Homework 1

Zach Neveu

May 10, 2019

1. Bubble Sort
   (a) Best-case running time: If the list is already sorted, Bubble Sort only requires a single pass, or $O(n)$ time to verify this.
   (b) Worst-case running time: If the list is reverse-sorted, then every item will need to be swapped on each pass, which makes the worst case runtime $O(n^2)$
2. Sequential Search
   (a) Best case running time: Desired item is first in array, leading to a runtime of $O(1)$
   (b) Worst case running time: Desired item is last in array, leading to a runtime of $O(n)$
3. Proofs
   (a) **True**: $f(n) = O(g(n))$ means that $f(n)$ is upper-bounded by $g(n)$. Likewise, $g(n) = O(h(n))$ means that $g(n)$ is upper-bounded by $h(n)$. By the transitive property, $f(n) \leq g(n) \leq h(n)$, so $f(n) = O(h(n))$.
   (b) **False**: If $f(n)$ is linear search (worst case runtime $O(n)$) and $g(n)$ is bubble sort (worst case runtime $O(n^2)$), then $f(n) = O(g(n))$, but $g(n) \neq O(f(n))$
4. Optimization vs. Decision Problems
   (a) Minimum Spanning Tree in a weighted graph
      - Decision: Given a weighted graph, determine whether a spanning tree exists for which the sum of the weights of the included edges is less than k.
      - Optimization: Given a weighted graph, find the spanning tree which minimized the sum of the weights of the included edges.

   (b) Maximum Matching in a Graph
      - Decision: Given a graph, determine whether a matching exists which contains at least k nodes.
      - Optimization: Given a graph, find the matching containing the largest number of nodes possible.

   (c) Shortest Path
      - Decision: Given two vertices in a weighted graph, determine whether a path between them exists for which the sum of the included edge weights is less thank k.
      - Optimization: Given two vertices of a weighted graph, find the path between them which has the smallest sum of included edge weights.
5. Definitions

(a) Heuristic: An approximate method of solving a problem which is not guaranteed to find an optimal solution

(b) Polynomial-time algorithm: An algorithm whos runtime grows as a polynomial function of the instance size

(c) Intractable Problem: A problem which provably cannot be solved by any polynomial-time algorithm.

(d) Complexity class P: All problems which can be solved by a polynomial-time algorithm.