CPSC 320 Notes, Asymptotic Analysis

1 Comparing Orders of Growth for Functions

For each of the functions below, give the best Θ bound you can find and then arrange these functions by increasing order of growth.

$n + n^2$	2^n
$55n + 4$	$1.5n \lg n$
$n!$	$\ln n$
$2n \log(n^2)$	$\frac{n}{\log n}$
$(n \lg n)(n + 1)$	$(n + 1)!$

1.6^{2n} *tricky, but doable!*

2 Functions/Orders of Growth for Code

Give and briefly **justify** good Θ bounds on the worst-case running time of each of these pseudocode snippets dealing with an array A of length n. Note: we use 1-based indexing; so, the legal indexing of A is: $A[1], A[2], \ldots, A[n]$.

Finding the maximum in a list:

```
Let max = -infinity
For each element a in A:
  If max < a:
    Set max to a
Return max
```

"Median-of-three" computation:

```
Let first = A[1]
Let last = A[n]
Let middle = A[floor(n/2)]

If first <= middle And middle <= last:
  return middle
Else If middle <= first And first <= last:
  return first
Else:
  return last
```

Counting inversions:

```
Let inversions = 0
For each index i from 1 to n:
  For each index j from (i+1) to n:
    If a[i] > a[j]:
      Increment inversions
Return inversions
```

3 Progress Measures for While Loops

Assume that `FindNeighboringInversion(A)` consumes an array `A` and returns an index `i` such that `A[i] > A[i+1]` or returns `-1` if no such inversion exists. Let's work out a bound on the number of iterations of the loop below in terms of n, the length of the array `A`.

```
Let i = FindNeighboringInversion(A)
While i >= 0:
  Swap A[i] and A[i+1]
  Set i to FindNeighboringInversion(A)
```

1. **Give and work through two small inputs** that will be useful for studying the algorithm. (What is "useful"? Try to find one that is simply common/representative and one that really stresses the algorithm.)

2. **Define an inversion** (not just a neighboring one), and **prove that if an inversion exists at all, a neighboring inversion exists**.

3. Give **upper- and lower-bounds on the number of inversions** in A.

4. Give a "measure of progress" for each iteration of the loop in terms of inversions. (I.e., how can we measure that we're making progress toward terminating the loop?)

5. Give an upper-bound on the number of iterations the loop could take.

6. Prove that this algorithm sorts the array A (i.e., removes all inversions from the array).