READ THIS NOW!

- Print your name in the space provided below.
- There are 7 short-answer questions, priced as marked. The maximum score is 100.
- Aside from the allowed one-page fact sheet, this is a closed-book, closed-notes examination.
- No laptops, calculators, cell phones or other electronic devices may be used during this examination.
- You may not discuss this examination with any student who has not taken it.
- Failure to adhere to any of these restrictions is an Honor Code violation.
- When you have finished, sign the pledge at the bottom of this page and turn in the test and your fact sheet.

Name (Last, First) Solution
printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

signed

1. Consider the following algorithm:

```
for (i = 1; i <= N; i++) {              // 1 before, 2 per pass, 1 to exit

   x = 0;                               // 1 per pass of outer loop
   y = N;                               // 1 per pass of outer loop

   for (j = 1; j <= i/2; j++) {         // 1 before, 3 per pass, 2 to exit

      if ( i % j < 2 ) {                // 2 per pass of inner loop
         x = y + j;                     // 2, if done
         y--;                           // 1, if done
      }
      else {
         x = y - j;                     // 2, if done
      }
   }
}
```

a) [12 points] Using the rules given in class for counting operations, find a simplified function T(N) that counts the exact number of operations the algorithm would perform.

$$T(N) = 1 + \sum_{i=1}^{N}\left(2+1+1+1+\sum_{j=1}^{i/2}\left(3+2+\max(3,2)\right)+2\right)+1$$

$$= \sum_{i=1}^{N}\left(\sum_{j=1}^{i/2}(8)+7\right)+2$$

$$= \sum_{i=1}^{N}\left(8\cdot\frac{i}{2}+7\right)+2$$

$$= \sum_{i=1}^{N}(4i+7)+2$$

$$= 4\frac{N(N+1)}{2}+7N+2$$

$$= 2N^2+9N+2$$

b) [4 points] To what big-Θ complexity class does your answer to the previous part belong? (No proof is necessary.)

Obviously, T(N) is $\Theta(N^2)$.

2. [16 points] Consider the following two functions:

$$f(n) = n\log n \text{ and } g(n) = \log^2 n$$

Determine the complexity relationship between the two functions. That is, determine whether f is strictly $\Omega(g)$, or f is strictly $O(g)$, or f is $\Theta(g)$. Whichever conclusion you reach, state it clearly and justify your conclusion completely by applying relevant theorems from the course notes.

Since g(n) isn't listed in the functions for Theorem 5, you cannot apply that theorem. So, you'll have to either apply the definition, which is not advisable, or Theorem 8. Consider the relevant limit:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{n\log n}{\log^2 n} = \lim_{n\to\infty}\frac{n}{\log n} \quad \text{which is of the form } \frac{\infty}{\infty}.$$

So, apply l'Hopital's Rule:

$$\lim_{n\to\infty}\frac{n}{\log n} = \lim_{n\to\infty}\frac{1}{1/n\ln 2} = \lim_{n\to\infty} n\ln 2 = \infty$$

By the Corollary to Theorem 8, since the limit is infinite we know that $f(n)$ is strictly $\Omega(g(n))$.

3. Assume that you have a collection of data objects that correspond to points with integer coordinates in the square region of the xy-plane bounded by x = 0, y = 0, x = 1024, y = 1024. Assume that you will organize the data objects in a PR-quadtree (with bucket size 1).

   a) [10 points] If the tree contains N data objects, what is the minimum number of levels the tree could have (as a function of N)? Justify your answer.

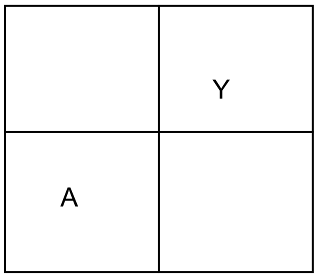
Since it's a quadtree, the maximum number of nodes in each level is 4 times the number in the previous level. So, the maximum number of nodes in level k would be 4^k. Data values are only stored in leaves, and it's clear that to achieve the shortest possible tree we'd need to have the maximum width in each level and the data all stored in leaves in the bottom level (or something close to that).

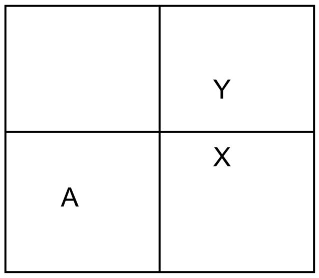
So, we need for *k* to satisfy $4^k \geq N$ or $k \geq \log_4 N$. The number of levels would be *k+1*, so the minimum number of levels in the tree would actually be $1 + \log_4 N$..

   b) [10 points] Suppose that the PR-quadtree is full enough that every leaf node represents a region that is 8 x 8 or smaller. A new data object, X, is inserted into the tree, and the distance between X and the closest existing data object in the tree is 2. Will the insertion of X necessarily cause a node to split? Either way, justify your conclusion.

There will be a unique leaf to which X should be inserted. If the leaf is empty (i.e., has not been created yet), then no splitting will be required. If the leaf already exists, then it contains a value and is therefore full, and so it must be split.

Suppose that the closest existing data object to X is Y. The primary question is whether it is necessary that X will fall into the same sub-region that Y is in. If yes, then a split must occur. Thus, the real issue is whether it is possible that Y is in one quadrant of a larger region that has already been split, and X is in a different, empty quadrant of that region. Clearly that situation can occur; suppose the current situation is shown by the diagram on the left (note that there must be another data value besides Y in order for this region to have already been split). Then X could fall in a different, empty sub-region and still be close to Y, and no splitting would occur:





4. [8 points] The PR-quadtree used in the GIS project is "bucketed". That is, the leaf nodes are designed to store more than one value. What is the advantage of that? Be precise.

Making each leaf a "bucket" means that leaf nodes will not split as soon, and hence that some branches will be shorter than they would be otherwise.

That means that the PR-quadtree will use somewhat less memory since there will be fewer internal nodes in those branches.

But the primary advantage is that searches down those branches will be faster; since there are fewer levels of internal nodes, we have less cost of determining which sub-region to go into and fewer pointer dereferences.

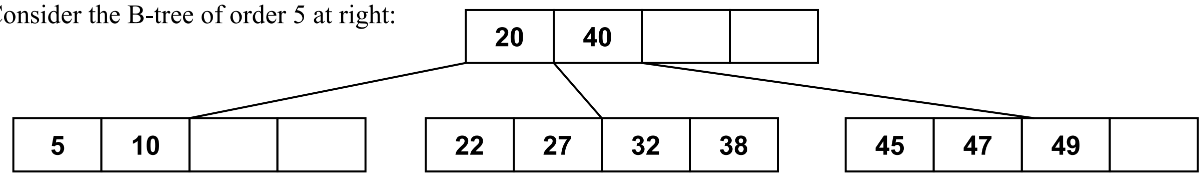
5. [8 points] What is the specific advantage of the B-tree over, say, a BST in organizing access to very large collections of records?

Unless the trees are stored on disk, the B-tree offers NO advantage.

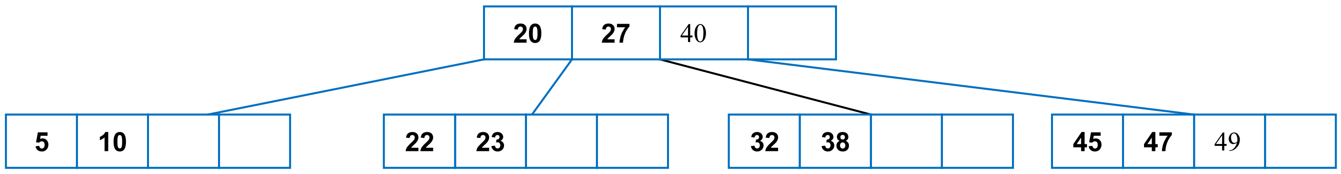
For on-disk storage, the B-tree provides for more efficient accesses to the file in two ways:

- **The B-tree nodes are large, containing many data values, so each read from disk retrieves more relevant information that in a BST, where each node would store only one data value.**
- **The B-tree has many fewer levels than the BST, and therefore fewer disk reads will be needed.**

6. Consider the B-tree of order 5 at right:



a) [8 points] Draw the resulting B-tree tree if the value **23** is inserted.



b) [8 points] Consider the original tree. Would deleting any of the values from it cause a restructuring of the tree? If so, which value(s) and why?

It depends on what you mean by "restructuring".

It is possible to delete values from the original tree that would cause a node besides the root to "underflow", to have less than 2 data elements.

However, since the second leaf has "extra" values, the result would be that the left-most leaf would have to "borrow" a value from its sibling node to remove the underflow condition. So, there wouldn't be any physical restructuring of the tree, although values would be moved around.

7. Suppose you want to implement a container to store a collection of objects that represent tasks, each of which has a priority attribute, represented as a positive integer. The container must efficiently support two primary operations: removing the object representing the highest-priority task, and adding new objects. You will consider top options for the physical structure used within the container: a sorted array and a max-heap. For the following questions, assume that the container currently stores N objects.

a) [8 points] Compare the efficiency of the two candidates for the physical structure for supporting the removal of the highest-priority task from the container. Remember that the removal must leave the physical structure in a proper state – still a heap or still a sorted array. Your answer must make use of what we know about the complexity of operations on the two structures; mentioning Theta results would be a good idea.

In the max-heap, the highest-priority task will always be at the root. So the cost of finding it is Theta(1). When the root is removed, the last leaf value is moved to the root, which is also Theta(1), and that value is then sifted down, which is Theta(log N). So, the total cost for the max-heap would be Theta(log N).

For the sorted array, the highest-priority item is at one end or the other. Either way, the cost of finding it would be Theta(1). If the array stores elements in ascending order, the cost of removing the highest-priority element would be Theta(1). If the array stores elements in descending order, the cost of shifting elements would be Theta(N). But, the array could be used in circular fashion, and that would eliminate the need to shift. So, the cost of removal for the sorted array is either Theta(1) or Theta(N), depending on how clever we are.

b) [8 points] Repeat the first part of this question, but now consider the operation of adding a new task.

To add a new element to the heap, we insert it into a new, last leaf, and then sift it up to the proper level. Adding a leaf is Theta(1), sifting up is Theta(log N), so the cost of insertion is Theta(log N).

Finding the proper location for the new element could use a binary search and the cost of that would be Theta(log N). However, note that there's no point in doing this as a separate step…

No matter how we use the sorted array (see answer to part a), insertion will require shifting a section of the elements to create a "hole" for the new element in its proper location. This might as well be done as part of the search. On average, we'll probably shift half the elements in the array, so the cost would be Theta(N).