3-0235 — 50 SHEETS — 5 SQUARES
3-0236 — 100 SHEETS — 5 SQUARES
3-0237 — 200 SHEETS — 5 SQUARES
3-0137 — 200 SHEETS — FILLER
COMET

GUIDING PRINCIPLE #1: FOCUS ON WORST-CASE ANALYSIS

By focusing on worst-case analysis our running time bound holds for every input of length n.

AVERAGE-CASE ANALYSIS - Evaluate an algorithm's average running time by accounting for relative frequency of different input lengths.
* Requires domain specific knowledge

BENCHMARK ANALYSIS - Evaluate an algorithm's running time to execute a pre-specified/standardized set of inputs the algorithm will most likely see.
* Requires domain specific knowledge.

WORST-CASE ANALYSIS - Evaluate an algorithm as if every time it runs the algorithm will take its maximum amount of time to execute.

Particularly appropriate for "general-purpose" problems in which we do not know what the inputs to the algorithm will be beforehand.

Worst-case is mathematically easier to analyze

GUIDING PRINCIPLE #2: DO NOT WORRY ABOUT CONSTANT FACTORS OR LOWER ORDER TERMS

JUSTIFICATIONS:
1) Much easier mathematically to remove constant factors and lower order terms
2) Constant factors will depend on and change based on computer architecture, the compiler, programmer, and so on.
3) Lose very little predictive power. We lose some granularity of analysis, but overall the higher order terms will give a strong snapshot as to the speed of the algorithm.

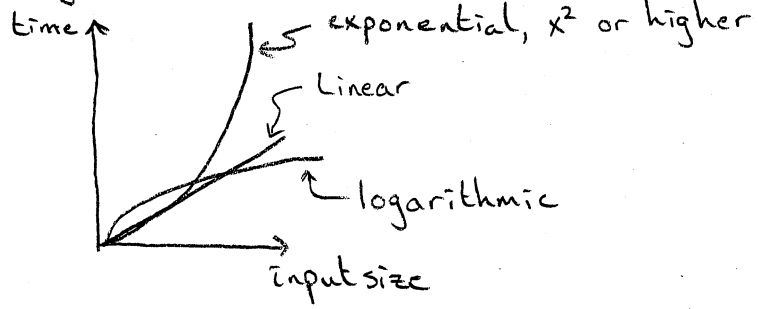
(con't on next sheet)

3-0235 — 50 SHEETS — 5 SQUARES
3-0236 — 100 SHEETS — 5 SQUARES
3-0237 — 200 SHEETS — 5 SQUARES
3-0137 — 200 SHEETS — FILLER
COMET

GUIDING PRINCIPLE #3: USE ASYMPTOTIC ANALYSIS

Asymptotic analysis uses large inputs to the algorithm to determine if the algorithm has run-time limits it cannot improve upon.

For example, x^2 and higher order terms grow much more quickly than linear or lower order terms with increasing input size.



Therefore, when n is large an algorithm of runtime n^2 is slower than an algorithm of runtime n.

While there will be instances when a higher order runtime algorithm will perform better than a lower order algorithm these instances will all be when the input size is small ($n < 100$) due to constant factors influencing the higher order terms.

You can optimize a solution to a problem by switching algorithms after reaching a given input size, but this should only be after optimizing for the large number of inputs. Otherwise, the optimization efforts would be premature.

WHAT IS A FAST ALGORITHM?

A fast algorithm is an algorithm whose worst-case running time grows slowly as input size increases.

Quadratic run time better than exponential n^2 vs. 2^n

Linear runtime is better than quadratic n vs n^2

Logarithmic runtime is better than linear $\lg n$ vs n

Constant runtime is better than logarithmic 10 vs $\lg n$