INSTRUCTIONS
Students should feel free to discuss these problems.
KEY CONCEPTS Sorting, Asymptotic notation, loop invariants.

1. For each of the following claims, say whether it is true or false, and give a short explanation (2 points correct answer, 2 points explanation).

   (a) $2^n \in O(n!)$
   True. $n! = n*(n-1)*(n-2)*...2*1$. $n-1$ of these factors are at least 2, so $n! > 2^{n-1} = 1/22^n$ for $n \geq 3$. This meets the definition of O with $k = 3, C = 2$.

   (b) $2^n \in o(n!)$
   True. For almost the same reason as above. $2^n/n! = 2^n/n(n-1)! < 2^n/(n2^{n-2}) = 4/n$ which goes to 0 as n goes to infinity. So 2^n is strictly smaller asymptotically than $n!$.

   (c) $2^{2n} \in O(2^n)$
   False. $2^{2n}/2^n = 2^n$ which goes to infinity as n goes to infinity. Therefore, 2^{2n} is not in $O(2^n)$.

   (d) $(n^2+n+3)^3 \in \Theta(n^6)$.
   True. The leading term of this expression is $(n^2)^3 = n^6$, so by the "order of the sum is the maximum" rule, the expression is $\Theta(n^6)$

   (e) $\sum_{i=1}^{\lceil \log_2 n \rceil} n/2^i \in \Theta(n)$.
   True. While we can't use the order of the sum is the maximum, because there are unbounded terms, $\sum_{i=1}^{\lceil \log_2 n \rceil} n/2^i =$
   $n(\sum_{i=1}^{\lceil \log_2 n \rceil} 1/2^i < n\sum_{i=1}^{\infty} 1/2^i = n$ because this sum coverges to 1. On the other hand, for $n \geq 2$, even the first term, $n/2$ is $\Omega(n)$, so the sum is at least $n/2$ and at most n for $n \geq 2$.

As part of selection sort, we saw how to find the minimum element of an array $A[1..n]$ of distinct integers using $n-1$ comparisons. We could also find the maximum element of the array using the same number of comparisons.

2. (a) (10 points) Give an algorithm that finds BOTH the maximum and minimum value in the array using exactly $3n/2 - 2$ comparisons for a list with n elements. (assume that n is even.).
   If $A[1] > A[2]$ we initialize min to be $A[2]$ and max to be $A[1]$, otherwise, we initialize min to be $A[1]$ and max to be $A[2]$. For $I = 2$ to $n/2$ we first compare $A[2I]$ with $A[2I+1]$. We compare the larger of the two to max and update max to this value if it is larger than max. We compare the smaller of the two to min and update min if the value is smaller than min.
   We return the values max and min.

   (b) State and prove a loop invariant for this algorithm, and use it to conclude your algorithm is correct.
   We prove that, after the iteration for $I = i$, we have $max = max_{1 \leq j \leq 2i}(A[j])$ and $min = min_{1 \leq j \leq 2i}(A[j])$. We consider the initialization step the iteration for $I = 1$.
   By construction, we initialize $min = min(A[1], A[2])$ and $max = max(A[1], A[2])$, so the claim is true after initialization for $i = 1$.
   Assume the claim is true for $I = i$, i.e, after this iteration, and $max = max_{1 \leq j \leq 2i}(A[j])$. and $min = min_{1 \leq j \leq 2i}(A[j])$. If both $A[2i+1]$ and $A[2i+2]$ are at least the current value of min, then min does not change in this iteration, and $min_{1 \leq j \leq 2i+2}(A[j]) = min_{1 \leq j \leq 2i}(A[j]) = min$. If one of them is smaller than min, then the minimum of the two is smaller than min, and this becomes the new value of min. In this case $min_{1 \leq j \leq 2i+2}(A[j]) = min(A[2i+1], A[2i+2]) = min$ after the next iteration. So in either case the invariant still holds for min after this iteration. Symeterically, the invariant still holds for max. Thus, by induction, the invariant always holds.

In particular, when $I = n/2$, $2i = n$, the algorithm ends, and the invariant tells us and $max = max_{1 \le j \le n}(A[j])$ and $min = min_{1 \le j \le n}(A[j])$, as desired.

(c) Explain carefully the number of comparisons this algorithm uses (exact, not just order)

In the intialization step, we make one comparison. Then in $n/2 - 1$ iterations, we make three comparisons. Thus, the total number is $1 + 3(n/2 - 1) = 3n/2 - 2$.

(d) Design a sorting algorithm (related to SelectionSort) that uses the algorithm from part a as a subroutine.

At iteration t , we are sorting $A[t...n-t+1]$. We use the above to find the minimum and maximum element of the array. and swap the minimum with $A[t]$ and the maximum with $A[n-t+1]$.

This maintains the invariants that $A[1..t]$ are the t smallest in sorted order, and $A[n-t+1..n]$ are the t largest in sorted order.

(e) Recall that SelectionSort does $n(n-1)/2$ comparisons for any input of size n. How many comparisons does your algorithm from part b do for an input of size n? (You can assume that n is an even number.)

In the first iteration, we do $3n/2-2$ comparisons. In the second iteration, we do $3(n-2)/2-2 = 3n/2-5$ comparisons. In the t'th iteration, we do $3(n-2t+2)/2-2 = 3n/2-3t+1$ comparisons. There are $n/2$ iterations in all. Thus, the total number is $\sum_{t=1}^{n/2}(3n/2 - 3t + 1) = (n/2)(3n/2 + 1) - 3\sum_{t=1}^{n/2} t = (n/2)(3n/2 + 1) - 3((n/2 + 1)(n/2)/2)$ using the sum formula from class. This simplies to $3/8n^2 - n/4$ comparisons, about a 25 % savings over Select sort.

3. For each algorithm, compute the exact number of times the algorithm prints in terms of n and compute the runtime in terms of Θ, where $n \ge 3$. (show your work.)

(a)
```
for i = 1 to n:
    for j = 1 to i − 1:
        print (i, j)
```

The number of prints is $\sum_{i=1}^{i=n}(i-1) = \sum_{i=1}^{i=n-1} i = (n)(n-1)/2$ using the "little Gauss sum" formula from class. This is $\Theta(n^2)$ total time.

(b)
```
i = 0
j = 0
while  i < n:
    print i.
    i = i + 2j + 1
    j = j + 1
```

This algorithm maintains the invariant that after the t'th iteration, $j = t$ and $i = t^2$. (This is true when $t = 0$, and if it is true for t, then the new value for $i = i + 2j + 1 = t^2 + 2t + 1 = (t + 1)^2$ and the new value for j is $t + 1$.) Thus, the while loop ends when $i = t^2 \ge n$, i.e., $t = \lceil \sqrt{n} \rceil$, and prints exactly once each iteration, and takes constant time in each iteration. So $T(n) \in \Theta(n^{1/2})$.

(c)
```
for i = 1 to n − 2:
    for j = i + 1 to n − 1:
        for k = j + 1 to n:
            print (i, j, k)
```

This algorithm prints each triple i, j, k with $1 \le i < j < k \le n$ exactly once. There are $(n)(n-1)(n-2)/3$ such triples, which is $\Theta(n^3)$.

4. State if each statement is true or false and give a justification either way. (you may use any of the methods used in class including the limit rules.)

(a) $n/2 + n/3 + n/4 + \ldots + n/n \in O(n)$

Hint: You can use the fact that the harmonic series is divergent in your proof.

False. The left side is $n(1/2+1/3+...+1/n)$. The sequence $1/2+1/3+....$ is called the harmonic series and diverges, meaining it is greater than any fixed constant, and so for any C, the sum on the left is strictly larger than Cn for large enough n. (The sum is $\Theta(\ln n)$, because you can approximate it with the integral of $1/x$ from $x = 2$ to $x = n$, so the left side is actually $\Theta(n \log n)$.)

(b) $2^{log_2(n)} = O(2^{log_4(n)})$

False, $2^{log_2 n} = n$ whereas $2^{log_4 n} = (2^{log_2 n/\log_2 4}) = (2^{log_2 n})^{1/2} = \sqrt{n}$.

(c) $\left(\sqrt{n} + \sqrt[3]{n}\right)^2 \in O(n \log n)$

True. The $n^{1/2}$ dominates $n^{1/3}$, so the order of the sum is $\Theta(n^{1/2})$. We then square it, making the left hand side $\Theta(n)$, which is $O(n \log n)$.

(d) $n! \in O(n^n)$

True. $n! = n(n-1)..1 \leq n * n * n... * n = n^n$, so $n! \in O(n^n)$.

(e) $1 + 4 + 4^2 + 4^3 + \ldots + 4^n \in \Theta(4^n)$

True. A geometric series is dominated by its largest term.

(f) $1^2 + 2^2 + ..n^2 \in= O(n^2)$.

False. $1^2 + 2^2 + ...n^2 \geq (n/2)^2 + (n/2+1)^2 + ...n^2 \geq n/2(n/2)^2 = n^3/8$. so the sum will grow at $\Theta(n^3)$ not $O(n^2)$.

5. Design a "binary search style" algorithm for the following problem: The input is an array of integers $A[1..n]$ so that $A[1] > A[n]$. The problem is to find an integer I so that $1 \leq I \leq n-1$ and $A[I] > A[I+1]$, i.e., a particular place where the list is not sorted. Use loop invariants to show that your algorithm correctly solves this problem, and that it takes $O(logn)$ time.

Here's an algorithm that uses a binary seach like approach to find the position I.

UnsortedWitness($A[1..n]$, array of integers with $n \geq 2, A[1] > A[n]$.)

(a) $I = 1, J = n$

(b) While $J > I + 1$ do:

(c) $M = \lfloor (I+J)/2 \rfloor$.

(d) If $A[M] > A[M+1]$ return M

(e) If $A[I] > A[M]$ then do: $J = M$

(f) else do: $I = M + 1$.

(g) Return I

We prove that this algorithm maintains the invariant that $A[I] > A[J]$ after any number t iterations of the while loop. First, at $t = 0$, $I = 1, J = n$ and $A[1] > A[n]$ by the condition on the input. Assume after t iterations, $A[I] > A[J]$, and we continue for the next iteration without returning, so $A[M] \leq A[M+1]$ for $M = \lfloor (I+J)/2 \rfloor$. If $A[I] > A[M]$, then the new value of J is M, and the invariant still holds. Otherwise, the new value of I is $M+1$ and $A[M+1] \geq A[M] \geq A[I] > A[J]$ by the invariant at the previous iteration. So the invariant still holds after this iteration.

In particular, when the algorithm returns, it is either because $A[M] > A[M+1]$ and we output M, or $J = I + 1$ and we output I, in which case the invariant tells us that $A[I] > A[J] = A[I+1]$. In either case, we output a position where A is out of order.

To analyse the running time, we show a second invariant: that after t iterations, $1 \leq J - I < n/2^t$. At the start, $J - I = n - 1 < n/2^0$ so the base case holds. Assume after t iterations $1 \leq J - I \leq n/2^t$. Then in the next iteration, $J - I$ is either $M - I$ or $J - (M+1)$ for $M = \lfloor (I+J)/2$. In the first case $M - I \leq (I+J)/2 - I = (J-I)/2 \leq (n/2^t)/2 = n/2^{t+1}$ In the second case, $J - (M+1) \leq J - (I+J+1)/2 - 1 < (J-I)/2 < n/2^{t+1}$. So in all cases the invariant holds.

In particular, when $t = logn - 1$, $J - I < n/2^t = n/(n/2) = 2$, so either the algorithm already terminated, or $J = I + 1$ and the algorithm terminates in this step. Thus, the algorithm takes less than $logn$ iterations to terminate, and each iteration is constant time, so the total time is $O(\log n)$.