

CSE 361 Complexity of Sequential and Parallel Algorithms

Spring 2008 Exam I – Solutions

1. Consider the following algorithm:

Repeat

Run the algorithm \mathcal{A} (starting from the beginning) for $2T_n$ time steps;

If the answer for π is ready, output this and quit;

forever

Let a *basic step* refer to running the algorithm \mathcal{A} for $2T_n$ time steps. Using Markov's inequality, probability that any basic step does not produce the answer for π is $\leq \frac{1}{2}$. This means that the probability that the above algorithm takes more than $\alpha \log n$ basic steps is $\leq (1/2)^{\alpha \log n} = n^{-\alpha}$. In other words, the run time of the above algorithm is $\tilde{O}(T_n \log n)$.

2. Keep two 2-3 trees N and S . In N store all the records with the name as the key for each record and in S store all the records with the social security number as the key for each record. To process $\text{Find_Name}(SSN)$, we search for a record whose key is SSN in the tree S . The name in this record will be output. The run time is $O(\log n)$. We process $\text{Find_SSN}(Name)$ in a similar manner.

3. To begin with create n singleton sets, where each set has a person. At any given time a set represents a set of persons that are related to each other.

For each edge (i, j) , do a $\text{Collapsing_Find}(i) = A$ and $\text{Collapsing_Find}(j) = B$. If the resulting sets are different, do $\text{Weighted_Union}(A, B)$. After processing all the edges in this fashion, do $\text{Collapsing_Find}(p_1)$ and $\text{Collapsing_Find}(p_2)$. If these two are the same, output that p_1 and p_2 are related. Otherwise output that they are not related.

In the worst case, if there are m edges, we perform $(2m + 1)$ Collapsing_Find operations and n Weighted_Union operations.

Applying the theorem of Tarjan and van Leeuwen, the total run time is $O((m + n)\alpha(m))$.

4. Recurrence relations for the run times of the two algorithms are:

$$T_A(n) = 5T_A\left(\frac{n}{3}\right) + \Theta(n^2) \text{ \& } T_B(n) = 10T_B\left(\frac{n}{4}\right) + \Theta(n^{1.8}).$$

These solve to $T_A(n) = \Theta(n^2)$ and $T_B(n) = \Theta(n^{1.8})$ (using the Master Theorem)

Hence, B is preferred.

5. Sort $a[]$ in time $O(n \log n)$. Then run the following algorithm:

ThreeSum(x)

(1) for $(i = 1; i \leq n; i++)$, $(j = 1; j \leq n; j++)$ do

(2) $k = \text{BinarySearch}(x - a[i] - a[j], 1, n)$;

(3) if k is nonzero, return (k, i, j) ;

(4) print "not found" and exit;

Since binary search takes $O(\log n)$ time and there are n^2 invocations of binary search, the run time of the above algorithm is $O(n^2 \log n)$.

6. One way of sorting the given input sequence X is as follows: Partition X into X_1, X_2, \dots, X_m where $m = n/d$. X_1 is the sequence of the first d elements of X , X_2 is the sequence of the next d elements of X , and so on. Sort each X_i ($1 \leq i \leq m$) using merge sort, for example. This will take a total of $O(d \log d \times m = n \log d)$ time. Now merge X_1 with X_2 ; merge X_2 and X_3 ; and so on. This will take a total of $O(n)$ time. At the end of this X will be in sorted order. The total run time is $O(n \log d)$.