**TEST II - Review**

(Test is on Monday, November 8, 2021)

CS 4120/5120 Fall, 2021

(Disclaimer: This is not an all-inclusive list; rather it is intended as a guide for your study.)

* **Read Chapters 6, 7, 8, 12, 15, and 16 (and also the Master Theorem from Chapter 4, which did not get much attention on the previous exam). Do not forget about Chapters 1, 2, 3, and (the rest of) 4 of your textbook, since our material builds on them.**
* **CS 5120 students should pay extra attention to dynamic programming, priority queues (not covered extensively in class), and view the topics more cumulatively. (That does NOT mean CS 4120 students may simply ignore these topics!)**
* **Notes:** Definition of terms discussed in class
  + Heap - The (binary) heap data structure is an array object that we can view as a nearly complete binary tree as shown in Figure 6.1. Each node of the tree corresponds to an element of the array. The tree is filled on all levels except possibly the lowest, which is filled from the left up to a point.
  + Pivot - The pivot or pivot element is the element of a matrix, or an array, which is selected first by an algorithm (e.g. Gaussian elimination, simplex algorithm, etc.), to do certain calculations. ... Overall, pivoting adds more operations to the computational cost of an algorithm.
  + principle of optimality - Principle of optimality states while solving the problem of optimization one has to solve sub-problems, solution of sub-problem will be the part of optimization problem" , if problem can be solved by optimal sub problem means it consist optimal substructure
  + greedy choice property - We can make whatever choice seems best at the moment and then solve the subproblems that arise later. The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the subproblem.
  + Divide-and-Conquer Strategy - an algorithm design paradigm. A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly.
  + Dynamic Programming - a mathematical optimization method and a computer programming method. ... Likewise, in computer science, if a problem can be solved optimally by breaking it into sub-problems and then recursively finding the optimal solutions to the sub-problems, then it is said to have optimal substructure.
  + Greedy Algorithms - an algorithmic strategy that makes the best optimal choice at each small stage with the goal of this eventually leading to a globally optimum solution. This means that the algorithm picks the best solution at the moment without regard for consequences.
* **Textbook:**
  + **Chapter 4**: Sections 4.5 (pages 93-96 )
    - T (n)= aT(n/b) + *f*(n)
    - <https://www.nayuki.io/page/master-theorem-solver-javascript>
    - <https://www.wolframalpha.com/input/?i=T%5Bn%5D%3D%3D2T%5Bn%2F4%5D%2B1>
  + **Sorting and Order Statistics**: (pages 147-150)
    - Insertion sort takes Θ(n^2) time in the worst case. Because its inner loops are tight, however, it is a fast in-place sorting algorithm for small input sizes. (Recall that a sorting algorithm sorts in place if only a constant number of elements of the input array are ever stored outside the array.)
    - Table

      Description automatically generatedMerge sort has a better asymptotic running time, Θ(n log n), but the MERGE procedure it uses does not operate in place.
  + **Chapter 6**: Sections 6.1-6.4 (pages 151-161) [**CS 5120: Also** 162-164.]
    - [Be able to do the heapsort problem – homework 3, problem 4 – independently.]
    - A screenshot of a computer

      Description automatically generated with medium confidence<https://www.chegg.com/homework-help/questions-and-answers/4-show-heapsort-algorithm-works-following-numbers-showing-binary-tree-view-step-build-max--q86142949>
    - --
    - Text

      Description automatically generated--
    - Text

      Description automatically generated--
    - Text

      Description automatically generated--
  + **Chapter 7**: Sections 7.1-7.4 (pages 170-184) [**CS 5120:** Special emphasis: 174-178.] **[**Pay special attention to homework 3, problem 1.]
    - <https://www.chegg.com/homework-help/questions-and-answers/1-describe-could-create-efficient-algorithm-based-quicksort-following-problem-simple-way-a-q86467510>
    - Text

      Description automatically generated-
    - Text

      Description automatically generated-
    - Text, letter

      Description automatically generated-
    - Text

      Description automatically generated-

A screenshot of a computer

Description automatically generated with medium confidence

* + **Chapter 8**: Sections 8.1-8.2 (pages 191-196) [Understand Theorem 8.1]
    - We can view comparison sorts abstractly in terms of decision trees. A decision tree is a full binary tree that represents the comparisons between elements that are performed by a particular sorting algorithm operating on an input of a given size. Control, data movement, and all other aspects of the algorithm are ignored. Figure 8.1 shows the decision tree corresponding to the insertion sort algorithm from Section 2.1 operating on an input sequence of three elements.
    - Text, letter

      Description automatically generatedThe length of the longest simple path from the root of a decision tree to any of its reachable leaves represents the worst-case number of comparisons that the corresponding sorting algorithm performs. Consequently, the worst-case number of comparisons for a given comparison sort algorithm equals the height of its decision tree. A lower bound on the heights of all decision trees in which each permutation appears as a reachable leaf is therefore a lower bound on the running time of any comparison sort algorithm.
    - - Counting sort assumes that each of the n input elements is an integer in the range 0 to k, for some integer k. When k = O(n), the sort runs in Θ(n) time. Counting sort determines, for each input element x, the number of elements less than x. It uses this information to place element x directly into its position in the output array. For example, if 17 elements are less than x, then x belongs in output position 18. We must modify this scheme slightly to handle the situation in which several elements have the same value, since we do not want to put them all in the same position.
  + **Chapter 12**: Sections 12.1-12.3 (pages 286-299)
    - - Basic operations on a binary search tree take time proportional to the height of the tree. For a complete binary tree with n nodes, such operations run in Θ (lg n) worst-case time. If the tree is a linear chain of n nodes, however, the same operations take Θ(n) worst-case time. We shall see in Section 12.4 that the expected height of a randomly built binary search tree is O(lg n), so that basic dynamic-set operations on such a tree take Θ(lg n) time on average.
    - - The procedure begins its search at the root and traces a simple path downward in the tree, as shown in Figure 12.2. For each node x it encounters, it compares the key k with x:key. If the two keys are equal, the search terminates. If k is smaller than x:key, the search continues in the left subtree of x, since the binary-search-tree property implies that k could not be stored in the right subtree. Symmetrically, if k is larger than x:key, the search continues in the right subtree.
    - Text

      Description automatically generatedSearching We use the following procedure to search for a node with a given key in a binary search tree. Given a pointer to the root of the tree and a key k, TREE-SEARCH returns a pointer to a node with key k if one exists; otherwise, it returns NIL.
    - Text, letter

      Description automatically generated
  + **Chapter 15**: Sections 15.1-15.3 (pages 357-389)
    - -Dynamic programming, like the divide-and-conquer method, solves problems by combining the solutions to subproblems. (“Programming” in this context refers to a tabular method, not to writing computer code.) As we saw in Chapters 2 and 4, divide-and-conquer algorithms partition the problem into disjoint subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem. In contrast, dynamic programming applies when the subproblems overlap—that is, when subproblems share sub subproblems. In this context, a divide-and-conquer algorithm does more work than necessary, repeatedly solving the common subsubproblems. A dynamic-programming algorithm solves each subsubproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time it solves each subsubproblem. We typically apply dynamic programming to optimization problems. Such problems can have many possible solutions. Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value. We call such a solution an optimal solution to the problem, as opposed to the optimal solution, since there may be several solutions that achieve the optimal value.
    - Graphical user interface, text, application, email

      Description automatically generated-
    - Why is CUT-ROD so inefficient? The problem is that CUT-ROD calls itself recursively over and over again with the same parameter values
    - Diagram, schematic

      Description automatically generated-
  + **Chapter 16**: Sections 16.1-16.3 (pages 414-433) [You may skip parts not covered in class.]
* Algorithms that are based on algorithm design techniques discussed in class (i.e. of Divide-and-Conquer, Dynamic Programming, and Greedy strategies):
  + Heap sort
  + Quick sort
  + Rod cutting
  + Matrix-chain Multiplication
  + Activity Selection
  + Huffman Code.
* Devise algorithms to solve variations of known problems
* Analyze the complexity of given algorithms

**Sample Questions:**

* 1. Compare Dynamic Programming: vs. Divide and Conquer or vs. Greedy algorithms.
  2. Identify when the three cases of the Master Theorem apply and be able to solve a recurrence relation, or two.
  3. Analyze the complexity of a given algorithm, and show the results using order notation.
  4. Using the idea of quicksort, give an efficient algorithm, without completely sorting the data, to identify the kth smallest (or kth largest) of n numbers. (This is called the “quickselect” algorithm and is based on an idea similar to the one used in homework 3, problem 1.)

4. Consider the following problem (supplied at test time). Decide if the Principle of Optimality (Optimal Substructure) applies. Explain why/why not.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. **Divide-and-Conquer**

(Merge sort, Quicksort, …)

Recurrence relations: (Where do they come from? Why are they important? How do you solve one?)

1. **Dynamic Programming**

The basic concept, the principle of optimality Recursive nature

Storing sub problem solutions, avoiding suboptimal paths (Rod cutting problem, Matrix-chain multiplication) Memoization

Bottom-up versions that avoid recursion

1. **Greedy Algorithms**

The main distinction with Dynamic Programming, and the greedy choice property Examples: Activity selection, Huffman Code

What kind of Knapsack problem(s) lends itself to a greedy approach? Which one(s) can be solved by dynamic programming?