ALGORITMIEK: some solutions to exercise class 9

Problem 2.
We use the algorithm from the lecture slides to fill the knapsack table row-by-row. This yields the following table:

	i	capacity j 0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
$w_1 = 3, v_1 = 25$	1	0	0	0	25	25	25	25
$w_2 = 2, v_2 = 20$	2	0	0	20	25	25	45	45
$w_3 = 1, v_3 = 15$	3	0	15	20	35	40	45	60
$w_4 = 4, v_4 = 40$	4	0	15	20	35	40	55	60
$w_5 = 5, v_5 = 50$	5	0	15	20	35	40	55	65

The maximal value of a feasible subset is $F[5][6] = 65$. The optimal subset is $\{\text{item } 3, \text{item } 5\}$.

Problem 3.
a. As said, $P(i, j)$ is the probability of A winning the series if A needs i more games to win the series and B needs j more games to win the series. If team A wins the next game, which happens with probability p, A will need $i - 1$ more wins to win the series while B will still need j wins. If team A looses the game, which happens with probability $q = 1 - p$, A will still need i wins while B will need $j - 1$ wins to win the series. This leads to the recurrence relation:

$$P(i, j) = p \cdot P(i - 1, j) + q \cdot P(i, j - 1) \text{ for } i, j > 0$$

The initial conditions follow immediately from the definition of $P(i, j)$:

$$P(0, j) = 1 \text{ for } j > 0,\ \ P(i, 0) = 0 \text{ for } i > 0$$

b. Here is the dynamic programming table in question, with its entries rounded-off to two decimal places. (It can be filled either row-by-row, or column-by-column, or diagonal-by-diagonal.)

$i \backslash j$	0	1	2	3	4
0		1	1	1	1
1	0	0.40	0.64	0.78	0.87
2	0	0.16	0.35	0.52	0.66
3	0	0.06	0.18	0.32	0.46
4	0	0.03	0.09	0.18	0.29

c.

```
Algorithm WorldSeries (int n, double p)
// Computes the odds of winning a series of n games
// Input: A number of wins n needed to win the series
//        and probability p of one particular team winning a game
// Output: The probability of this team winning the series
{ q = 1-p;
  for (j=1; j<=n; j++)
```

```
    P[0][j] = 1.0;
  for (i=1; i<=n; i++) {
    P[i][0] = 0.0;
    for (j=1; j<=n; j++)
      P[i][j] = p * P[i-1][j] + q * P[i][j-1];
  }

  return P[n][n];
}
```

Problem 6.

a. We use algorithm `bin3` from the lecture slides to fill the table row-by-row. We only fill columns 0–3, because we do not need higher columns to compute $C(6,3)$. This yields the following numbers:

$i \backslash j$	0	1	2	3
0	1			
1	1	1		
2	1	2	1	
3	1	3	3	1
4	1	4	6	4
5	1	5	10	10
6	1	6	15	20

b. Yes, the table can also be filled column-by-column, with each column filled top-to-bottom starting with 1 on the main diagonal of the table. This is achieved with the following code:

```
int bin4 (int n, int k) {
  for (j=0; j<=k; j++)
    for (i=j; i<=n; i++)
      if (j==0 || j==i)
        C[i][j] = 1;
      else
        C[i][j] = C[i-1][j-1] + C[i-1][j];

  return C[n][k];
}
```