Homework 2 - solutions

Problem 1 (1.2 - 1)
Solution:
The following function searches for the minimum in an array $A$ and returns the index where it first encounters it.

FIND_MIN($A$)
1    $\text{min} = A(1)$
2    $\text{min\_index} = 1$
3    for $i=2$ to $\text{length}(A)$
4        do if $\text{min} > A(i)$
5            then $\text{min} = A(i)$
6        $\text{min\_index} = i$
7    return $\text{min\_index}$

SELECTION_SORT($A$, $B$)
1    $n = \text{length}(A)$
2    for $i=1$ to $n$
3        do $\text{mi} = \text{FIND\_MIN}(A)$
4            $B(i) \leftarrow A(\text{mi})$
5            $A(\text{mi}) \leftarrow A(n)$
6    $\text{length}(A) \leftarrow \text{length}(A)-1$

The presence of for loops only indicates an easier analysis: the only difference between best and worst cases comes from the if statement on line 4 in FIND_MIN($A$): namely in the worst case $\Theta(n)$ more assignments are done, however the number of comparisons is the same independently of the data. If we call $T(n)$ the number of comparisons for sorting an array of size $n$ we
get, and by $C(n)$ the number of comparisons done in FIND_MIN for an array of size $n$:

$$T(n) = C(n) + C(n-1) + C(n-2) + \ldots + C(1) + C$$  \hspace{1cm} (1)

where $C$ is the summed up cost of lines 4, 5, 6 and the comparison in the for loop from line 2. Now the number of comparisons in FIND_MIN is

$$C(n) = n \text{(from line 3)} + (n-1) \text{(in line 4)} = 2n - 1.$$

After introducing this value in (1) we get:

$$T(n) = C(1) + C(2) + \ldots + C(n) + nC = (2 \cdot 1 - 1) + (2 \cdot 2 - 1) + \ldots + (2 \cdot n - 1) + nC$$

$$= 2(1 + 2 + \ldots + n) - n + nC = 2\frac{n(n + 1)}{2} - n + nC = n^2 + nC = \Theta(n^2)$$

Problem 2 (1.3 - 2)

Solution:

It is assumed that the 2 arrays $B$ and $C$ are sorted and will be merged into $A$:

MERGE(A, B, C)
1 s1 <- length(B)
2 s2 <- length(C)
3 c1 <- 1
4 c2 <- 1
5 i <- 1
6 while c1 <= s1 and c2 <= s2
7 do if B(c1) <= C(c2)
8 then A(i) <- B(c1)
9 c1 <- c1+1
10 else A(i) <- C(c2)
11 c2 <- c2+1
12 i <- i+1
13 if c1 = s1+1
14 then for j = 0 to s2-c2-1
15 do A(i+j) = C(c2+j)
16 else for j = 0 to s1-c1-1
17 do A(i+j) = B(c1+j)
Please note that some changes in the indices are needed when translating pseudo-code into C/C++ as array-indices in C/C++ start at 0.

Problem 3 (1.3 - 4)
A recursive version for INSERTION-SORT
Solution:

REC-INSERTION-SORT(A, n)
1 \textbf{if } n > 1 \\
2 \textbf{then } REC-INSERTION-SORT(A, n-1) \\
3 \textbf{end if}

In the worst case the cost of the insertion (lines 5 - 10) is $\Theta(n)$. The recursive relation is:

$$T(n) = T(n - 1) + \Theta(n)$$

Problem 4 (1.3 - 5)

Binary Search
Solution:
The following code searches for the \textbf{key} in the array \textbf{A} between and incl. \textbf{p} and \textbf{q}.

BINARY_SEARCH(A, p, q, key)
1 \textbf{if } p > q \\
2 \textbf{then return nil} \\
3 \textbf{else } r = \text{floor}((p+q)/2) \\
4 \textbf{if } key = A(r) \\
5 \textbf{then return } r \\
6 \textbf{else if } key < A(r)
7 then return BINARY_SEARCH(A, p, r-1, key)
8 else return BINARY_SEARCH(A, r+1, q, key)

In addition to a few (a constant number \( C \), independent of \( n = q - p \))
comparisons done in the lines 1 - 6, we have to solve one more problem of
size \( n/2 \), hence:

\[
T(n) = T(n/2) + C
\]
is the recursive equation for the work. We then write

\[
T(n) = T(n/2) + C = \frac{(T(n/2^2) + C) + C}{T(n/2)}
\]

\[
= T(n/2^2) + 2C = \frac{(T(n/2^3) + C) + 2C}{T(n/2^2)}
\]

\[
= T(n/2^3) + 3C
\]
We see he following pattern:

\[
T(n) = T \left( \frac{n}{2^i} \right) + iC
\]

(2)

A proof by induction - which we now skip - is necessary to show that indeed
the above holds for all \( i \). We choose \( i \) such that \( 2^i = n \), i.e. \( i = \lg n \) which
turns (2) into

\[
T(n) = T(1) + C \lg n = \Theta(\lg n)
\]