Outline of the proposed solutions for selected problems.

1. (5 p)

(a) \(\log n, n, n^2, n^2 + \log n, n - n^3 + 7n^5, 2^n, n!\)

Justification is omitted.

(b) Denote by \(T(n)\) the time needed for computing the value of the function \(F(n)\). \(F(0)\) and \(F(1)\) are computed in a constant time, respectively \(t_0\) and \(t_1\). For \(n > 1\) we get

\[
T(n) = T(n - 1) + T(n - 2) + c \quad \text{where } c \text{ is the time needed for adding two natural numbers.}
\]

Thus, for all \(n \geq 2\) we have: \(T(n) > 2T(n - 2)\). For odd \(n = 2m + 1\) we have thus

\[
T(n) > \min(t_0, t_1)2^m = \min(t_0, t_1)2^{(n-1)/2}.
\]

Similarly for even \(n > 1\) we get

\[
T(n) > \min(t_0, t_1)2^{(n-2)/2}.
\]

Hence \(T(n)\) is in \(\Omega(2^{\lceil n/2 \rceil})\).

(c) For \(n > 1\) the value of \(F(n)\) is used for computing \(F(n + 1) = F(n) + F(n - 1)\) and for computing \(F(n + 2) = F(n - 1) + F(n)\). Thus we can design an iterative algorithm which keeps and updates two values: the current value \(F(i)\) and the previous value \(F(i-1)\), for \(i = 2, 3, \ldots, n\). Thus the main loop for computing \(F(n)\) for \(n > 1\) could have a form

\[
\text{for } i \text{ from } 2 \text{ to } n
\]

\[
\quad \text{temp} \leftarrow \text{prev};
\]

\[
\quad \text{prev} \leftarrow \text{curr};
\]

\[
\quad \text{curr} \leftarrow \text{curr} + \text{temp};
\]

where on the exit from the loop the value of \(\text{curr}\) is \(F(n)\). Before entering the loop \(\text{curr}\) is to be initialized to 1 and \(\text{prev}\) - to 0. The loop including 3 constant time operations is executed \(n - 1\) times thus the algorithm executes in time \(\Theta(n)\).

2. (2 p)

(a) Hash table can be used to implement ADT Map as well as ADT Dictionary. In the case of ADT Map every data item in the hash table has a unique key. If a new data item with the same key is to be inserted the old value associated with this key in the table is updated. In the case of ADT Dictionary the hash table may include several data items with the same key (see [G&T] 4th ed. Section 9.3.2). The following solution assumes that the Hash table is used to implement a Dictionary. The solution for ADT Map differs in the last step.
Insert(15, c) - - - - 15c - - - - - - 
Insert(4, a) - - - - 15c 4a - - - - - - 
Insert(26, b) - - - - 15c 4a 26b - - - - 
Delete(15) - - - - 4a 26b - - - - - - 
Insert(5, d) - - - - 4a 26b 5d - - - - 
Insert(4, e) - - - - 4a 26b 5d 4e - - - 
(b) 

Insert(15, c) - - - - 15c - - - - - - 
Insert(4, a) - - - - 15c 4a - - - - - - 
Insert(26, b) - - - - 15c 4a - - 26b - - 
Delete(15) - - - - 15cx 4a - - 26b - - x-delete mark 
Insert(5, d) - - - - 15cx 4a - - 26b - 5d 
Insert(4, e) - - - - 4e 4a - - 26b - 5d 

3. (3 p) 
(a) 

• Insertion Sort:  
  best case data: sorted, e.g. 1 2 3 4 5  
  worst case data: reverse sorted, e.g. 5 4 3 2 1  

• Quick Sort - the efficiency depends on selection of the pivot.  
  best case data: the element selected as pivot at every stage of partition is the median  
  of the partitioned set  
  worst case data: the element selected as pivot at every stage is the maximal element  
  of the partitioned set (or the minimal element)  
  If the last element selected as pivot (see [G&T] 4th ed. Section 11.2):  
  example of best case data: 1 2 4 5 3  
  example of worst case data: 1 2 3 4 5  

• QuickSelect(S, k) - the efficiency depends on selection of the pivot for given k.  
  Discussion of best case/worst case data is only possible for the fixed way of pivot  
  selection. The best case data is if the selected pivot is equal to the k-th large  
  element of S. The worst case data is if the pivot is the the minimal element of S  
  for k ≥ |S|/2 and the maximal element of S for k ≤ |S|/2. If the first element of a  
  set is always selected as a pivot then for k = 3:  
  example of best case S: 3 1 2 4 5  
  example of worst case S: 1 2 3 4 5
• True. The best case running time of Insertion Sort [G&T] 4th ed. p. 104 is $O(n)$ since for an already sorted array the condition of the internal while loop is always false. Thus the body of the external loop executes in constant time. As the external loop has always $n - 1$ iterations, Insertion Sort on sorted array runs in time $O(n)$. 

• As proved in p. 507 of [G&T] 4th ed. the worst case time of Quick Sort is $O(n^2)$. As $n^2$ is in $O(n^3)$ the statement is true by transitivity of big-O (see Lectures slide 1.33)

• False. Consider the algorithm in p. 530 of [G&T] 4th ed. The size $n$ of data is the length of $S$. Assume that all elements of $S$ are different. If the pivot is minimal element of $S$ and $k = n$ then $|L|=0$ and $|X|=1$. Thus the argument $G$ of the recursive call is of the size $n - 1$. The time needed for a partition of $S$ is proportional to $n$. As in this case the length at each recursive call reduces only by 1 the total time is in $\Theta(n^2)$. 

4. (4 p) 

(a) 

```
29
 / \
22 33
 / \ / \
15 25 32
 / \ / / \
8 19 23 26
```

Replace 22 by its inorder successor 23, or by its inorder predecessor 19. (slide 5.16.)

(b) To show that this is an AVL tree show that it fulfills the height-balance property [G&T] 4th ed. p. 429. AVL insertion is discussed in Section 10.2.1 of [G&T] and in slides 6.5-6.9. The critical node for insertion of 24 is the root (i.e. 29). According to the terminology of the slides the nodes $a$, $b$ and $c$ are respectively, 22, 25 and 29. Thus following the principle in slides 6.8-6.9 the result of insertion is:

```
25
 / \
22 29
 / \ / \ 
15 23 26 33
 / \ / / \
8 19 24 32
```

(c) (2,3) trees belong to the family of multiway search trees discussed in Section 10.4.1. and of (a,b)-trees discussed in 14.3.1 of [G&T] 4th ed. Insertion in such trees is discussed in the book for (2,4)-trees in Section 10.4.2 and for (2,3)-trees in lecture slides 6.15-6.16.
(d) As discussed in slide 7.16 one can consider min-heap with the minimal key in the root or max-heap with the maximal key in the root. The solution below shows min-heap.

```
  8
 /  \\
15   22
 /    \\
26   /    /   \
   19  25  23
   /   \\
 29   33  32
```