **if** $x =$ NIL
  3. **return** NIL
  4. **else return** x

  For the correctness of these operations, we need to prove that if an item x is enqueued after item y, then x is dequeued after y. If x is enqueued after y, then since the sequence number always increases, the key of x is greater than the key of y. When x is enqueued, we have two cases. If y has already been dequeued, then there is nothing to prove. Otherwise, both x and y are in the priority queue. Since the key of x is greater than the key of y and the dequeuing operation always extracts the item with the minimum key, it follows that y is dequeued before x.

- Stack: We will use a priority queue that provides the INSERT and EXTRACT-MAX operations. We assume that if S is empty then EXTRACT-MAX returns NIL. The operation PUSH is exactly the same as ENQUEUE above. The operation POP is exactly the same as DEQUEUE above with EXTRACT-MIN replaced by EXTRACT-MAX. The correctness of these operations follows in a manner analogous to the correctness of the first-in-first-out queue operations given above.

  The operation STACK-EMPTY can be defined as follows:

  STACK-EMPTY(T, x)
  1. **if** EXTRACT-MAX(S) = NIL, then **return true**
  2. **else return false**.

4. (10 points) Building a heap using insertion

Problem 6–1, page 142.

Answer:

(a) BUILD-HEAP' and BUILD-HEAP may return different heaps on the same input. Consider the input array $A = [1, 2, 3]$. The routine BUILD-HEAP only runs HEAPIFY on the node with value 1 to obtain the heap $[3, 2, 1]$. On the other hand, BUILD-HEAP' first inserts the element 2 into the subarray $[1]$ to obtain the heap $[2, 1]$ and then inserts the element 3 into the array $[2, 1]$ to obtain $[3, 1, 2]$. So the two heaps obtained are different.

(b) We know that the HEAP-INSERT operation takes $O(\lg k)$ time on a heap of size k. Since we have n HEAP-INSERT operations, each running on a heap of size at most n, the total running time is $O(n \lg n)$.

We now prove a matching lower bound. Consider the input $[1, 2, \ldots, n]$; that is, the elements are in increasing order. In each of the n HEAP-INSERT operation executed by BUILD-HEAP', the element being inserted is sent to the root of the binary tree. That is, when element i is being inserted, then $\lfloor \lg i \rfloor$ comparisons are performed. Therefore, the total running time is at least

$$\begin{aligned}
\sum_{i=2}^{n} \lfloor \lg i \rfloor &\geq \sum_{i=2}^{n} (\lg i - 1) \\
&\geq (\sum_{i=\lceil n/2 \rceil}^{n} \lg i) - n \\
&\geq \frac{n}{2} \lg \left(\frac{n}{2} \right) - n \\
&= \frac{n \lg n}{2} - \frac{3n}{2}.
\end{aligned}$$

We would like to show that $(n \lg n)/2 - (3n)/2 = \Omega(n \lg n)$. We do this by selecting positive constants c and n_0 such that $(n \lg n)/2 - (3n)/2 \geq cn \lg n$ for all $n \geq n_0$. Since we have $(n \lg n)/2$ on the left hand side of the inequality, we need to pick $c < 1/2$. Let us pick $c = 1/4$. So we need to prove that $(n \lg n)/2 - (3n)/2 \leq n \lg n/4$. On simplifying this, we obtain the condition that $\lg n \geq 6$, which translates to $n \geq 64$. Thus, we can set $c = 1/4$ and

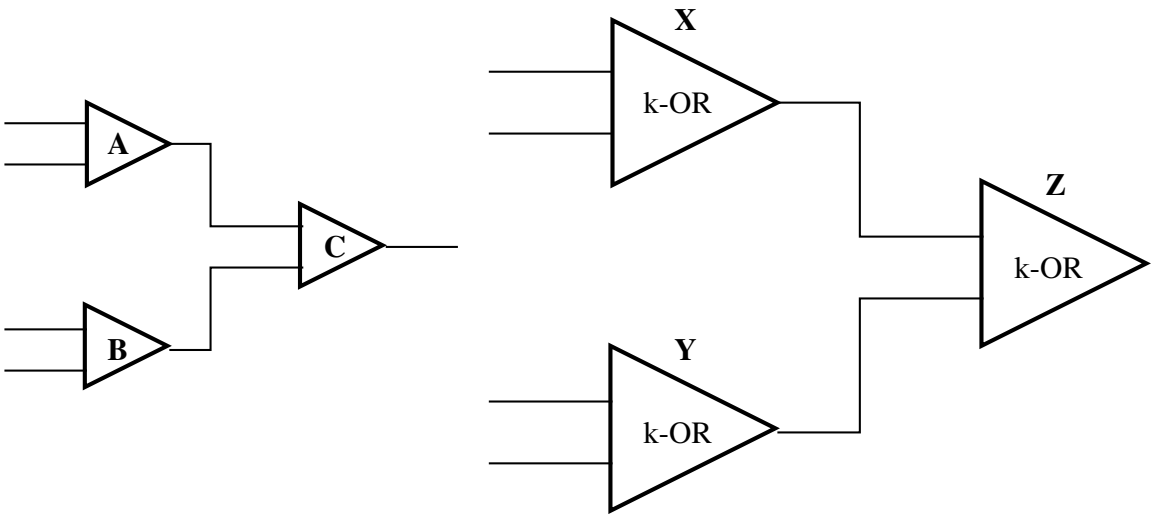
$n_0 = 64$ to obtain that $(n \lg n)/2 - (3n)/2 \geq cn \lg n$ for all $n \geq n_0$. Therefore, the worst-case running time of BUILD-HEAP' is $\Omega(n \lg n)$.

5. (10 bonus points) A fault-tolerant OR-gate

Assume we are given an infinite supply of two-input, one-output gates, most of which are OR gates and some of which are AND gates. Unfortunately the OR and AND gates have been mixed together and we can't tell them apart. For a given integer $k \geq 0$, we would like to construct a two-input, one-output combinational "k-OR" circuit from our supply of two-input, one output gates such that the following property holds: If at most k of the gates are AND gates then the circuit correctly implements OR. Assume for simplicity that k is a power of two.

For a given integer $k \geq 0$, we would like to design a k-OR circuit that uses the smallest number of gates. Design the best possible circuit you can and derive a Θ-bound (in terms of the parameter k) for the number of gates in your k-OR circuit.

Answer: We construct a $\Theta(k^{\lg 3})$-gate k-OR circuit. The construction is recursive (divide-and-conquer). For $k = 1 = 2^0$, we have the circuit given in Figure 1(a). The circuit for $k = 2^{i+1}$ is constructed out of 3 2^i-OR circuits, as depicted in Figure 1(b).



(a) A 1-OR circuit (b) A 2k-OR circuit

Figure 1: Recursive construction of a k-OR circuit

In order the prove the correctness of the circuit, we first note that the output of the AND and OR gates match when both the inputs are identical (that is, when they are either 0 0 or 1 1). It follows that any 2-input circuit consisting of AND and OR gates only will yield as an output 0 (resp., 1) if both of its inputs are 0s (resp., 1s), regardless of the number of AND gates. Therefore, any circuit consisting of AND and OR gates only will yield the correct OR output when the two inputs are identical. So we need only consider the case when one of the inputs is a 0 and the other is a 1. In this case, the desired output is a 1.

The proof of correctness is by induction on i. For the induction base $k = 2^0 = 1$, we note that at

most one of A, B, or C is an AND gate. If C is an AND gate, then the outputs of A and B yield 1 each, implying that the output of C is also a 1. If C is an OR gate, then one of A and B is an OR gate; therefore, at least one of the inputs to C is a 1, implying that the output of C is also 1, thus establishing the correctness of the circuit.

As the induction hypothesis, we assume that k-OR circuits for $k = 2^i$ are correct, for some $k \geq 0$. We now prove the correctness of the $2k$-OR circuit. Since there are at most $2k$ AND gates, only one of the 3 circuits X, Y, and Z has more than k AND gates. This implies that two of the 3 circuits correctly compute an OR. If X and Y are OR circuits, then the outputs of the 2 gates are 1s. Since the circuit Z consists of AND and OR gates only, the output of Z is 1 (see the observation above). On the other hand, if Z and one of X and Y are OR circuits, then at least one of the inputs into Z is a 1 and the output of Z is also a 1, this establishing the correctness of the circuit.

We now compute the number of gates. This is given by the recurrence $N(k)$ as follows.

$$N(k) = \begin{cases} 3 & k = 1 \\ 3N(k/2) & k = 2^i, i > 0 \end{cases}$$

On solving the recurrence by simple iteration, or using the Master Theorem, we get $N(k) = 3k^{\lg 3}$.

Note: You may have wondered what is the size of an optimal k-OR circuit? This problem is still open! To the best of my knowledge, the best lower bound for the problem currently known is $\Omega(k \lg k/(\lg \lg k))$; so there is quite a gap. Interested readers may look up the following paper for more information on the problem.

D. Kleitman, T. Leighton, and Y. Ma "On the design of Boolean circuits that contain partially unreliable gates", *Journal of Computer and System Sciences*, 55(3):385-401, December 1997.