Definitions of estimation tools: big-\(O\), \(\Omega\), \(\Theta\).

- the inequalities between functions defining these must hold up to a positive constant factor. For instance, \(f(n) = O(g(n))\) requires \(f(n) \leq c \cdot g(n)\), for some positive \(c\). Often the first challenge in proving a big-\(O\) relationship is guessing \(c\), i.e. finding a \(c\) that will work.

- The inequalities may be wrong for a few small values of \(n\). It is only required that they be true for all values of \(n\) after some threshold. For instance, \(n = O(n \log(n))\) even though when \(n = 1\), we see that \(n > c \cdot n \log(n)\), regardless of \(c\). On the other hand – and what counts – for all \(n \geq 2\), we see that \(n \leq n \log(n)\) is true because \(\log(n) \geq 1\) for \(n \geq 2\).

Master theorem and Muster theorem

- Master theorem (applies to recurrences diving down to \(n/b\)) and Muster theorem (applies to recurrences stepping down to \(n - b\))

- divide and conquer sorting (merge sort) (p50)

- lower bound for sorting (p51)

- divide and conquer selection (randomized median) (homework problem)

Themes: divide and conquer algorithms may be organized around applicable case of Master theorem (see algorithms list below).

Algorithms to which Master theorem applies:

- Case 1 of Master theorem: selection (randomized median)

- Case 2 of Master theorem: merge sort, binary search

- Case 3 of Master theorem: stooge sort has recurrence \(T(n) = 3T((2/3)n)\), for \(n > 2\).

Algorithms to which Muster theorem applies:

- Case 1 of Muster theorem: \(a < 1\) doesn’t come up much.

- Case 2 of Muster theorem: insert sort has recurrence \(T(n) = T(n-1) + O(n)\) [ insert sort on the first \(n-1\) followed by insert the last elt.]

- Case 3 of Muster theorem applies to many exponential algorithms: For instance on a robot arm tour variant. The problem as to find a minimal cost tour through all the points of a weighted graph in which all nodes have degree at most \(k\). Pick a point \(a\), remove it, and, for each pair \(b,c\) of neighbors, combine them into one point. find the optimal tour through the resulting graph, add the cost of going from \(b\) to \(a\) to \(c\). Take the minimal of all those tours. The recurrence is \(T(n) = (k)(k-1)/2 \cdot T(n-2) + O(n)\). Muster theorem says \(T(n)\) is \(O(nk^n). [((k^2)^{n/2} = k^n].\)

Graphs

- Breadth first search (bfs) in (undirected) graphs and digraphs (directed graphs).

- Single source shortest path via bfs.
Themes: basic graph representation and terminology, bfs is basis of solving some problems.

Chapter 6.1: Minimal Spanning Trees. Prim’s (uses priority queue) and Kruskal’s (uses union-find) algorithms

Chapter 6.3: Dijkstra’s algorithm (assuming a priority queue) for single source shortest paths in weighted graphs.

Overall: Things to know about each algorithm:

- Why is it correct (vis a vis it’s input/output specification)? What are the theorems and properties used to explain it’s workings?

- What measure, n, of it’s input is used as basis for analysis (for instance, n = bound on number of bits in numbers, or n = size of an array)?

- What formula (function of that n) estimates it’s runtime? Usually the formula is a recurrence relation.

- What is the solution of that formula/recurrence up to big O or Θ?

Kinds of questions: multiple choice, short answer, analyze given algorithm, write (simple) algorithm.