Some practice problems

CS218, Winter 2020

Analysis of iterative algorithms

• Give a tight bound on the number of Hello’s produced as a function of $n$

Algorithm LOOP2 ($n : integer$)
   for $j \leftarrow 1$ to $n^2 \log n$ do
      for $i \leftarrow 1$ to $j$ do
         print “Hello”
Analysis of iterative algorithms

• Give a tight bound on the number of Hello’s produced as a function of \( n \)

\[
\text{Algorithm LOOP1} \ (n : \text{integer}) \\
\text{for } i \leftarrow 1 \text{ to } n \log^2 n \text{ do} \\
\text{for } j \leftarrow 1 \text{ to } j \text{ do} \\
\text{print } \text{“Hello”}
\]

• Answer: \( \Theta(n^4 \log^2 n) \)

Analysis of iterative algorithms

• Give a tight bound on the number of Hi’s produced as a function of \( n \)

\[
\text{Algorithm LOOP2} \ (n : \text{integer}) \\
\text{for } j \leftarrow 1 \text{ to } n^2 \log n \text{ do} \\
\text{for } i \leftarrow 1 \text{ to } j \text{ do} \\
\text{print } \text{“Hello”}
\]

• Answer: \( \Theta(n^4 \log^2 n) \)
Analysis of iterative algorithms

• Give a tight bound on the number of Hi’s produced as a function of $n$

Algorithm $\text{LOOP1} (n: \text{integer})$

for $i \leftarrow 1$ to $n \log^2 n$ do
   $j \leftarrow i$
   while $j \leq n$ do
      print “Hi”
      $j \leftarrow j + 1$

• Answer: $\Theta(n^2)$

Analysis (recurrence relation)

Solve using the Master Theorem

$$T(n) = \begin{cases} 
1 & n = 1 \\
7T\left(\frac{n}{2}\right) + n^2 & n > 1
\end{cases}$$

Solution: The first case applies.

$T(n) \in \Theta\left(n^{\log_2 7}\right)$
Analysis (recurrence relation)

Solve using the Master Theorem

\[ T(n) = \begin{cases} 
1 & n = 1 \\
9T\left(\frac{n}{3}\right) + n^2 \log n & n > 1 
\end{cases} \]

**Solution:** The second case applies \((k = 1)\).

\( T(n) \in \Theta\left(n^2 \log^2 n\right) \)

Analysis (recurrence relation)

Solve using the Master Theorem

\[ T(n) = \begin{cases} 
1 & n = 1 \\
T\left(\frac{n}{2}\right) + n \log n & n > 1 
\end{cases} \]

**Solution:** The third case applies.

\( T(n) \in \Theta\left(n \log n\right) \)
Lower bounds

The authors of a paper that you are asked to review claim to have designed a new data structure for priority queues that supports both the operations \textsc{Insert} and \textsc{Extract-Minimum} in $O(1)$ worst-case time. Should you reject the paper? Why? You cannot make any assumption on the functionality of the data structure.

Amortized analysis

We want to use an unsorted array to support the following two operations for a set $S$ of integers:

- \textsc{Insert}(S, x) inserts integer $x$ into set $S$
- \textsc{Delete-Larger-Half}(S) deletes the largest $\lfloor |S|/2 \rfloor$ integers from $S$ (where $|S|$ is the current size of $S$)

Explain how to implement these operations on an unsorted array so that any sequence of $m$ operations runs in $O(m)$ time (or $O(1)$-time amortized, per operation). For simplicity, you can assume that the array is large enough so that we do not have to deal with reallocation when it gets full.
Divide & Conquer

Divide & Conquer ("black box")

Given an unsorted array $A$ of $n$ distinct floating point numbers we want to print the smallest $\lfloor \sqrt{n} \rfloor$ numbers of $A$ in sorted order. For instance given $A = \{3.1, 4.2, 1.013, 2.12, 5.36, 6.12, 0.15, 8.2, 9.1\}$ containing 9 numbers, the algorithm is supposed to print $0.15, 1.013, 2.12$ in sorted order. Give a $O(n)$-time algorithm for this problem. Hint: Use linear-time SELECT (as a black box) to solve this problem.
Divide & Conquer (design)

You are given two sorted arrays of size $m$ and $n$, respectively. Give a $O(\log k)$ time algorithm for computing the $k$-th smallest element in the union of the two lists. Note: Observe that the $k$-th smallest element in the union of the arrays $a[1...n]$ and $b[1...n]$ has to be contained in $a[1...k]$ or $b[1...k]$.

Divide & Conquer (design)

Suppose you have $k$ sorted arrays, each with $n$ elements, and you want to combine them into a single sorted array of $kn$ elements. Describe a divide-and-conquer algorithm that takes $O(kn \log k)$ time. Explain carefully why your algorithm takes $O(kn \log k)$ time.