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. In both breadth-first search bfs and depth-first search dfs (together with their worker methods bfsFromVertex and dfsFromVertex respectively) we can represent our collection of found vertices as a set of subtrees, each with a particular root at level 0, and with the found vertices at levels 1, 2, ....

We consider these two graph traversal methods on directed graphs.

(a) Argue that for a directed graph $G = (V, E)$, with vertex $v$ being reachable from the vertex $r$, that the order of considering vertices at the top-level of bfs may affect whether $v$ will belong to the tree generated by bfsFromVertex($r$), but if $v$ is generated as part of bfsFromVertex($r$), it will always be at the same “level” (same distance from $r$).

(b) Show by example that for a directed graph $G = (V, E)$, with vertex $v$ being reachable from the vertex $r$, that the order of edges in the adjacency lists may affect the level at which $v$ is generated in the tree generated by dfsFromVertex($r$) (assuming $r$ is as-yet unvisited).

3. 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 $V = V_1 \cup V_2$ 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.
4. 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?