Informatics 2 – Introduction to Algorithms and Data Structures

Tutorial 5

1. In lectures we showed that the worst-case running-time for QuickSort is $\Omega(n^2)$. We mentioned without proof that the average-case running-time is $\Theta(n \lg n)$.

   In this question we show that the best-case running-time of QuickSort is $\Theta(n \lg n)$.

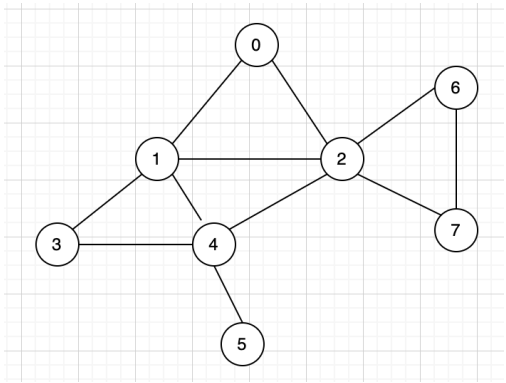
   (a) Try to give *an actual example* of an input array where QuickSort takes only $O(n \lg n)$ time to sort. This is asking what kind of pattern will cause the array to repeatedly be split (roughly) in half. I'm not looking for something very precise here - if you can get the answer for $n = 15$ that would be very good.

   (b) (a bit hard) First show that the *best case* running time will always be at least $cn \lg n$, for some constant $c > 0$.

   [**Hint:** For this part, think about the recursion tree for QuickSort, and the total amount of work done across a level.]

   For both parts of this question it will help to work with an n of the form $2^h - 1$ to help ensure equal splits (plus an excluded item at "*split*") can be achieved recursively.

2. Consider the following graph:



   (a) Write the adjacency matrix and the adjacency list representation for the graph. For the adjacency list (for this example, for consistency wrt solutions) we ask you to consider the neighbours of a vertex in ascending numerical order in the list for each node.

(b) Run Breadth-First Search (BFS) on G starting from node 0. Explain the steps of BFS and note the level that each node will be assigned to during the execution of BFS. Write down the spanning tree produced by BFS in adjacency list representation.

(c) Run Depth-First Search (DFS) on G starting from node 0. Explain the steps of DFS and note the order in which the nodes will be explored by DFS. Write down the spanning tree produced by DFS in adjacency list representation.

(d) Compare the two spanning trees produced by DFS and BFS starting from node 0 respectively. What do you observe?

3. Prove the following property for the layers produced by BFS: for any edge (u, v), either u and v are in the same layer of the "breadth-first tree", or else $|L(u) - L(v)| = 1$ where $L(x)$ is the layer of node x.

4. Suppose we are given an undirected graph $G = (V, E)$ and asked to determine whether the graph is *bipartite* - that is, whether V can be partitioned into two subsets $V = V_1 \uplus V_2$ such that every edge $e = (u, v)$ has one endpoint in V_1 and one endpoint in V_2.

   Show how to answer this question in $O(n + m)$ time.

5. A directed graph $G = (V, E)$ is said to be *acyclic* (sometimes called a *dag*)if there is no subset of vertices which forms a directed cycle in G. It is well-known in graph theory that if a graph is *acyclic* (sometimes called a *dag*) then there must be some reordering of V so that for every edge $(u, v) \in E$, u appears before v in the ordering. Such a recording is called a *topological sort*.

   Consider the following method for performing *topological sorting* on a directed acyclic graph $G = (V, E)$: repeatedly find a vertex of in-degree 0, output it, and then remove it and all of its outgoing edges from the graph.

   (a) How can you implement this so that the entire algorithm will be $O(|V| + |E|)$?

   Hint: Note that in regard to in-degree, we only need to track the *number* of incoming edges to each v.

   (b) How will you detect that the graph has cycles?

6. (*) Design an algorithm to sort and return the least k elements of a list using the same `partition` subroutine of `quickSort`. How does the worst case execution time of this algorithm compare to that of `quickSort`?