CS310: ALGORITHMS AND DATA STRUCTURES

Why/ Why not Worst Case?

- Worst case T(n) is useful because
  - provides a guarantee about running time performance
  - guarantee might be needed in some real time systems
- Our goal will be to design algorithms with good worst case performance but worse case input…
  - may not appear very frequently
  - performance on a typical input may be significantly better than the performance on the worst case input
  - may also not be easy for a programmer to implement

Why/ Why not Other Cases?

- Average-case T(n) is
  - a good predictor of performance on a typical input
  - but it is harder to compute mathematically

- Best-case T(n) is not useful, because
  - a slow algorithm (high worst and average case) can be made to run very fast on some special inputs
- Best case T(n) can be sometimes useful when
  - Worst Case T(n) of some algorithms might be less than Best Case T(n) of another algorithm
  - If you have input close to the output (e.g. almost sorted), then knowledge of best case T(n) helps

Example: Deciding Upon an Algorithm

- Let A1, A2 be two algorithms for the same problem
  - A1 = very good worst-case performance, good avg case performance
  - A2 = very bad worst-case performance, very good avg case performance
- Will you prefer A1 or A2?
- A1 may also be significantly more complex, harder to implement

Problem – 1

- Let A[1…n] be an array of distinct numbers. If i<j and A[i] > A[j], then the pair (i, j) is called an inversion of A
  - List all the inversions of <2,3,8,6,1>
  - What array from the set {1,2,…,n} has the most inversions? How many does it have?
  - What is the relationship between the number of inversions in the input array and insertion sort?
  - What is the relationship between running time of insertion sort and the number of inversions in an array?

Problem – 2

- If all permutations are equally likely, what is the expected number of inversions in a randomly chosen permutation of 1,2,…n?
- n (n-1)/2
- n(n-1)/4
- n(n+1)/4
- $2n\log_2 n$