[**Closed book/notes/calculator, you may use one side of an 8.5 $\times$ 11 inch sheet of paper**] Show all of your work clearly in the space provided or on the additional page at the end of the exam. If the additional page is used, clearly identify to which exam question it is related. Be sure to **read each problem carefully**. You should answer all 5 questions, and you may wish to answer the bonus question if you have time. Note that the exam is double sided.

1. In lab 2, suppose that you found that your InsertionSort algorithm required x seconds to sort a vector of N strings while your HeapSort algorithm required y seconds to perform the same task.

(a) (10 points) Set up the equations to determine the constant coefficient for each of your algorithms when run on an N element vector. In other words, if algo() is a $\Theta(N)$ algorithm, find α for which αN is the actual runtime of your algorithm.

(b) (10 points) Suppose that for small input sizes, the constant coefficient is c_i for your InsertionSort algorithm and c_h for your HeapSort algorithm where $c_i < 50c_h$. Suggest an approach for improving the execution time of your HeapSort algorithm. Be specific.

2. (15 points) How many comparisons (± 5) will quicksort make when sorting an array of N **equal** elements?

3. (20 points) Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Consider the following statement:

$\min(f(n), g(n)) = \Theta(f(n) + g(n))$

If true, use the basic definition of Θ-notation to prove it. If false, provide a specific example for which the equation fails to be true.

4. (20 points) At the end of the quarter, you need to store your favorite sorted array x of N words. Fortunately, Sand 'n Stuff, a company that manufactures storage devices, has a good deal on a device for storing data. You buy it, store x on it, and put in storage for the summer. Suppose that you return in the fall to find that the data has been scrambled.

You contact Sand 'n Stuff to complain. They assure you that all of the data is still there, it has just been rearranged slightly. You learn that a flaw in the memory device causes elements in your array to "float." For example, an item that was located at $x[1]$ can float to $x[3]$, forcing the item that used to sit in $x[3]$ to float somewhere also.

The good news is that the memory cells didn't float very far. You have be told that the maximum float distance is k ($k > 0$). That is, the final resting place of an element at the end of the summer is no more than k positions away from its location at the beginning of the summer. (I.e., the data that used to be at $x[i]$ is now somewhere between $x[i-k]$ and $x[i+k]$.)

What is the worst case time complexity (Θ) for InsertionSort when applied to your slightly scramble array, x? Your answer should be given as a function of the array size, N, and float distance, k. Be sure to justify your answer.

5. (25 points) An element x of an array A is the *majority* element of A if at least 50% of the array elements have the value x (Note: it is not neccessary for an array to have a majority element.)
Suppose you can perform comparisons on the elements of the array. The comparison test can give three possible outcomes: "=", "$<$", and "$>$". Describe in C++, pseudocode, or unambiguous English an algorithm to determine whether or not a majority element exists. Your algorithm should require $O(n)$ comparisons. Hint: the following algorithm may be a helpful part of your solution.

Select(A[1..n], int p, int r, int i)

```
{ // Returns the i^th order statistic
  if(p=r) {
    return A[p];
  }
  q ← partition(A, p, r)  // The one from Quicksort
  k ← q − p + 1
  if(i≤k) {
    return Select(A, p, q, i)
  } else {
    return Select(A, q+1, r, i−k)
  }
}
```

bonus (10 points… no partial credit) Suppose that you an a basket of fruit containing apples, oranges, and bannanas. Comparing apples to oranges is a no-no, so we are left with a comparison operator that only has two outcomes: "=" and "!=". Describe in C++, pseudocode, or unambiguous English an algorithm to determine whether or not a majority element exists. Your algorithm should require $O(n)$ comparisons.
Hint: First show that if $A[i] \neq A[j]$ are both removed from the array, x is the majority element of the resulting array if and only if x was the majority element of the original array.

Additional work area for any problem. Clearly identify to which problem the work on this page is related.

Additional work area for any problem. Clearly identify to which problem the work on this page is related.