CS545 — Spring 2006
Brief Class Notes

August 4, 2006 We discussed augmented data structures — see the slides. We expanded this material and also discussed *interval trees.* This structures accepts as an input a set of intervals $S = \{(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)\}$, where $x_i < y_i$. After preprocessing this structures supports the following operations: Given a query point q, it reports all intervals of S that contain q in time $O(k + \log n)$, where k is their number. The constructions of the data structures is as follows:

Assume first that the subset $S' \subseteq S$ of intervals has the property that they all contain some point m. In this case we create two data structures for S', that are used to answer the same type of queries. The first, $L(S')$ is used to answer the query when the query point q is to the **left** of m. Let $\}x'_1, x'_2, , x'_{|S'|}\}$ denote the endpoints of the intervals of S', sorted from left to right. $L(S')$ contains these endpoints, sorted from left to right. To answer a query, we check whether $x'_1 \geq q$. If not, then since x'_1 is the leftmost point, q is outside all intervals of S', and we are done. If $x'_1 \geq q$ then, since q is to the left of m, it must be that q lies in the interval of S' associated with x'_1. We report that q is contained in this interval, and continue to check x'_2.

The list for $R(S')$ is symmetric, but with the right endpoints of S'. It is used when q is to the right of m.

Note that the time for answer a query is proportional to the number of reported intervals, since excluding the last checked interval, each interval that is checked contained q.

Finally, if there is no points that is contained in all intervals of S. we do the following: We find the point $m(S)$, the median of all endpoints of intervals of S. We split S into 3 disjoint subsets:

S_L — this set of intervals of S fully to the left of m.

S_R — this set of intervals of S containing m.

S_R — this set of intervals of S fully to the right of m.

We construct a tree binary tree T as follows. The root v of T is associated (that is, containing pointers to) $m(S)$, and the two structures $L(S_L)$ and $R(S_R)$.

We associated the left child of v with an interval tree constructed recursively for S_L. Similarly, we associated the right child of v with an interval tree constructed recursively for S_L.

September 4, 2006 We reviewed common techniques for generating hash functions. We introduced universal families of functions, and construct one such family. We saw the usage of these families to generate a perfect hash function. This material is covered very nicely in the textbook, and in the slides that you can find in the course webpage.

Sep 11, 2006 We started studying Amortized analysis. We consider the binary counter examples. In this example a binary array represents a number, and we presented pseudo-code to support the `Inc` operations, increasing the value of this number by 1. The time required for this operation is $\Theta(k+1)$ where k is the length of the run of '1' in the counter that need to be modified for this operations Let m be the size of the counter, and assume that n `Inc` operations are executed. Clearly $O(nm)$ is a bound on the time needed to execute all n operations. We presented a directed proof that the time it actually takes is only $O(n+m)$. Next we will use this example to see how other techniques of amortize analysis can be used to obtain the same proof. See chapter Chapter 17 (page 405) of the textbook.

9/13/06 We gave more proofs for the bounds on the time analysis needed for a sequence of insertion in the binary counter. We also discussed the problem of implementing a queue using two stacks. Our discussions in these two problems followed the textbook.

We continued applications of amortized analysis for dynamizing data structures. As an important applications, we introduced *Voronoi diagram*, for a set $S = \{s_1 \dots s_n\}$ of sites in the plane, which (when stored in an appropriate point location data structure can support the following operations: Given a query point q, find the nearest site of S to

q. We saw how we decompose the problem so that answering a query takes $O(\log^2 n)$, and moreover, we can add sites to S so that adding each point takes amortized time $O(\log^2 n)$. See the slides about this topic.

9/18/06 We continued with the application of dynamizing the nearest site Voronoi diagram. Then we presented another usage of amortized analysis, namely increasing the size of a table as elements are added to the table. We saw that even though we need to copy a large set of elements from time to time, the amortized time per insertion is still $O(1)$. The slides is a good reference.

9/27/06 We discussed spaly trees, (from a different textbook), and see how to use amortize analysis to analyze their behaviors.

10/4/06 We studied Binomial and Fibonacci heaps. The slides contains almost all the materials relevant for these topics. The only part that was presented on the whiteboard was the use of potential functions for amortize analysis. This topic is covered properly in the textbook (CLRS).