Problem 1. Assume that

- Insertion Sort uses $2n^2$ instructions to sort n values.
- Merge Sort uses $50n \lg n$ instructions to sort n values.

Assume that

- Computer A executes 10 billion instructions / second.
- Computer B executes 10 million instructions / second.

(a) How much time does Computer A use to sort 10 billion values using Insertion Sort? Show your work, but not the actual calculations.

(b) How much time does Computer B use to sort 10 billion values using Merge Sort? Show your work, but not the actual calculations.

(c) What do we learn from this?

Problem 2. Here is an algorithm to find the largest value in a list of size n.

```
largest ← 1
for i = 2 to n do
    if A[i] > A[largest] then largest ← i
end for
print(A[largest])
```

An alternative is to run the algorithm as a ladder, as in done in the NCAA basketball championship (March Madness): Compare the first and second elements of the array, compare the third and fourth elements of the array, compare the fifth and sixth elements of the array, etc. Then compare the larger of the first and second elements of the array, with the larger of the third and fourth elements of the array, compare the larger of the fifth and sixth elements of the array, with the larger of the seventh and eighth elements of the array, etc. Etc.

Write the ladder algorithm in pseudo-code. You may only use a constant amount of extra memory. You may reorganize the elements of the array, but you may not destroy them. Do NOT use recursion (which, in any case, would use extra memory). You can assume that the array is stored in locations $1, \ldots, n$ or locations $0, \ldots, n-1$, whichever you prefer. Assume that n is a power of 2.

Problem 3. **Challenge problem. Will not be graded.** Write the ladder algorithm in pseudo-code assuming general n (not necessarily a power of 2).