CPSC 320 Final Exam

July 27, 2007

Name: _____ Student ID: _____

Signature: _____

- You have 2 hours and 30 minutes on this examination. A total of 36 marks are available.
- *Justify all of your answers.*
- 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!*

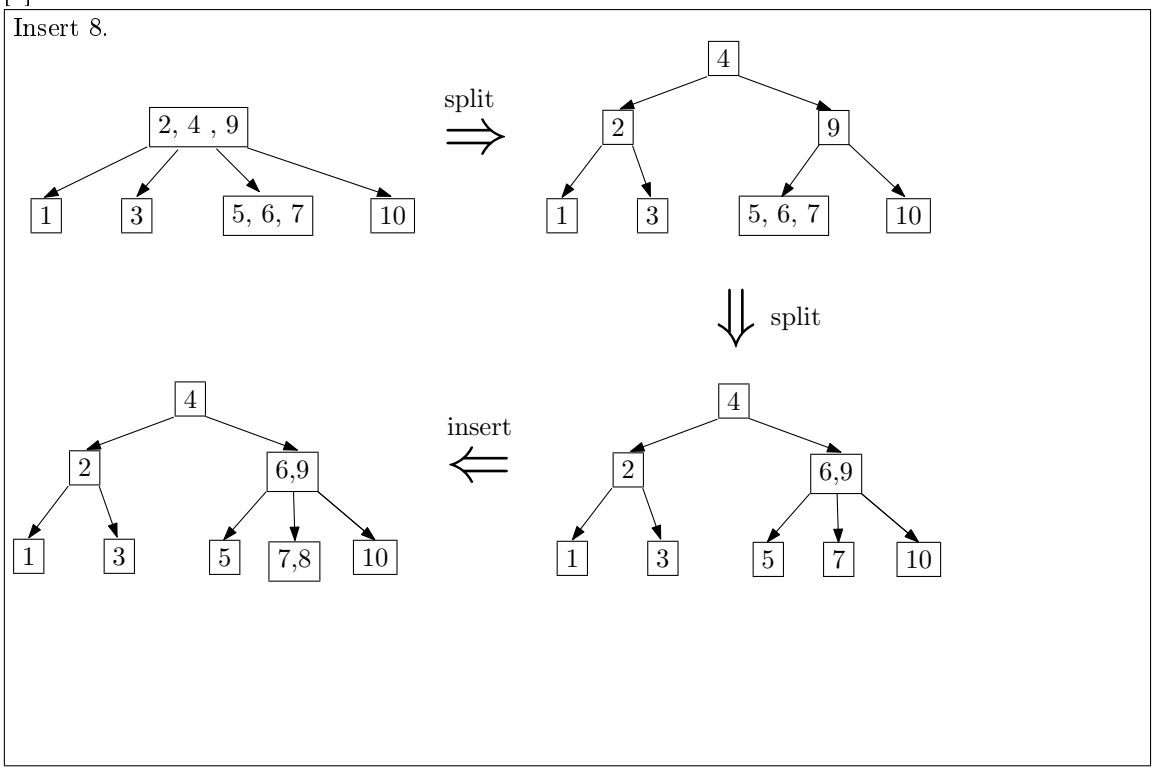
Question	Marks	Score
1	5	
2	5	
3	3	
4	6	
5	5	
6	4	
7	8	
Total	36	

UNIVERSITY REGULATIONS:

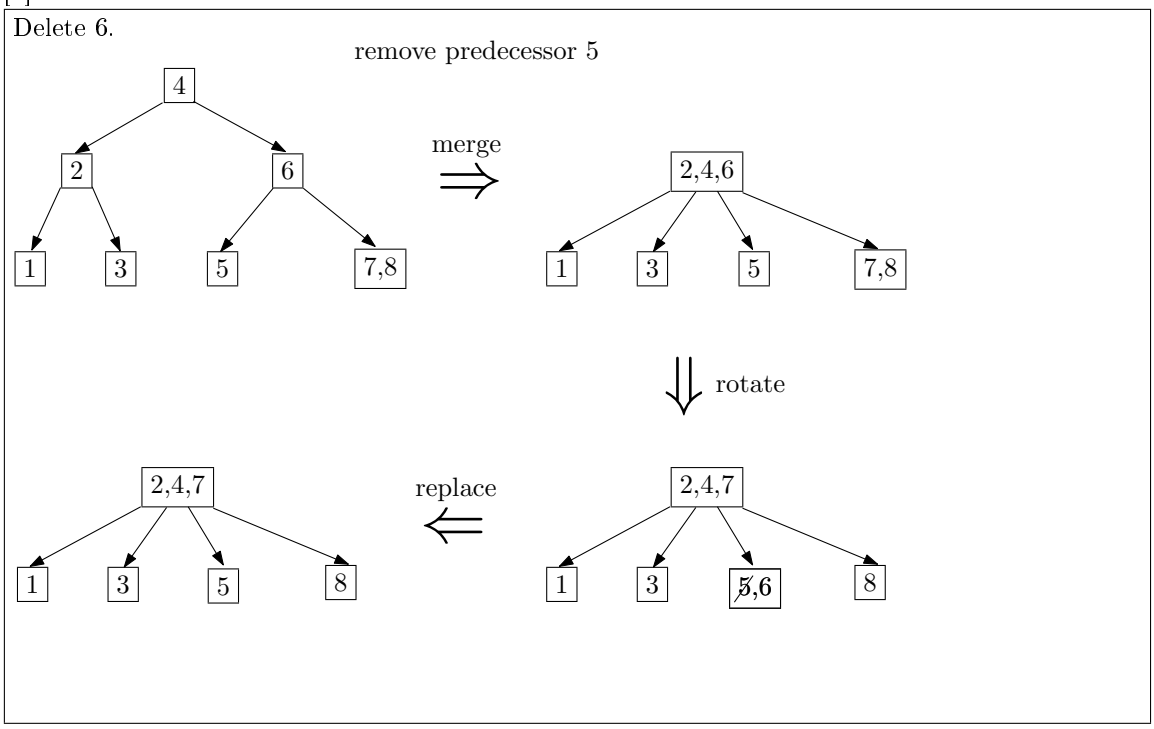
- Each candidate should be prepared to produce, upon request, his/her library card.
- No candidate shall be permitted to enter the examination room after the expiration of one half hour or to leave during the first half hour of the examination.
- CAUTION: Candidates guilty of any of the following, or similar, dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action.
    1. Having at the place of 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 the permission of an invigilator.

1. [5] Perform the listed operations on the given 2-3-4-Tree. You **must** use the algorithms discussed in class: *do not use the algorithms from your textbook*. By showing all of your work, we can give you partial credit, if you make a mistake. When in doubt, remember that your instructor is left-handed.

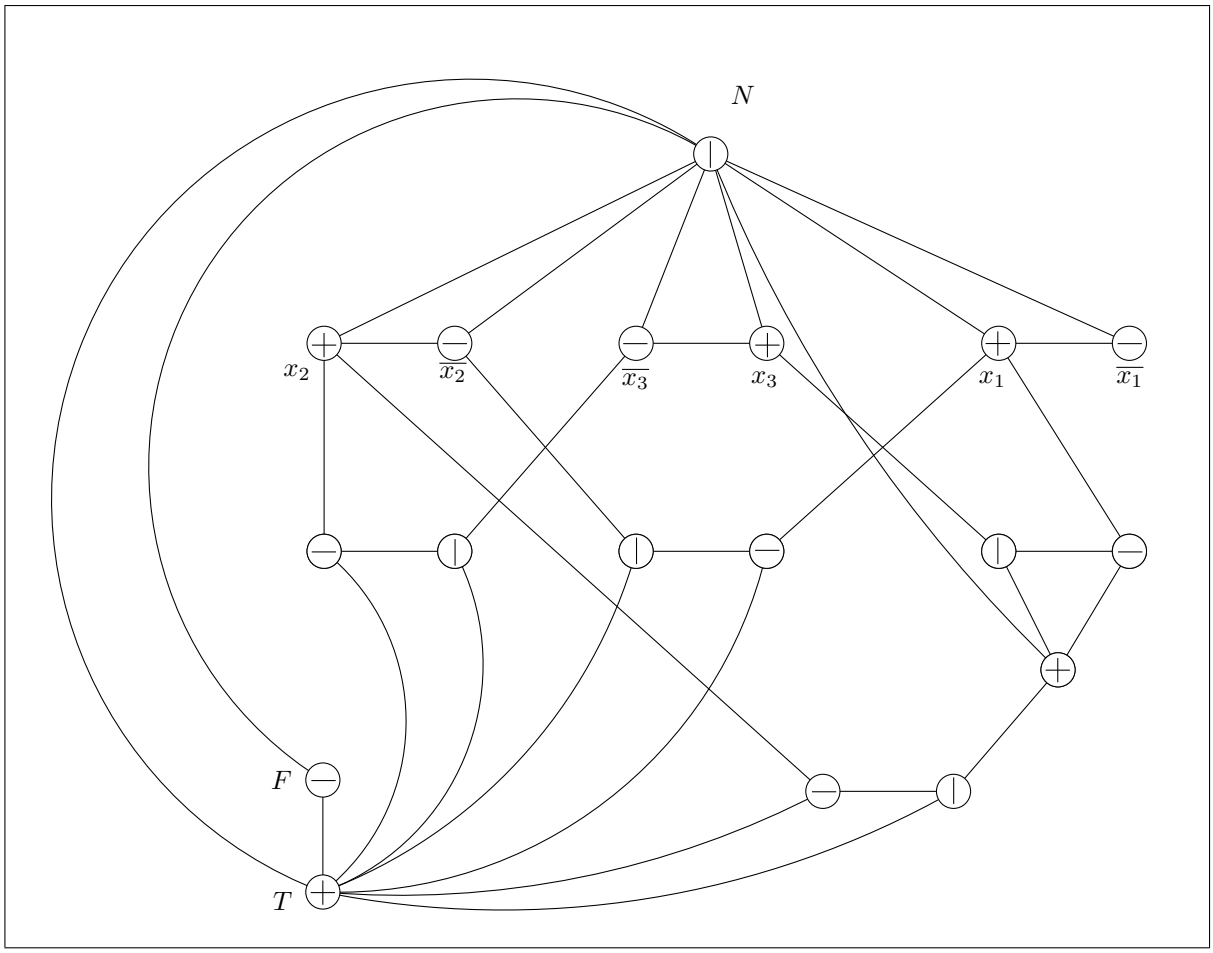
(a) [2]



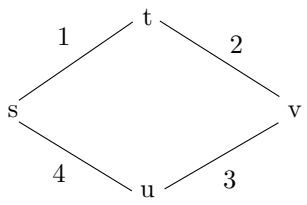
(b) [3]



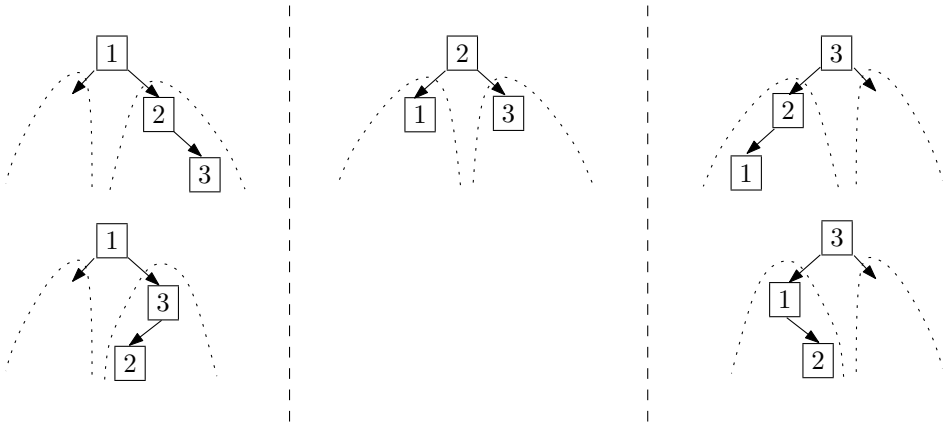
2. [5] The graph below was generated from the SAT expression $(x_1 \vee \overline{x_2}) \wedge (x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3)$. Label the nodes that correspond to $N, T, F, x_1, \overline{x_1}, x_2, \overline{x_2}, x_3$ and $\overline{x_3}$. Then colour the graph, using a $-$, $|$, and $+$ to represent the three colours. *Hint*: It is often easier to find a satisfying assignment of the variables and than a colouring.



3. [3] Give the edges below positive weights so that Prim's algorithm and Dijkstra's algorithm **must** find different trees when they start at s. That is, do not rely on tie breaking to create different trees. *Hint*: If every edge weight is unique, the minimum spanning tree is unique.



4. [6] Suppose that you want to count how many different binary search trees store the same n unique keys. Note that two trees differ if they have different roots. If they have the same root, the two trees differ if either their left subtrees or their right subtree differs.



Above are all of the binary search trees with the keys $1, 2, 3$. The dashed lines separate the trees into groups with common roots. Within each group, the trees differ by their subtrees: the trees with 1 as their root have different right subtrees, and the trees with 3 as their root have different left subtrees.

Let $k_1, k_2, \ldots, k_n$ be n unique keys in sorted order. If k_i is the root of a tree holding all the keys, the keys $k_1, k_2, \ldots, k_{i-1}$ are in the left subtree and the keys $k_{i+1}, k_{i+2}, \ldots, k_n$ are in the right subtree. Let $T(n)$ be the number of different binary search trees with n unique keys. Then there are $T(i-1) \times T(n-i)$ different trees with k_i as the root. Therefore,

$$T(n) = \sum_{i=1}^{n} T(i-1) \times T(n-i)$$

Applying this to the example above, you get $T(3) = T(0)\times T(2)+T(1)\times T(1)+T(2)\times T(0) = 2+1+2 = 5$.

(a) [3] Write a dynamic program that calculates $T(n)$. Analyse the run-time of your algorithm. Hint: It is easier to analyse the run-time if you write an iterative solution.

```
A ←new Integer[0...n]
A[0] ← 1
for nn ← 1 to n do
    A[nn] ← 0
    for i ← 1 to nn do
        A[nn] ← A[nn] + A[i − 1] × A[nn − i]
return A[n]
```

Runs in $O(n^2)$.

(b) [3] Use the guess-and-test method to show that $T(n) \in \Omega(2^n)$.

Our inductive hypothesis will be that $T(n) \geq \frac{1}{2} \times 2^n$. Then $T(0) = 1 \geq \frac{1}{2} = \frac{1}{2} \times 2^0$ and $T(1) = 1 \geq 1 = \frac{1}{2} \times 2^1$ are our base cases. Consider the inductive step:

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} T(i-1)T(n-i) \text{ by counting argument given earlier} \\ &\geq T(1-1) \times T(n-1) + T(n-1) \times T(n-n) \text{ keep just the first and last term} \\ &= 2T(0) \times T(n-1) = 2T(n-1) \\ &\geq 2 \times \frac{1}{2} \times 2^{n-1} \text{ by inductive hypothesis} \\ &\geq \frac{1}{2} \times 2^n \end{aligned}$$

So $T(n) \geq \frac{1}{2} \times 2^n$ for all $n \geq 0$. Hence $T(n) \in \Omega(2^n)$.

5. [5] Suppose that a 2-3-4-tree storing integer keys is represented with a data structure similar to what was discussed in class:

```
class 2-3-4-Tree {
   integer n
   boolean leaf
   integer key[1...n]
   2-3-4-Tree child[1...(n+1)]
   ...
}
```

Recall that leaf indicates whether a node is a leaf. Also recall the keys are ordered similarly to how they are in binary search trees: let α be any key stored in child$[i]$ and β be any key stored in child$[i+1]$; then $\alpha <$ key$[i] < \beta$.

(a) [2] Write a routine that prints out all of the keys stored in a 2-3-4-tree in sorted order. The algorithm should take $O(n)$ time, where n is the number of keys stored in the tree. Briefly justify your run-time.

```
Algorithm PrintAll(2-3-4-Tree T)
   if (not T.leaf) then
      PrintAll T.child[1]
   for i ← 1 to T.n do
      print T.key[i]
      if (not T.leaf) then
         PrintAll(T.child[i + 1])
```

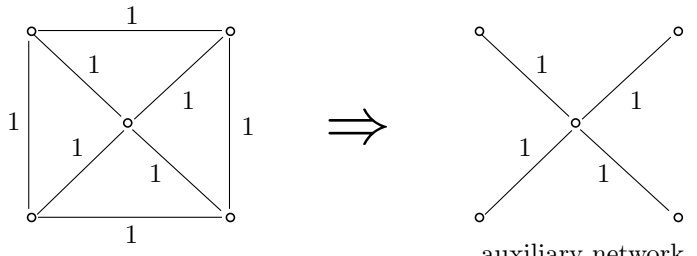
Each node is visited once. Every node has at least one key. So there are $O(n)$ nodes. At each node, a constant amount of work is done.

(b) [3] Adding a key to an 2-3-4-tree requires $\Theta(\log n)$ comparisons, where n is the size of the tree. Argue that no matter how clever we are, we cannot reduce this to $\Theta(1)$ comparisons. *Hint*: You need to show a lower bound; think carefully of other lower bounds we proved in this course, and use the fact that a solution for (a) exists.

Suppose for contradiction that we could reduce this to $O(1)$ time. Then we can add n keys to a 2-3-4-tree using $O(n)$ comparisons. By the previous question, we can output the keys in sorted order, which violates the $\Omega(n \log n)$ lowerbound on comparisons from decision trees.

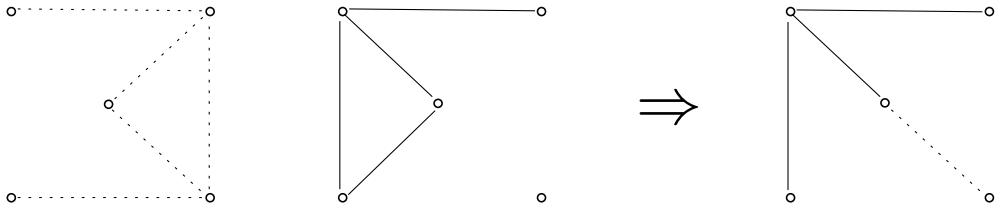
6. [4] You are given, as input, an existing computer network represented as an undirected graph: the nodes are computers and the edges are direct links between them. A network is robust if messages can still be sent between computers if a link fails. To improve robustness, you are asked to build an auxiliary network from a list of potential links P. Then, you can use the auxiliary network to communicate, if the original network fails. The auxiliary network is not meant to augment the original network: it is meant to replace it in the even of an emergency. To keep costs low, the links will be slow and you want to build as few links as possible, while guaranteeing that the auxiliary network is still connected. Specifically, you want your auxiliary network to be a tree.

   Note that you can build a tree from a list of potential links P with a minimum spanning tree algorithm:



potential links P auxiliary network

   To make this problem interesting (and difficult), you might need to use a few of the original links O to guarantee connectivity.



original links O potential links P auxiliary network

   You want to use as few of the links of the original network as possible because the auxiliary network is supposed to be a fail-over network. This way a broken link in the original network is less likely to cause a failure in the auxiliary network.

   Stated another way, you are given two sets of edges O and P. Find a spanning tree using edges from $O \cup P$ that uses as few edges from O as possible.

(a) [2] Describe an efficient algorithm to solve this problem. What is its run-time?

Create a graph with edges $O \cup P$. Set the edges from P to have weight 1. Make the edges from $O \setminus P$ have weight 2. Run Prim's algorithm. Let $|E| = |O \cup P|$. Then the run-time is $O((|V| + |E|) \log |V|)$.

(b) [2] Prove that your solution to (a) is correct.

Let p be the number of edges selected from P and o be the number of edges selected from O. Then $o+p = |V|+1$, which is a constant defined by input. The total cost is $2o+p = 2o+(|V|+1-o) = o+|V|+1$. This is minimized by minimizing the edges selected from o.

7. [8] Consider the closest pair of points on a line problem: Given the numbers $x_1, x_2, \ldots, x_n$ find the pair of numbers x_i, x_j such that $|x_i - x_j|$ is smallest and $i \neq j$. Solve this problem with a divide-and-conquer algorithm **without** directly sorting any numbers. *Do not call the closest pair of points algorithm covered in class as a subroutine because it sorts by y-coordinates.*

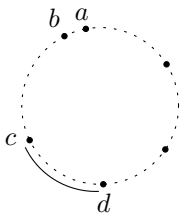
   (a) [3] Describe your algorithm. Why is it correct?

Pivot on the median. Recursively find the closest pair of points in the left partition and the closest pair of points in the right partition. Compare the rightmost point in the left partition to the pivot. No point on the left could be closer to any other point outside of the partition. Similarly, compare the leftmost point in the right partition to the pivot.

   (b) [2] Write a recurrence for its worst run-time. Use the Master Theorem to solve the recurrence.

Recurrence is $T(n) = 2T(n/2) + n$. This is case two and solves $T(n) \in \Theta(n \log n)$.

(c) [3] Consider the problem of finding the closest pair of points on a *unit radius* circle, where distance is measured by traveling along the circle (geodesic distance — similar to traveling between two points on the Earth). Briefly describe how your result from (a) could be used to solve this problem.



Closest pair of points are a and b.

Distance from c to d is length of solid arc.

Split the circle at the top and straighten to make a line. This is equivalent to measuring the points in radians from vertical. Find the closest pair of points as before. Repeat, but this time split the circle at the bottom. The breaks are exactly 1/2 circle apart. As no shortest path is more than half a circle, one of them will be unaffected by the break. The closest pair of the two different runs is the closest pair.