I/O-Efficient Algorithms: Prerequisites test

Question 1.

*InsertionSort* and *MergeSort* are both algorithms to sort a set of \( n \) numbers. The worst-case running time of *InsertionSort* is \( \Theta(n^2) \), and the worst-case running time of *MergeSort* is \( \Theta(n \log n) \). Somebody measured the actual running time of both algorithms on his computer, and found that for \( n < 100 \), *InsertionSort* is faster, and for \( n \geq 100 \), *MergeSort* is faster. Therefore she suggests the following algorithm:

**Algorithm** *CombinationSort*(\( A \))

1. \( \triangleright A[1..n] \) is an array of numbers
2. \( n \leftarrow \text{length}[A] \)
3. if \( n < 100 \)
4. then *InsertionSort*(\( A \))
5. else *MergeSort*(\( A \))

Is the worst-case running time of *CombinationSort* \( \Theta(n^2) \) or \( \Theta(n \log n) \)?

Question 2.

Let \( G \) be the undirected, unweighted graph shown above, in which every vertex has degree three.

(a) Suppose we run a *BFS* on \( G \), given in adjacency list representation, with vertex \( u \) as start vertex. What is the smallest possible number of vertices visited before vertex \( v \) is visited? What is the largest possible number of vertices visited before vertex \( v \) is visited?

(b) Suppose we run a *DFS* on \( G \), given in adjacency list representation, with vertex \( u \) as start vertex. What is the smallest possible number of vertices visited before vertex \( v \) is visited? What is the largest possible number of vertices visited before vertex \( v \) is visited?

(We say that a vertex is *visited* as soon as we access its adjacency list for the first time. Do not forget to include \( u \) in the count! Do *not* include \( v \).)
Question 3.

Rank the following functions by order of growth; that is, find an arrangement \(g_1, g_2, \ldots, g_{10}\) of the functions satisfying \(g_1 = O(g_2), g_2 = O(g_3), \ldots, g_9 = O(g_{10})\). Indicate any pairs of functions \(f(n)\) and \(g(n)\) such that \(f(n) = \Theta(g(n))\).

\[
\log^2 n \quad n \log n \quad 1 \quad n^3 \quad n^{-1} \quad n^{\log n} \quad n^{\log 8} \quad n \quad 1/\sqrt{n} \quad \sqrt{n}
\]

Question 4.

Solve the following recurrences:

(a) \(T(n) = O(n) + 2T(n/2)\)
(b) \(T(n) = O(n) + T(n/2)\)
(c) \(T(n) = O(1) + T(n/2)\)

Question 5.

Consider the following algorithm:

**Algorithm** DoSomething \((G, w, s)\)
1. \(\triangleright G\) is a graph given in adjacency list representation; let \(V[G]\) be its vertices
2. \(\triangleright w : V[G] \times V[G] \rightarrow \mathbb{R}^+\) is a weight function that can be evaluated in \(O(1)\) time
3. \(\triangleright s\) is a vertex of \(G\)
4. Initialize an empty priority queue \(Q\)
5. for each vertex \(v \in V[G]\)
6. \hspace{1em} do \(d[v] \leftarrow \infty\)
7. \hspace{1em} Insert \(v\) in \(Q\) with key \(d[v]\)
8. \(d[s] \leftarrow 0\)
9. Decrease the key of \(s\) in \(Q\) to 0
10. while \(Q\) not empty
11. \hspace{1em} do \(u \leftarrow\) ExtractMin \((Q)\)
12. \hspace{1em} for each vertex \(v \in AdjacencyList[u]\)
13. \hspace{2em} do if \(d[v] > d[u] + w(u, v)\)
14. \hspace{2em} then \(d[v] \leftarrow d[u] + w(u, v)\)
15. \hspace{2em} Decrease the key of \(v\) in \(Q\) to \(d[v]\)

Assume that the priority queue is implemented as an ordinary heap.

(a) Analyse the worst-case running time of this algorithm and express it in \(O\)-notation. Give the best bound you know how to prove, in terms of relevant properties of the input. You may use the fact that every vertex is extracted from \(Q\) exactly once.

(b) Do you know what the algorithm computes?