First, just a review of the topics which we have seen before the first midterm:

- conversion to binary (NIM used as an example)
- loop invariants and proofs (insertion sort, selection sort, and bubble sort all used as examples)
- order of growth, recurrence relationships, and the Master Theorem
- mergesort, quicksort, heapsort
- quickselect: selection in expected linear time
- heap operations
- sorting lower bounds
- counting sort, radix sort, bucketsort, and what a “stable-sort” is.
- BSTs (review), 2-3 trees, top-down 2-3-4 trees, and B-trees

Here is a sample of practice problems for the midterm, which I have used for some semesters. Any topics covered in class can be covered, including those not covered on this sample test, but I think the list above pretty much covers what we have done in class. Any feedback you want to give me is needed by Wednesday, October 13, as I might actually create the exam after that. Many of these questions come from my past tests, and this review is unchanged from one used several times now. The following would be about three tests worth of questions total.

**Important:** Remember how points are given. If you do not try a problem, or if you mark it for non-grading, you will get 30% for that problem. You can take this option for up to half of the test.

You may use the Master Theorem. It follows, in a somewhat over-simplified format. (Restrictions for case 3 have been dropped, they won’t be needed for the test.) I will give the following on the exam as well.

For $T[n] = aT[n/b] + f(n)$, then

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T[n] = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T[n] = \Theta(f(n) \log n)$.
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, then $T[n] = \Theta(f(n))$.

1. Define what it means for a function $f(n)$ to be $O(n)$, $\Omega(n)$, and $\Theta(n)$.
2. Give $\Theta()$ bounds on the following recurrence relationships. Prove your bounds.
   
   - $T[n] = T[2n/3] + 1$
   - $T[n] = T[n/2] + n$
   - $T[n] = T[n - 1] + 1$
   - $T[n] = T[n - 1] + n$
   - $T[n] = 2T[n/2] + 1$
   - $T[n] = T[\sqrt{n}] + n^2$.
   - $T[n] = 2T[n/2] + n^2$.
3. (10 points) Consider the unsorted array:
72, 61, 53, 24, 15, 36, 87, 48. Imagine that you run a sorting algorithm on the array, and at some moment in time, the array is ordered as:
61, 72, 53, 24, 15, 36, 87, 48. Which of the following sorting algorithms may have been running? Circle all possibilities, and cross out impossibilities. (You start with 2 points, and get 2 per correct, and -2 per incorrect. For this problem, you can answer some questions but leave others blank.) Assume that for quicksort/mergesort, the only possibilities are just after the recursive calls or the separation/merge, and for radixsort, the only possibilities are just after one phase is completed. Furthermore, for quicksort, assume that the partition algorithm moves the pivot to the correct location, and divides other items around it, in a stable way. (This isn’t the case of the partition algorithm we saw in class, but will help you to simulate the partition algorithm much more quickly for this question.)

Quicksort (1st item as pivot) Mergesort
Heapsort (starting with linear build min-heap) Radixsort (base 10)

4. You run insertion sort on a set of \( n \) random integers, for some large \( n \). It takes \( x \) seconds to run. Next, you run it on \( 3n \) random numbers. How long should it take? Briefly explain your answer.

5. Given the following heap:

\[
\begin{array}{ccccc}
98 \\
94 & 34 \\
22 & 90 & 2 & 32 \\
20 & 4 & 87 \\
\end{array}
\]

- Draw the array ordering to hold this heap.
- Show the heap after deleting node 2.
- Insert 2 into the resulting heap and draw.
- Insert 95 into the resulting heap and draw.

6. Given the following numbers 5, 4, 7, 8, 9, 3, run the linear time make-heap algorithm and show the resulting heap. Show work.

7. Insert a 17 into the top-down 2-3-4-tree below: SOME 2-3-4 TREE GOES HERE

8. Delete the 17 from the top-down 2-3-4-tree below: SOME 2-3-4 TREE GOES HERE

9. You run two standard sorting algorithms, mergesort and bubblesort, on a set of \( n \) randomly ordered random integers, for \( n = 2^{20} \). It takes \( M \) seconds for mergesort, and \( B \) seconds for bubblesort. Next, you run a different set of random numbers, of unknown size, and bubblesort takes \( 4B \) seconds. How long should mergesort take? Briefly explain your answer.

10. You are given \( n \) numbers to sort, where each number is a real (double) chosen randomly between 0 and 1000. You sort it with “the best” sorting algorithm we have learned for such an input, and it takes \( X \) seconds. Next, you are asked to sort \( n/2 \) numbers, where each number is a real (double) chosen randomly between 0 and 500. How long will the same sorting algorithm take? Explain, including the name of the sorting algorithm used.
11. The following is supposed to represent a max-heap in array format: 8, 7, 4, 1, 6, 2, 1, 0, 4, 0, but it has a problem.

- Draw the “heap” in its more intuitive shape.
- Circle the problem in the “heap” above? (Circle it in your diagram.)
- Suppose that the problem comes from a change-key operation, which has just been called, but which has not yet run to completion. Correct the heap to show what it looks like after the operation is completed. (Neatly draw corrections above.)
- Starting with the corrected heap from the previous part, delete the 4 in the heap which is initially at array position 2 (the 3rd number in the array), and draw the resulting heap below.

12. We have seen that we can make a heap by iterated insertion, or by the clever linear-time make-heap operation. These two operations have different runtimes, but do they give different heaps? If they always give the same heap, give a short proof. If not, find a short counter-example. In either case, a long answer will not get full credit.

13. For the following recurrence relationships, prove as good a bound as you can. (Part credit for properly proving non-optimal bounds.) (You can use whatever method you like to get a “guess”, but should then show the guess correct as we have in class.)

- Give \( O(\ ) \) notation for \( T[n] = 2T[n/4] + n \).
- Give \( \Omega(\ ) \) notation for \( T[n] = 4T[n/2] + n \).

14. You are given an array with \( n \) distinct numbers. The array is guaranteed to be almost sorted: within the array, for \( 1 \leq i \leq n-1 \), the \( i \)th smallest number is in position \( i-1 \) or \( i \). (The array has indices 0 to \( n-1 \).) The largest number in the array (the \( n \)th smallest) is either in position 0 or \( n-1 \). (A fully sorted array will have the \( i \)th smallest in position \( i-1 \)).

Give a fast algorithm to return the \( \lfloor n/2 \rfloor \)th smallest value from the array. (You may assume that \( n > 2 \).) Briefly explain why your algorithm works.

15. Consider the game of NIM, covered in class and on your homework, where the object is to move last. Imagine that the numbers following are the numbers of sticks left on each line of a board, and that it is your turn. List all possible winning moves for the game below. (Show work for partial credit on a careless mistake.) (Note: this was assigned during a semester when NIM was one of the programming assignments. I would not give a question like this otherwise, I don’t consider the game to be a big topic for the class.)

1,3,4,5,6,6

16. Radix sort is run on \( n \) numbers, where each number can be represented by 10 hexadecimal digits. (Hexadecimal means base 16.) It takes \( R \) seconds. (Assume \( n \) is large, so you may ignore startup time, and that a base of 16 (much smaller than \( n \)) was used in the radix sort.) Next, each of the \( n \) numbers in the set is multiplied by each of the \( n \) numbers, giving a set of \( n^2 \) numbers, each larger than any number in the original set. How long should radix sort take on the new set, still using base 16? (Show work/explanation.)

17. Imagine that mergesort is called on the unsorted array 7,1,4,2,5,6,2,6. Which of the following are intermediate “snapshots” possible during mergesort? (Assume that the snapshots are only taken place after a merge operation has completed. Also assume that when creating two sublists, mergesort keeps each in increasing order, and recursively goes left to right.) (All sets have the same 8 numbers.) Circle all possible snapshots, cross out impossible snapshots. (You start with 3 points, and get 1 per correct, and -1 per incorrect. For this problem, you can answer some questions but leave others blank.)

1,7,4,2,5,6,2,6  1,4,7,2,5,6,2,6  1,2,4,7,5,6,2,6  1,2,4,5,6,7,2,6  1,2,2,4,5,6,7,6  1,2,4,7,2,5,6,6

3