Lecture 6

Efficiency of Algorithms (2) 
(S&G, ch.3)

Read S&G chapter 4

Sorting

- One of the most common activities of a computer is sorting data.
- Arranging data into numerical or alphabetical order for purposes of
  - Reports by category
  - Summarizing data
  - Searching data

Sorting (2)

- Given: n, N_1, N_2, ..., N_n
- Want: Rearrangement M_1, M_2, ..., M_n where M_1 ≤ ... ≤ M_n according to the appropriate understanding of ≤.
- There are many well-known algorithms for doing this. Preference may be due to
  - List size
  - Degree of order already present
  - Ease of programming
  - Program available

Sorting - Selection Sort

- Strategy:
  - Find the largest element in list
  - Swap it with last element in list
  - Find next largest
  - Swap it with next to last element in list
  - Etc
- Variables:
  - U - Marker for unsorted section
  - L - Position of largest so far
  - C - Current position

Selection Sort - The Algorithm

Set U to n
Repeat until U = 1
  Set L to 1
  Set C to 2
  Repeat until C > U
    If N[C] > N[L] then
      Set L to C
    Set C to C + 1
  Exchange N[L] and N[U]
Set U to U - 1
Stop
Efficiency Analysis of Selection Sort

- Makes $n - 1$ passes through the list
- Each pass uses Search For Largest, on the unsorted part of the list, to find the largest element
- If the unsorted part has length $U$, Search For Largest takes $U - 1$ comparisons

Efficiency of Selection Sort (2)

- To sort $n$ element list:
  - $n - 1$ comparisons to find largest
  - $n - 2$ comparisons to find second largest
  - ... 
  - 2 comparisons to find largest of last three
  - 1 comparisons to find largest of last two
- Total number of comparisons:
  $$\frac{(n-1)n}{2}$$
A Useful Formula

In analysis of algorithms, we often encounter a sum of consecutive integers:

\[ S = 1 + 2 + \cdots + N \]

Note that we can write:

\[ S = N + (N-1) + \cdots + 2 + 1 \]

Add \( 2S \):

\[ 2S = N + 1 + N + (N+1) + \cdots + N \]

Hence, \( S = N(N+1)/2 \)

Best, Worst, & Average

• Suppose there are \( N \) possible problem instances and that \( T_k \) = time to solve instance \( k \)
• Worst case: \( W = \max\{T_1, T_2, \ldots, T_N\} \)
• Best case: \( B = \min\{T_1, T_2, \ldots, T_N\} \)
• Average case: \( A = (T_1 + T_2 + \cdots + T_N)/N \)
  (if they are equally likely)
• If \( p_k \) is the probability of instance \( k \), then weighted average \( M = p_1T_1 + p_2T_2 + \cdots + p_NT_N \)

Motivation for Asymptotic Complexity

• We are interested in comparing the efficiency of various algorithms, independent of computer running them
• Therefore, we choose to ignore:
  – the absolute time taken by the instructions
  – constant initialization overhead
  – time taken by the less frequently executed instructions

Motivation for Asymptotic Complexity (2)

• Not concerned about efficiency for small problems (they’re all quick enough)
• Usually interested in using on large problems
• We investigate how resource usage varies on progressively larger problems
• Which is asymptotically better, i.e., for sufficiently large problems

Quadratic vs. Linear Growth

• On big enough problem, quadratic algorithm takes more time than linear algorithm
Equivalent Complexity

- For our purposes:
  - an algorithm taking time \(10n\) sec. is as good as one taking \(2n\) sec.
  - because absolute time for individual instructions doesn’t matter
- algorithm taking \(2n + 10\) sec. is as good as one taking \(2n\) sec.
  - because fixed overhead doesn’t matter
- algorithm taking \(n^2 + 2n\) sec. is as good as one taking \(n^2\) sec.
  - less frequently executed instructions don’t matter

Complexity Classes

- No growth: \(\Theta(1)\)
- Linear growth: \(\Theta(n)\)
- Quadratic growth: \(\Theta(n^2)\)
- Cubic growth: \(\Theta(n^3)\)
- Polynomial growth: \(\Theta(n^k)\) for any \(k\)
- Exponential growth: \(\Theta(k^n)\) for any \(k\)

\(\Theta\) (theta) is called the “exact order” of a function

Combinatorial Explosion

- Consider a game with \(k\) possible moves on each play
- Therefore:
  - \(k\) possible first moves
  - \(k^2\) possible replies
  - \(k^3\) possible replies to the replies, etc.
- In general, must consider \(k^n\) possibilities to look \(n\) moves ahead (exponential algorithm)
- “Brute force” solutions to many problems lead to “combinatorial explosion”

Intractable Problems

- For some problems, all known algorithms are exponential-time
- No known polynomial-time algorithms — not even \(\Theta(n^{100})\)!
- Sometimes there are efficient \textit{approximation algorithms}
  - give good, but not perfect answer
  - may be good enough for practical purposes

Limitations of Asymptotic Complexity

- Note that 1,000,000 \(n\) is \(\Theta(n)\),
  - but 0.000001 \(n^2\) is \(\Theta(n^2)\)
- Paradox: 1,000,000 \(n\) hrs >> 0.000001 \(n^2\) hrs
  - (for \(n < 10^{12}\)),
  - but \(\Theta(n)\) is “more efficient” than \(\Theta(n^2)\)
- Not useful if concerned with absolute real-time constraints (e.g., must respond in 2 sec.)
- Not useful if concerned with fixed size or bounded problems (e.g., \(n < 1,000,000\))
- Sometimes, “the constants matter”