Midterm Examination 2
GEM 1501: Problem Solving for Computing

Thursday 27.03.2014, duration half an hour

Matriculation Number: _______________

Rules
This test carries 10 marks and consists of 5 questions. Each questions carries 2 marks; full marks for a correct solution; a partial solution can give a partial credit.

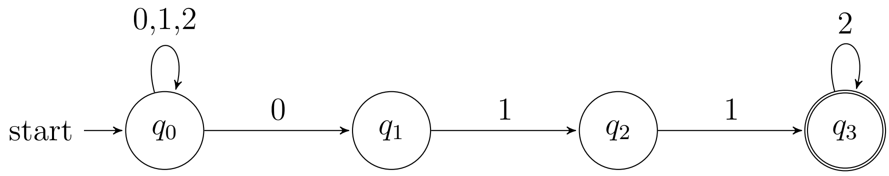
Question 1 [2 marks].
Explain how bubble sort works and write down its worst case time complexity.

Question 2 [2 marks].
Determine the worst case runtime complexity of the following program using the parameter n being the number of array elements of a:

```
function findthree(a)
  { var n = a.length; var m = false;
    var i; var j; var k;
    for (i=0;i<n;i++)
      { for (j=i+1;j<n;j++)
          { for (k=j+1;k<n;k++)
              { if ((a[i] != a[j]) && (a[j] != a[k])
                    && (a[i] != a[k])) { m = true; } } } }
    return(m); }
```

Write down the worst case time complexity in Θ-notation and indicate whether a better run time is possible with another algorithm. If so, give the program; if not, say why it cannot be done.

Question 3 [2 marks].
Consider a non-deterministic finite automaton with the following state table:



Make an equivalent deterministic finite automaton.

Question 4 [2 marks].
Some function f satisfies $f(n) > 0$ for all n and $f(n+m) = f(n) \cdot f(m)$. Furthermore, let

$$g(n) = f(0) + f(1) + \ldots + f(n)$$

for all n. The following algorithm uses a program for f as a subroutine and computes g using a divide-and-conquer algorithm.

```
function f(n) { ... }

function g(n)
  { if (n < 1) { return(f(0)); }
    if (n < 2) { return(f(0)+f(1)); }
    if (n < 3) { return(f(0)+f(1)+f(2)); }
    var m = Math.floor(n/2); var k = f(m+1);
    return(g(m)+k*g(n-m-1)); }
```

When it computes $g(n)$, this program uses $\Theta(n)$ many calls of f. Use dynamic programming or a similar method, make a new program which computes $g(n)$ using only $\Theta(\log(n))$ many calls of f.

Question 5 [2 marks].
Let a graph be given by an array `edge` such that each entry in `edge[k]` is an array `[v,w]` being equal to `[edge[k][0],edge[k][1]]` representing an edge going from vertex `v` to vertex `w`. Make a function which has as input an array `edge` and a starting vertex `v` and as outputs a vertex `w` such that `w` can be reached from `v` and either `w` can also be reached from itself by a loop (perhaps going through other vertices) or `w` is a sink, that is, a vertex without outgoing edge.

For example, consider the case where `edge` contains the three array elements `[0,1]`, `[0,2]` and `[1,2]`. If the algorithm is run with inputs `edge` and `0` then it has to return `2` as `2` is the sink.

```
function search(edge,v)
  { var w;








    return(w); }
```

END OF EXAMINATION.