CPSC 320 Final Exam

July 28, 2006

Name: _____ Student ID: _____

Signature: _____

- You have 2.5 hours to write the 9 questions on this examination. A total of 43 marks are available.
- *Justify all of your answers.*
- You are allowed to bring in one double-sided letter size sheet of paper and nothing else.
- Keep your answers short. If you run out of space for a question, you have written too much.
- The number in the square brackets next to the question number is the # of marks allocated for that question. Use these to help determine how much time you should spend on each question.
- Use the back of the pages for your rough work.
- *Good luck!*

Question	Possible	Marks
1	12	
2	3	
3	5	
4	2	
5	5	
6	2	
7	10	
8	2	
9	2	
Total	43	

UNIVERSITY REGULATIONS:

- Each candidate should be prepared to produce, upon request, his/her library card.
- No candidate shall be permitted to enter the examination room after the expiration of one half hour or to leave during the first half hour of the examination.
- CAUTION: candidates guilty of any of the following, or similar, dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action.
    1. Having at the place or writing, or making use of, any books, papers or memoranda, electronic equipment, or other memory aid or communication devices, other than those authorized by the examiners.
    2. Speaking or communicating with other candidates.
    3. Purposely exposing written papers to the view of other candidates. The plea of accident or forgetfulness shall not be received.
- Candidates must not destroy or mutilate any examination material; must hand in all examination papers; and must not take any examination material from the examination room without permission of the invigilator.

1. [12] You are given an unlimited supply of coins with the values $1¢, 2¢, \ldots, k¢$ and are asked to count the number of unique ways that you can make s cents change. This question guides you to an efficient algorithm to solve this problem. Complete all parts, preferably in order (for your benefit).

   Using $1¢, 2¢, 3¢$ coins, there are three unique ways to make $3¢$ change: $(1¢, 1¢, 1¢)$, $(1¢, 2¢)$, and $(3¢)$. In particular, the combinations $(1¢, 2¢)$ and $(2¢, 1¢)$ are not unique.

   (a) [2] What are the seven unique ways of making $6¢$ change using $1¢, 2¢, 3¢$ coins?

   $(1¢, 1¢, 1¢, 1¢, 1¢, 1¢)$, $(1¢, 1¢, 1¢, 1¢, 2¢)$, $(1¢, 1¢, 2¢, 2¢)$, $(2¢, 2¢, 2¢)$, $(1¢, 1¢, 1¢, 3¢)$, $(1¢, 2¢, 3¢)$, and $(3¢, 3¢)$

   (b) [2] Let $A[i]$ be the number of unique ways to make i cents change using $1¢, 2¢$ coins. Write an expression for the number of unique ways to make s cents change using $1¢, 2¢, 3¢$ coins in terms of $A[i]$.

   $$\sum_{j=0}^{\lfloor s/3 \rfloor} A\left[s - 3j\right]$$

(c) [3] Fill in the body of a recursive divide-and-conquer algorithm that answers the problem, "How many unique ways are there to make s cents change using $1¢, 2¢, \ldots, k¢$ coins?". It make be as inefficient as you like, so long as it is correct. Simpler is better.

```
Algorithm RecChangeCount(s,k)
   if (k = 1) then return 1
   if (s = 0) then return 1

   return ∑_{j=0}^{⌊s/k⌋} RecChangeCount(s − jk, k − 1)
```

(d) [3] Write an iterative dynamic programming solution to this problem that fills in a table $A[0 \ldots s][1 \ldots k]$ so that the answer is in $A[s][k]$.

```
for i ← 1 to k do
   A[0][i] ← 1
for i ← 0 to s do
   A[s][1] ← 1

for k' ← 2 to k do
   for s' ← 1 to s do
      A[s'][k'] ← A[s'][k' − 1]
      for j ← 1 to ⌊s'/k'⌋ do
         A[s'][k'] ← A[s'][k'] + A[s' − jk']
```

(e) [2] Write an iterative dynamic programming solution to the problem that fills in a table $A[0 \ldots s]$ so that the answer is in $A[s]$. Use at most $\Theta(1)$ additional storage.

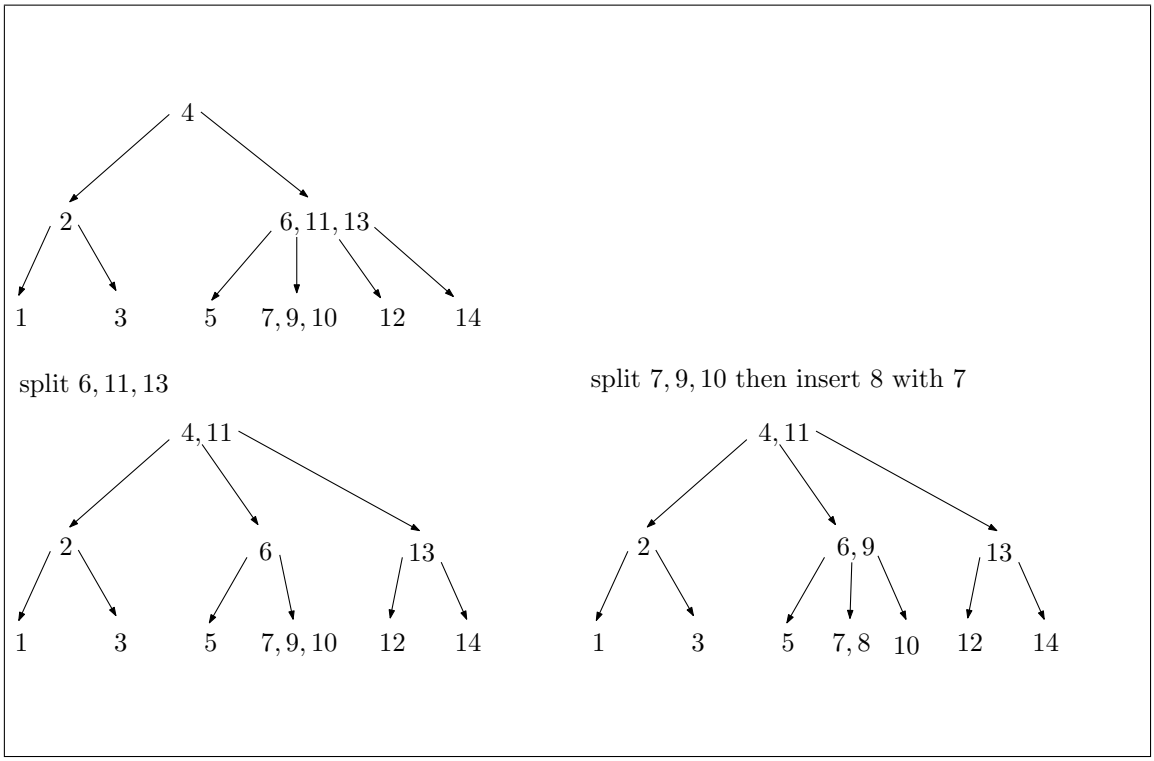
```
for i ← 0 to s do
    A[i] ← 1

for k' ← 2 to k do
    for s' ← s downto 0 do
        for j ← ⌊s'/k'⌋ downto 1 do
            A[s'] ← A[s'] + A[s' − jk']
```

2. [3] After computing the final grades, I want to sort the class list by decreasing grade first and by student ID in case of a tie. The list of possible grades is $A+$,A,$A-$,$B+$,B,$B-$,$C+$,C,$C-$, and D. What is the simplest, most efficient way to do this. You may use any of the algorithms that we covered in class.

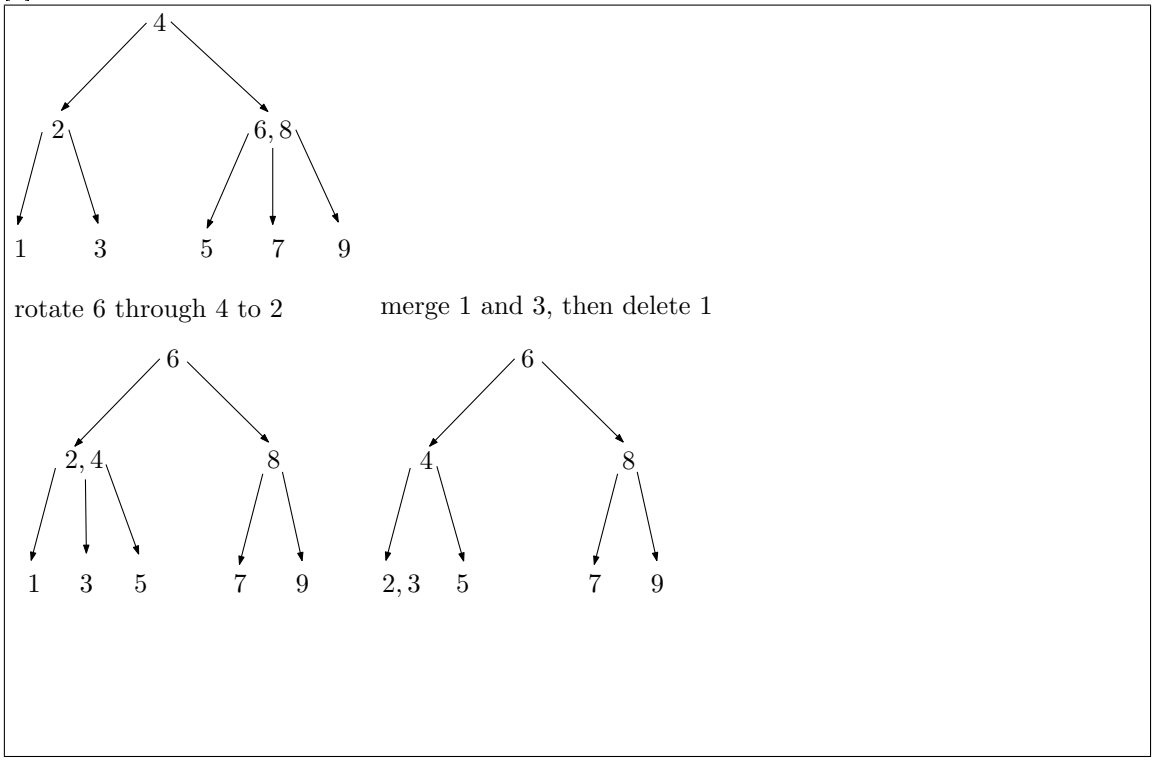
Mergesort by student ID and then bucket sort by grade. You could have radix sorted by student ID, but for all reasonable class sizes $2^{\text{digits of ID}} = 2^8 = 256$ is greater than the number of students in the class. So mergesort will be slightly faster.

3. [5] Perform the given operations on the given 2-3-4 tree using the algorithms from class (not the textbook). List the operations that you used (e.g. merge, split, and rotate), so that we can give you full marks. The correct result is not enough.

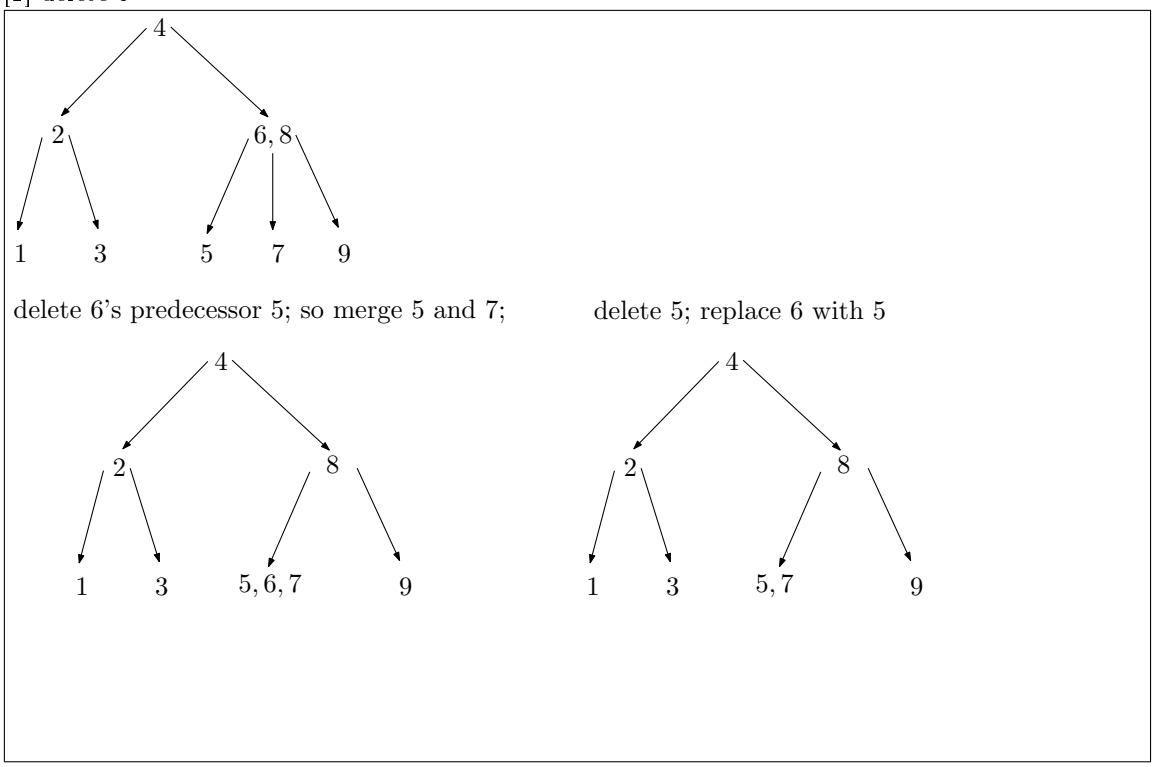
(a) [2] insert 8



(b) [2] delete 1



(c) [1] delete 6



4. [2] How would you modify the dynamic programming solution to the Maximum Weighted Activity selection problem to work with intervals of negative weight?

Eliminate all intervals of non-positive weights because we never want to select them. `FastDPMaxActivitySelect` needs no changes. All other versions do.

5. [5] If all edge weights in an undirected graph are $1, 2, \ldots, k$ where k is a small constant (say $k < 100$), what is the fastest that you can make Prim's run asymptotically? Justify your answer. Hint: ditch the heap.

Instead of using a heap to keep track of the minimum cost edge to use to extend the tree, we use an array (indexed by cost) of lists (representing vertices to be added with that cost).

To do a Q.deleteMin() we look at the lists in order from $1 \ldots k$ to find a non-empty list. As k is small and constant, this is a constant time operation. Using this data structure instead of a heap, the cost of adding and removing a vertex to the array of lists is constant.

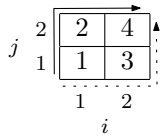
If we keep track of where a vertex is currently in a list (a bit of additional bookkeeping, but still constant time), the cost of updating the priority of a vertex is now $O(1)$. So the total cost of the algorithm (using the analysis from class) is now $O(|V| + |E|)$, which is asymptotically better than $O([|E| + |V|] \log |V|)$ using a heap.

6. [2] Show that $n \in O(n \log n)$.

$$\lim_{n \to \infty} \frac{n}{n \log n} = \lim_{n \to \infty} \frac{1}{\log n} = 0$$

So $n \in o(n \log n)$.

7. [10] You are given a $n \times n$ table $M[1 \ldots n][1 \ldots n]$ of real values, where $M[i][j]$ is the value in the i^{th} column and the j^{th} row (see below). We want to find paths from the bottom left corner $M[1][1]$ to the top right corner $M[n][n]$. Such paths move from cell to cell and are only allowed to travel to cells immediately above or to the left of the current cell.



The figure above shows the only two such paths in a 2×2 grid. The cost of a path is the sum of the cells that the path visits. For example, the cost of the solid path is $1 + 2 + 4$ and the cost of the dotted path is $1 + 3 + 4$. Given M, what is the cost of the cheapest such path?

(a) [2] One greedy algorithm is to always move to the cheapest cell immediately above or to the left. Give a 3×3 example where this strategy fails.

0	0	0
1	1	5
0	0	0

(b) [5] Give an iterative dynamic programming solution to this problem using $O\left(n^2\right)$ additional storage. Explain what the values of your table represent.

Let $C[i][j]$ be the minimum cost to get from $M[1][1]$ to $M[i][j]$. On such a path, we either go through $M[i-1][j]$ or $M[i][j-1]$ to get to $M[i][j]$ because we are only allowed to move left or up. If it was the latter, then the subpath from $M[1][1]$ to $M[i-1][j]$ must have been minimal cost because otherwise we could do better substituting a minimal cost path. Plus we can could use any minimal cost path from $M[1][1]$ to $M[i-1][j]$ and move left to get a minimal cost path from $M[1][1]$ to $M[i][j]$. This is the optimal substructure. A similar argument holds if we go through $M[i][j-1]$ on a cheapest path from $M[1][1]$ to $M[i][j]$.

So

$$C[i][j] = \begin{cases} M[1][1] & \text{if } i = 1, j = 1 \\ C[1][j-1] + M[1][j] & \text{if } i = 1, j \neq 1 \\ C[i-1][1] + M[i][1] & \text{if } i \neq 1, j = 1 \\ \min\{C[i-1][j], C[i][j-1]\} + M[i][j] & \text{otherwise} \end{cases}$$

To fill the table

```
for i ← 1 to n do
   for j ← 1 to n do
      C[i][j] ← expression above
```

(c) [3] We discussed several path finding algorithms in class which can be used to solve this problem. Choose the algorithm resulting in the asymptotically fastest solution, describe how you would use it, and give the asymptotic runtime of the solution in terms of n.

There are a couple of solutions to this problem. What follows is not the simplest, but it is the most obviously correct.

For each cell $M[i][j]$ we introduce to vertices $in_{i,j}$ and $out_{i,j}$. We introduce an edge between $in_{i,j}$ and $out_{i,j}$ with cost $M[i][j]$. Traversing this edge corresponds to visiting the cell $M[i][j]$. Then for each $out_{i,j}$ we add an edge to $in_{i+1,j}$ with cost 0, which allows us to move left in the matrix. Similarly, for each $out_{i,j}$ we add an edge to $in_{i,j+1}$ with weight 0, which allows us to move up. The shortest path in the matrix corresponds to a shortest path from $in_{1,1}$ to $out_{n,n}$ and vice versa.

Because the weight of an edge could have a negative value, we have to use Bellman-Ford. The number of vertices is $O\left(n^2\right)$ and the number of edges is $O\left(n^2\right)$, so the runtime is $O\left(n^2\right)$. However, the maximum number of vertices on such at path is $4n$, so it suffices to consider every edge $4n$ times for a runtime of $O\left(n^3\right)$ (that's a bonus, you're not expected to know that).

8. [2] Let $T(n) = 3T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n^2$ for $n > 2$ and $T(n) = 3$ otherwise. Use the Master Theorem to get an asymptotic bound on $T(n)$.

$\log_2 3 < \log_2 4 = 2$. So let $\epsilon = \log_2 4 - \log_2 3$. Then $f(n) = n^2 \in \Omega\left(n^{\log_2 3 + \epsilon}\right)$. So we try to apply case 3 of the master theorem.

Case 3 requires a regularity condition: $af\left(\frac{n}{b}\right) \leq \delta f(n)$ for some $\delta < 1$. $af\left(\frac{n}{b}\right) = 3\left(\frac{n}{2}\right)^2 = \frac{3}{4}n^2 = \frac{3}{4}f(n)$. So let $\delta = \frac{3}{4}$ to get the regularity condition. Hence $T(n) \in \Theta\left(n^2\right)$.

9. [2] Convert the SAT instance $(x_1 \vee x_2) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$ into a 3-colouring problem.