

Exam – extra part: solution

Tasks: solutions

Task 1 (4 points + 3 points)

a) (4 points)

```
public static void sort (int[] numbers)
{
    int    lastPos = numbers.length - 1;
    int    minPos = 0;
    for (int currentPos = 0; currentPos < lastPos; currentPos++)
    {
        minPos = currentPos;
        for (int p = currentPos + 1; p <= lastPos; p++)
            if (numbers[p] < numbers[minPos])
                minPos = p;

        int    n = numbers[currentPos];
        numbers[currentPos] = numbers[minPos];
        numbers[minPos] = n;
    }
}
```

b) (3 points)

In the worst case there is exactly 1 element exchange for each pass through the main loop. There are $n-1$ passes, and therefore the total number of exchanges is:

$$n - 1$$

This means that the worst case time complexity of the algorithm, in terms of the number of element exchanges, can be given by the following complexity function:

$$w(n) = n - 1$$

$$w(n) \in \Theta(n)$$

Task 2 (3 points)

Definition: the set $\Theta(n^2)$

A complexity function $f(n)$ belongs to the set $\Theta(n^2)$ if and only if there exists two real, positive constants c_1 and c_2 , and a non-negative integer N , for which the following inequalities holds for all $n \geq N$:

$$c_1 n^2 \leq f(n) \leq c_2 n^2$$

Task 3 (4 points + 3 points + 3 points)

a) (4 points)

```
// add adds a given element to the array
public void add (E e)
{
    if (!(countElements < elements.length))
    {
        Object[]    newElements = new Object[2 * elements.length];
        for (int pos = 0; pos < elements.length; pos++)
            newElements[pos] = elements[pos];
    }
}
```

```
        elements = newElements;
    }

    elements[countElements++] = e;
}
```

b) (3 points)

```
// get returns the element in a given position
public E get (int index)
{
    return (E) elements[index];
}
```

c) (3 points)

