TDDB56 & TTIT33
DALG part of the exam 2006-12-16
Outline of the proposed solutions for selected problems.

1. (a) The pseudocode describes the Insertion Sort algorithm. The worst case is when the data is sorted in the decreasing order. In that case for \( i = 2, \ldots, n \) the body of the inner loop is executed, respectively, \( 1, 2, \ldots, n - 1 \) times. Hence the total time is bounded by \( t(1 + 2 + 3 + \ldots + n - 1) \) where \( t \) is a constant. Hence the worst case time complexity is \( O(n^2) \). The best case is when the data is sorted and the data does not include equal elements. In that case the condition \( S[j] \geq x \) never holds so the body of the inner loop is never executed. In that case the time complexity is thus \( O(n) \).

(b) The big-O notation characterizes the asymptotic behaviour of a function. For a small size of input data (smaller than \( n_0 \) in the definition of the \( O \) relation) the asymptotic time complexity provides no information. The measurement might have been done on such data.

- The input data could have been chosen in such a way that they were close to the best case data for A and close to the worst case data for B.
- The big-O notation gives only the upper bound for time complexity. If A is in \( O(n^2) \) it may also be in e.g. \( O(n \log n) \).

The problem discusses comparison of running time of two algorithms on the same data. Thus it implicitly assumes that the algorithms were run on the same machine. So the guess that A was run on faster machine and B on a slower one is not a relevant explanation. Similarly, a guess that one of the algorithms might have required communication with external memory while the other did not would make comparison of running time irrelevant.

2. (a) The table and diagram provided illustrate the comparison of the two algorithms with different data sets. The table shows the input data for each algorithm and the output is indicated in the respective columns. The diagram visually represents the same data, with nodes and edges indicating the flow of the algorithms.
Notice that we implement a Map, not a Dictionary, thus 6b is to be replaced by 6d; the key 6 can only appear once in the hash table! 8e can be inserted in the position marked as deleted, but before that one has to check that the key 8 does not appear in the table. As shown above one can instead insert 8e in a first free position found by using the increment 2 determined by $h_2$. Recall that $h_2$ in double hashing gives an increment to be iteratively added to the position determined by $h_1$ until a free position in the hash table is found.

(b) (3)

• The tree

\[
\begin{array}{c}
52(-1) \\
/ \ \\
18(-1) 70(1) \\
\ \ / \ \\
40 62(0) 74 \\
/ \\
59 65
\end{array}
\]

is an AVL tree, since the balance of each node is 0, 1 or -1.

• Insert(50) final result:

\[
\begin{array}{c}
T1 \\
52 \\
/ \\
40 70 \\
/ \\
18 50 62 74 \\
/ \\
59 65
\end{array}
\]

Insert(64) The critical node is 67. Insertion will include 53 as the left child of 61. After that re-balancing is needed. It is achieved by two rotations: left rotation around 49:

\[
\begin{array}{c}
52 \\
/ \\
18 70 \\
\ \ / \ \\
40 65 74 \\
/ \\
62 \\
/ \\
59 64
\end{array}
\]

and right rotation around 67. Final result:
Delete(70) First step replaces 70 by its inorder successor 74:

This is not an AVL tree; the balance of 74 is to be restored by right rotation. The obtained tree is an AVL tree.

An alternative solution is to replace 70 by its inorder predecessor 65, in which case no rebalancing is needed.

The above solution shows application of each the operations to the original tree. If each of the operation is applied to the result of the previous one then: (1) $T'_2$ obtained by insertion of 64 to $T_1$ has the left subtree as in $T_1$ and right subtree as in $T_2$; (2) Removal of 70 from $T'_2$ does not destroy the AVL property and does not require any re-balancing.

(c) The final result:

3. (6 p)

(a) Merge Sort

| 9 4 15 8 | 16 7 12 5 |
| 9 4 15 8 | 16 7 12 5 |
| 9 4 15 8 | 16 7 12 5 |
| 4 9 8 15 | 7 16 5 12 |
| 4 8 9 15 | 5 7 12 16 |
| 4 5 7 8 9 12 15 16 |
Merge sort is stable: partition does not change the relative order of data with the same keys, the merge procedure in case of items with equal key takes first the left item, hence does not change the original relative order.

(b) Radix Sort

\[
\begin{array}{c}
\text{aba, dab, bbc, ccc, abd, cad} \\
\text{[aba, abd, bbc, cad, ccc, dab]}
\end{array}
\]

(c) Heap Sort (the trees not shown)

\[
\begin{array}{c}
\text{Heapification} \\
\text{Sorting}
\end{array}
\begin{array}{c}
\begin{bmatrix} 9 & 7 & 11 & 8 & 10 \\ 11 & 7 & 9 & 8 & 10 \\ 11 & 8 & 9 & 7 & 10 \\ 11 & 10 & 9 & 7 & 8 \\ 8 & 10 & 9 & 7 & 11 \\ 10 & 8 & 9 & 7 & 11 \\
7 & 8 & 9 & 10 & 11 \\ 9 & 8 & 7 & 10 & 11 \\ 7 & 8 & 9 & 10 & 11 \\ 8 & 7 & 9 & 10 & 11 \\ 7 & 8 & 9 & 10 & 11 
\end{bmatrix}
\end{array}
\]

(d) Quick Select we choose the first element as the pivot

\[
\begin{array}{c}
\begin{array}{c}
S \\
E \\
G
\end{array}
\begin{array}{c}
\begin{bmatrix} 5 & 4 & 10 & 9 & 11 & 13 \\ 10 & 8 & 9 & 7 & 11 \\
7 & 8 & 9 & 10 & 11 \\ 9 & 8 & 7 & 10 & 11 \\ 7 & 8 & 9 & 10 & 11 \\ 8 & 7 & 9 & 10 & 11 \\ 7 & 8 & 9 & 10 & 11 
\end{bmatrix}
\end{array}
\end{array}
\]

4.

(a) e.g.: \(a, b, c, e, d\) According to the definitions in p.611 of the course book in this DFS we get back edges: \((c, a), (e, a)\) and the cross edge \((d, c)\). There are no forward edges.

(b) We refer to the pseudocode of DFS in the course book. The algorithm checks incident edges of every vertex of the graph. If the graph is represented by the adjacency matrix this requires for every vertex traversal of the whole respective row of the matrix. For a graph with \(n\) vertices traversal of the row takes \(O(n)\) time. Consequently the algorithm executes in time \(O(n^2)\).