

- 
1. Consider the list [57, 91, 64, 13, 27, 88, 63]. Show how Insertion Sort sorts the list by showing the list after each iteration of the outer for loop, and stating the number of inversions present in the array at that point. [10 pts]
  2. Referring to InsertionSort:
    - (a) Prove using mathematical induction that for all  $0 \leq i \leq n - 1$  that after the for loop ends each  $i$  that the  $0, \dots, i$  elements of the list are sorted. [15 pts]
    - (b) Use your answer to (a) to conclude mathematically that InsertionSort works. [5 pts]
  3. *Gnome Sort* is a cross between Insertion Sort and Bubble Sort. It behaves like Insertion Sort, but uses progressive downwards Bubble Sort-style swaps instead of storing the value to be inserted. Here is the pseudocode of Gnome Sort: [15 pts]
- 

```
\\ PRE: A is a list of length n.
for i = 1 to n-1
    j = i
    while (j > 0 and A[j] < A[j - 1])
        swap A[j] and A[j - 1]
        j = j - 1
    end
end
\\ POST: A is sorted.
```

---

Assume assignments take  $A$  time, conditional checks take  $C$  time, and swaps take  $S$  time, all of which are constants. Compute the average-case runtime of Gnome Sort, simplify to a polynomial in terms of  $n$ , and state its time complexity. (Ignore the maintenance of the for loop. You may need to first compute the runtime of Gnome Sort in terms of both  $n$  and the number of inversions  $I$  of the input list, and then use what you know about the expected number of inversions.)

4. Show the steps of binary search when looking for the value 17 in the list: [10 pts]

{-3, 4, 7, 17, 20, 30, 40, 51, 105, 760}

At each step give the value of  $C$  and how the comparisons update  $L$  and  $R$ . Inside the while loop how many comparisons are made?