CPSC 320 Notes, Asymptotic Analysis

January 12, 2015

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. (Some of the later ones are especially tricky!)

$$\begin{array}{ll}
2^n & n + n^2 \\
1.5n \lg n & 55n + 4 \\
\ln n & n! \\
(n+1)! & (n \lg n)(n+1) \\
2n \log(n^2) & 1.6^{2n} \\
\frac{n}{\log n} & \sqrt{n}^{\sqrt{n}}
\end{array}$$

2 Functions/Orders of Growth for Code

Give good Θ bounds on the worst-case running time of each of these pseudocode snippets dealing with an array A of length 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[length of A]
Let middle = A[floor((length of A)/2)]

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

Counting inversions:

```
Let inversions = 0
For each index i from 1 to length of A:
  For each index j from (i+1) to length of A:
    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 index = FindNeighboringInversion(A)
While index > 0:
  Swap A[i] and A[i+1]
  Set index to FindNeighboringInversion(A)
```

1. First, prove that if an array has an inversion (two legal array indices i and j such that $i > j$ but $A[i] < A[j]$), then it has a neighboring inversion (an inversion in which the second index is one greater than the first).

2. Prove that the swap in the loop removes an inversion but does not **introduce** an inversion.

3. Give a "measure of progress" for each iteration of the loop in terms of inversions.

4. Give an upper-bound on the number of possible inversions in the array.

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

6. Prove that this algorithm sorts the array A.

4 Challenge Problem

Imagine that rather than `FindNeighboringInversion`, we'd used `FindInversion`, which returns two arbitrary indices `(i, j)` such that `i < j` but `A[i] > A[j]` and then in our loop swapped `A[i]` and `A[j]`. Could the loop run forever? If it terminates, would the array be sorted? Can you upper- and lower-bound the loop's runtime?