212 Test 2 Review Sheet

Cheat sheet: you will be permitted to write (hand-write or type, not photocopy, print PowerPoints, etc.) and bring a single, two-sided 8.5x11” sheet of notes. Coverage: everything in chapters 3, 4, 5, and 6 that we covered and that wasn’t covered on Test 1 ((2,4)-trees through DFS, BFS, and applications), including readings, lectures and homework.

In general, you should know algorithms and runtimes that we studied in this section. You should be able to simulate the algorithm on example data. You should also know the algorithm design techniques we studied – greedy, divide-and-conquer, and dynamic programming. You should know definitions and terminology we learned, such as that pertaining to graphs and to parallel algorithms.

Concerning the analysis of recursive functions, you should be able to look at a recursive function and write down a recurrence relation for its runtime using the Master Theorem, when it applies. You will not be asked to solve recurrences. (Be sure you know the Master Theorem or have it written on your cheat sheet.)

Topics

Chapter 3: Trees
- (2,4)-trees—properties, insertion
- Red-black trees: definition, concepts, properties

Chapter 4: Sorting
- Merge sort
- Quicksort (variations: randomized; cut-off to insertion sort)
- Lower bound on comparison sorting
- Radix sort
- Parallel algorithm terms: PRAM model, SIMD/MIMD, CREW/EREW, Speedup, efficiency
- Parallel algorithms: parallel max, merge sort (Odd/Even Merge)

Chapter 5: Design techniques
- Greedy
  - What’s the general technique? When does it apply?
  - Greedy algorithm for making change
  - Fractional knapsack
  - Task scheduling using a minimal number of machines
- Divide and conquer
  - Examples: Binary search, Merge sort, Min&Max
  - Recursive algorithm analysis:
    - Given a recursive function, write down a recurrence relation for its runtime
    - Use the master theorem to solve a recurrence relation
- Dynamic programming
  - Describe the technique. Bottom-up and memoization as speed-up techniques.
  - When does dynamic programming apply?
  - Computing Fibonacci numbers as a DP problem
  - DP algorithm for making change
  - 0/1 knapsack (e.g. given the following knapsack problem, show the subproblem solutions that would be needed to solve the original problem)
  - Longest common subsequence

Chapter 6. Graphs
- Definitions and terminology
- Adjacency list and adjacency matrix representations
- Depth-first search; applications: cycle finding, connected components, reachability, DFS spanning tree
- Breadth-first search; applications: shortest path (number of edges)
Sample Questions

1. Show a (2,4)-tree after inserting 1, 2, and 3, then after inserting 4, 5, and 6, and again after inserting 7, 8, and 9.

2. What is the definition of a Red-Black tree?

3. Given the array of numbers 3 5 1 6 8 4 2 7 9 0, explain how Quicksort would sort the list if it were to pick 6 as the pivot. What steps would be involved? What subproblem solutions would it have to compute recursively? How would the answer differ in the quicksort variant that uses a cut-off to insertion sort for lists of size 4 or less?

4. Show how the following list of binary numbers would be sorted by radix-2 sort, with a cutoff to insertion sort on the last 2 bits of the numbers. (Show the list after each pass of radix sort and after the insertion sort pass.) 1010 1101 1011 0101 0100 0110 1110 1001

5. In the divide-and-conquer Min&Max algorithm we studies, what would the last two comparisons it would make in finding the min and max of the list 3 5 1 6 8 4 2 7?

6. Give a recurrence relation for the runtime of the following functions.

   1. int fun1(int n) {
      if (n<=1) return 1;
      return fun1(n/2) * fun1(n/2) + 1; }

   2. int fun2(int n) {
      if (n<=1) return 1;
      int sum=0;
      for (int i=1; i<=n; i++)
         sum *= fun2(n-2);
      return sum;
   }

7. Use the Master Theorem to write down asymptotic solutions to the following recurrence relations:
   a. T(n) = 2T(n/2) + c n^0.9
   b. T(n) = 2T(n/2) + c n
   c. T(n) = 2T(n/2) + c n^1.1

8. Show the DFS traversal of the following graph, if the algorithm always chooses the lowest-numbered vertex when it has a choice.

9. For a graph with v vertices and e edges, what is the runtime of a BFS traversal?

10. Given a collection computers, routers, and network cables, together with information about how they are plugged together, explain how you might determine whether every computer can talk to every other computer.

11. Describe an algorithm for finding the connected components of an undirected graph.

12. Show how odd-even merge would sort the following list of numbers in parallel. What’s the runtime of the algorithm? What is its efficiency? Is it EREW, CREW, or CRCW?

Selected answers:
4. 1010 1011 1001 1101 0101 0100 0110 1110; 0101 0100 0110 1011 1001 1001 1100; 0100 0101 0110 1001 1010 1011 1100
5. 1&2, 6&8.
6. T(1)=1; T(n)=2T(n/2)+1, n>1. T(1)=1; T(n)=nT(n-2)+n, n>1
7. O(n), O(n log n), O(n^1/2)
10. Treat the computers and routers as nodes and the cables as edges in a graph. Run DFS to determine whether the graph is connected.