Algorithms: Solutions 5

```
                    X
                    X
                    X
                    X
                    X
                    X
                    X
                    X
                    X
                    X
                    X
                    X
                    X
                    X
                    X
                    X
                    X
number of           X     X
homeworks           X     X
                    X     X
                    X     X
              X     X     X
              X     X     X
              X  X  X     X
              X  X  X     X
     X  X     X  X  X     X
     X  X     X  X  X  X  X
     -----------------------
     3  4  5  6  7  8  9  10
             grades
```

Problem 1
Write an algorithm for finding the kth smallest element of a given array.

The following algorithm uses the same PARTITION procedure as QUICK-SORT. The average-case time of this algorithm is $\Theta(n)$, but the worst case is $\Theta(n^2)$.

SELECT(A, n, k)
$p \leftarrow 1$
$r \leftarrow n$
while $p < r$
    do $q = \text{PARTITION}(A, p, r)$
        if $k \leq q$
            then $r \leftarrow q$
            else $p \leftarrow q + 1$
return $A[k]$

You may find a more sophisticated algorithm in Section 10.3 of the textbook; the worst-case time of the textbook algorithm is $\Theta(n)$.

Problem 2

Consider a computer environment where the control flow of a program can split three ways after a single comparison $a_i : a_j$, according to whether $a_i < a_j$, $a_i = a_j$, or $a_i > a_j$. Argue that the number of these three-way comparisons required to sort an n-element array is $\Omega(n \lg n)$.

Suppose that all numbers in the array are *distinct*. Then, a comparison sort never encounters the "$a_i = a_j$" situation, and we may represent its control flow by a binary decision tree. The height of this tree is $\Omega(n \lg n)$; hence, the worst-case complexity of sorting an array of distinct numbers is $\Omega(n \lg n)$. Therefore, the complexity of sorting an arbitrary array is also $\Omega(n \lg n)$.

Problem 3

Suppose that $A[1..n]$ is an array of integer numbers, and some value k occurs at least $\lfloor n/2 \rfloor + 1$ times in this array. Write an efficient algorithm for finding this value.

The "frequent" element is the median of the array, that is, it is the $\lfloor n/2 \rfloor$-th smallest element. We can find it using the SELECT algorithm from Problem 1, with $k = \lfloor n/2 \rfloor$.

Problem 4

We consider an array $A[1..n]$ and define a segment sum from p to r as follows:

$$sum(p, r) = \textstyle\sum_{p \le i \le r} A[i].$$

Write a *linear-time* algorithm that determines the maximum over all segment sums.

MAX-SEGMENT(A, n)
Local-Max $\leftarrow 0$
Global-Max $\leftarrow 0$
for $i \leftarrow 1$ **to** n
  do *Local-Max* $\leftarrow \max(A[i],$ *Local-Max*$+A[i])$
    $\triangleright$ *Local-Max* is the maximum over the segments whose last element is $A[i]$.
  Global-Max $\leftarrow \max($*Local-Max, Global-Max*$)$
    $\triangleright$ *Global-Max* is the maximum over all segments in $A[1..i]$.
return *Global-Max*