CS4311 Design and Analysis of Algorithms

Homework 1

Due: 11:10 am, March 13, 2008 (before class)

1. (15%) Give asymptotic upper bound for $T(n)$ in each of the following recurrence. Make your bounds as tight as possible.

   (a) $T(n) = 9\ T(n/2) + n^3$

   (b) $T(n) = 7\ T(n/2) + n^3$

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

   (d) $T(n) = 0.5\ T(n/2) + n$

   (e) $T(n) = 3\ T(n/3) + n/3$

2. (15%) Using the definitions of O-notation and ω-notation, show that:

   $$\text{if } f(n) \in \omega(g(n)), \text{ then } f(n) \notin O(g(n)).$$

3. (15%) Given an input list of n numbers, recall that `Mergesort` first divides the list into two parts, then sorts each part recursively, and finally merges the two sorted parts together. The running time of `Mergesort` is $\Theta(n \log n)$.

   Now suppose that we divide the input list into three parts, sort each part recursively, and finally merge the three sorted parts together (how?). Will this new algorithm run faster or slower in the asymptotic sense? Why?

4. Consider the following code segment:

```
for (i = 1; i <= n; i++) {
   for (j = 1; j <= n; j += i)
       x = x + 1;
}
```

   (15%) Analyze the running time of the code in terms of n. (Use Θ-notation.)

5. Let A be a sequence of numbers. Define `Greedy-Pick` to be a function that extracts an increasing sequence from A, using the following method:

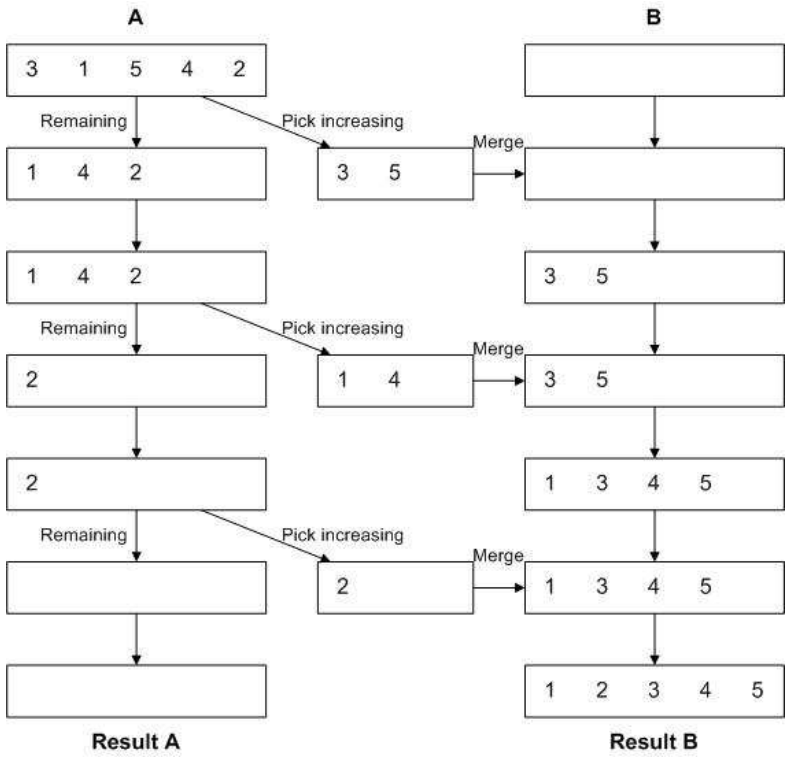
   1. Create an empty list L;
   2. Scan the first element in A, say x.
   3. Remove x from A, and append x at the end of L;
   4. **while** (there is unscanned element in A)
   5. { Scan the next element in A, say y;
   6. **if** (y is greater than the last element in L)
   7. { Remove y from A, and append y at the end of L; }
   8. }
   9. **return** L;

For example, suppose that $A = (1, 4, 3, 5, 2, 6)$ originally. After `Greedy-Pick` on A, we will pick a sequence $(1, 3, 5, 6)$, and A becomes $(4, 2)$.

Next, consider the following sorting algorithm based on `Greedy-Pick`:

1. Create two empty lists A and B;
2. Put the original input sequence into A;
3. **while** (A is not empty)
4. { `Greedy-Pick` a sequence from A, called L;
5. Merge L into B;
6. }
7. **return** B;

The figure below shows a sample run of the above algorithm for the input sequence $(3, 1, 5, 4, 2)$.



(15%) Show that the above sorting algorithm is correct (i.e., given any input sequence, the algorithm will output the sequence in sorted order). Describe a *worst-case input* such that the above algorithm will run in $\Theta(n^2)$ time, and show your analysis.

6. (a) (15%) Your friend, John, is holding an integer array $A[1..n]$, and he tells you that the array is sorted, and contains all the integers from 0 to n except one. John challenges you to find the missing integer. He allows you to ask him questions in the form:

"What is the jth bit of $A[i]$?"

and he is going to answer you honestly. So, you can vary the values of i and j in each question you ask him.

Design an efficient algorithm to find out the missing integer using $O(\log^2 n)$ questions.

(b) (15%) Now, John has changed the input, and the array A becomes unsorted. Design an algorithm using $O(n)$ questions to find out the missing integer.

7. (10%)[1] Cinderella's stepmother has newly bought n bolts and n nuts. The bolts and the nuts are of different sizes, each of them are from size 1 to size n. So, each bolt has exactly one nut just-fitting it.

   These bolts and nuts have the same appearance, so that we cannot distinguish a bolt from another bolt, or a nut from another nut, just by looking at them. Fortunately, we can compare a bolt with a nut by screwing them together, to see if the size of the bolt is too large, or just fitting, or too small, for the nut.

   Cinderella wants to join the ball held by Prince Charming. But now, her wicked stepmother has mixed the bolts, and mixed the nuts, and tells her that she can go to the ball unless she can find the largest bolt and the largest nut in time. An obvious way is to compare each bolt with each nut, but that will take $\Theta(n^2)$ comparisons (which is too long). Can you help Cinderella to find an algorithm requiring only $o(n^2)$ comparisons? For instance, $\Theta(n)$ comparisons?

[1] Q7 is a bonus question. Total mark is calculated by: (Sum of Q1 to Q6) $\times$ (100%+ Q7).