Exercise 2.1  Recurrence relations.

Find a closed form for a recurrence relations of the form

\[ T(n) = \begin{cases} aT\left( \frac{n}{b} \right) + cn + d & n > 1 \\ e & n = 1 \end{cases} \]

with \( a, b, c, d, e \in \mathbb{N}, b > 1 \). Prove your answer using mathematical induction. You can assume that \( n \) is a power of \( b \).

Hint: Consider the cases i) \( a \neq b, a \neq 1 \) ii) \( a \neq b, a = 1 \) and iii) \( a = b \) in your proof.

Exercise 2.2  Estimating asymptotic running time (part of an exam in January 2012).

Give the asymptotic running time of the following code fragments depending on \( n \in \mathbb{N} \) using \( \Theta \)-Notation (you don’t have to prove your answer):

a)  
```cpp
for(int j = 0; j < n; ++j)
    for(int k = 2; k <= n; k *= 2)
        for(int l = 0; l < n; l += 10)
            ;
```

b)  
```cpp
for(int h = 0; h < n; h += 2)
    { 
        for(int j = 1; j <= n*n; j *= 3)
            ;
        for(int k = 2; k*k <= n; ++k)
            ;
    }
```

c)  
```cpp
for(int j = 1; j <= n; j *= 2)
    for(int k = 0; k < n*n*n; k += j)
        ;
```
Exercise 2.3  
Comparison of sorting algorithms.

The following code fragments are naive implementations of the basic algorithms to sort the elements $A[l], A[l+1], \ldots, A[r]$ of an array $A$ (in increasing order): insertionSort, selectionSort, bubbleSort, and quickSort.

```cpp
void insertion(vector<int>& A, int l, int r)
{
    for(int i = l; i <= r; ++i)
        for(int j = i-1; j >= l; --j)
            if(A[j] > A[j+1])
                swap(A[j], A[j+1]);
        else
            break;
}

void selection(vector<int>& A, int l, int r)
{
    for(int i = l; i < r; ++i) {
        int minJ = i;
        for(int j = i+1; j <= r; ++j)
            if(A[j] < A[minJ])
                minJ = j;
        if(minJ != i)
            swap(A[i], A[minJ]);
    }
}

void bubble(vector<int>& A, int l, int r)
{
    for(int i = r; i > l; --i)
        for(int j = l; j < i; ++j)
            if(A[j] > A[j+1])
                swap(A[j], A[j+1]);
}

void quick(vector<int>& A, int l, int r)
{
    if(l >= r) return;
    int i = l + 1, j = r;
    do {
        while(i < j && A[i] <= A[l])
            ++i;
        while(i <= j && A[j] >= A[l])
            --j;
        if(i < j)
            swap(A[i], A[j]);
    } while(i < j);
    swap(A[l], A[j]);
    quick(A, l, j - 1);
    quick(A, j + 1, r);
}
```

In this exercise, we will count the number of permutations and comparisons that these methods require. Any 'swap' in the above implementations will count as a permutation, every comparison of two elements from 'A[]' will count as a comparison. Write in a table the asymptotic number of permutations and comparisons that each of these implementations require at least and at most. Also, describe an input sequence of the numbers 1, 2, …, n for which these cases occur. The sequences should be written such that n can be chosen arbitrarily (for example, the descending sequence can be written as $n, n-1, \ldots, 2, 1$). Write your results in a table of the form

<table>
<thead>
<tr>
<th></th>
<th>insertionSort</th>
<th>selectionSort</th>
<th>bubbleSort</th>
<th>quickSort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparisons</td>
<td>min</td>
<td>max</td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>Input sequence</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Permutations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input sequence</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise 2.4  
Algorithm design: divide-and-conquer.

We will say that an array of $n$ elements $A[1..n]$ has a majority vote if more than half of his entries are equal. Design an algorithm that discovers whether a given array has a majority vote, and also finds the corresponding entry. You cannot assume that the entries come from an ordered universe, so the operator ,<” cannot be used. You can, however, decide in constant time whether two entries are the same or if they are different.

a) Show how this problem can be solved in time $O(n \log n)$ using the divide-and-conquer paradigm.  
   Hint: if the array is split in two equal parts, what can you infer from the partial solutions that helps towards the overall solution?

b) Bonus question: Can the problem in a) be solved more efficiently? In this case, can you provide a faster solution?

Hand-in: until Wednesday 7th March 2012.