CPSC-320 Intermediate Algorithm Design and Analysis Winter 2000

Practice Homework 6: Sample Final Exam Questions

The questions in this sample are representative of those that will be on the exam. The exam will be cumulative, covering all material in the class, so the questions on the midterm are also representative. You will have two hours in which to do the exam, which will have five questions. You may refer to your notes and text during the exam.

1. Show that the *Class Scheduling (CS)* problem, defined below, is *NP*-complete.

   A college wishes to offer evening courses and has n possible courses that can be offered. There are s students interested in taking a course and each student is interested in some subset of the n possible courses. To minimize costs, the college has a bound b on the number of courses it is willing to actually offer. Is there a way to choose b courses out of the n possibilities so that every student can take a course in which he or she is interested?

   Formally, an instance of the *CS* problem consists of

   - list of possible courses, $1, 2, \ldots, n$,
   - s subsets $C_1, C_2, \ldots, C_s$ of the numbers $\{1, 2, \ldots, n\}$, (the sets of courses of interest to each of the students) and
   - a bound b (the maximum number of classes actually offered).

   An instance is in the language *CS* if and only if there is a subset B of $\{1, 2, \ldots, n\}$, where the size of B is $\leq b$, such that $B \cap C_i$ is not empty for all i, $1 \leq i \leq s$.

   [You can use the fact that the problems discussed in class are *NP*-complete].

2. Give examples of decision problems in the following classes (no explanation necessary).

   (i) the class P of polynomial-time recognizable problems,

   (ii) the class of NP-complete problems (i.e. NPC),

   (iii) the class of problems in NP that are thought unlikely to be in NPC.

3. The Knuth-Morris-Pratt algorithm constructs a table $\pi[1, \ldots m]$ for pattern P of length m, where $\pi[q]$ is the length of the longest prefix of $P[1, ..., q]$ that is also a suffix of $P[1, ..., q]$. For the pattern $P = xxyxxyxy$ fill in the values in the table π:

	1	2	3	4	5	6	7	8
π								

4. The *random permutation* problem is: given an array $A[1, \ldots, n]$ with distinct entries, randomly permute the entries of A so that each permutation of the entries is equally likely.

   For example, if $n = 3$ and initially A contains the entries 2,7,5 in that order, then your algorithm should produce one of the following arrays, with each equally likely:

   2,7,5 2,5,7 5,2,7 5,7,2 7,2,5 7,5,2.

Your algorithm may use a function *rand(i,j)* which returns a random number k in the range $i \leq k \leq j$. Assume that this function takes constant time.

(a) Design an algorithm that solves this problem. What is the running time of your algorithm?

(b) Explain carefully why your algorithm produces a random permutation.

5. Consider the following algorithm, which finds both the max and the min elements of an array $A[i, \ldots, j]$, where the size, $j - i + 1$, is a power of 2. Let $T(n)$ be the *number of comparisons* performed by the algorithm on an input array of size n. Find a recurrence for $T(n)$ when n is a power of 2 and solve it as carefully as you can, finding the constants in the terms if possible.

```
function find-max-min(i,j)
   {returns a pair (Max, Min)}
   if i+1 = j then if A[i] > A[j], return (A[i], A[j])
                   else return (A[j], A[i])
   else k <-- (j+i-1)/2
      (MaxL, MinL) <-- find-max-min(i,k)
      (MaxR, MinR) <-- find-max-min(k+1,j)

   if MaxL > MaxR then Max <-- MaxL else Max <-- MaxR
   if MinL < MinR then Min <-- MinL else Min <-- MinR

   return (Max,Min)
```

Describe and analyze the divide and conquer algorithm that divides the array into three parts instead of two parts. You need only analyze the case when n is a power of 3 (in which case the problem divides nicely into three equal parts at every stage).