Russ Johnson
Data Structures and Algorithms
Assignment 5
February 6, 2013

A.
2.31 The binary search will still work. The difference is that it will include the middle element in the next search, which we already know is not the element we are looking for. It has the same big-Oh notation and will not run noticeably slower.

C
4.8
prefix: $- * * ab + cde$
infix: $((a * b) * (c + d)) - e$
postfix: $ab * cd + * e-$

Maze 1

```
###########
#.........#
#.##### #.#
..  #   #.#
### # ###.#
#     #...#
# # ###.###
# #   #....
# ### # # #
#     #   #
###########
```

Maze 2

```
#####################
#     #     #   #   #
# # # # ### # # # # #
# #   # #   #   # # #
# ##### # ####### # #
#     # #       #   #
##### # ####### ### #
#   #   #     #   # #
# # ##### ### # # # #
# #       #   # #   #
# ##### ### ### ### #
#     #   # #       #
### # ### # # # #####
..  # #   # # #     #
#.### # ### # ### # #
#...  #     #   # # #
###.######### # # # #
#...#...#...# #   # #
#.###.#.#.#.# ##### #
#.....#...#..........
#####################
```

Maze 3

```
#########################################
#     #       #       #         #   #   #
# # # # # ### # ### # # ####### # # # # #
#   # # # #   #     # #       #     # # #
# ### # # # # # ##### ##### # ####### # #
# #   # #   # # #   #     #   #       #..
# # # # ##### # # # # ### ##### # #####.#
#   # #       # # #   #   #   # # #.....#
# ### ### ##### # ##### ### # # # #.# ###
# #       #   # # #     #   # # # #.#   #
# # # ##### # # # # ##### ### # # #.### #
# # #     #     # # #       #   # #.    #
# # ##### ####### # # ### # ##### #.#####
# #   #       #   # #     #       #.#...#
# ### # ##### # ### ### ##### #####.#.#.#
#   # # #     #     #   #   #     #...#.#
# # # # # ### ####### ### # ##### #####.#
# # #   #     #         #   #...........#
# # ########### # ##### #####.### ##### #
# #       #   #   #   #.......  #     # #
# ####### # # ##### # #.####### ##### # #
#         #   #...#   #...#         #   #
##### ### #####.#.#######.# ####### #####
#   #   # #.....#...#.....  #     # #   #
# # ### # #.### # #.#.####### # # # # # #
#   #   # #.#   # #...    #   #   # #   #
##### # # #.# ### ### ### # ####### #####
#   #   # #.#   #   # #   #   #   # #   #
# # ##### #.### # # # # # ### # # # # # #
# # #     #.#   # # #   # #   # #   # # #
# # # ### #.# # # # ##### # ### ##### # #
# # # #   #.# #     #   # #   # #     # #
# # # # ###.# ####### # # ### # # ##### #
# #   # #...#     #   # #   # # #     # #
# ##### #.####### # # # ### # # ##### # #
#   #...#.  #   # # # #   #   #     #   #
### #.#.#.# # # # # # ### ####### # #####
#   #.#...#     # # #   # #     # #     #
# ###.########### # ### # # # # # # ### #
......            #       #   #   #     #
#########################################
```

Maze 4

```
###########
#         #
#         #
..        #
#.        #
#.        #
#.        #
#..........
#         #
#         #
###########
```

```
#include <iostream>
#include <fstream>
#include <string>
#include <queue>
using namespace std;

class MazeSolver {

private:

    char maze[100][100];
    //number of rows and columns in maze
    int rows, cols;
    //entrance row and column
    int entrance_r, entrance_c;
    //exit row and column
    int exit_r, exit_c;

    bool solved;

    struct square {
        int r;
        int c;
        square * parent;
    };

    queue<square *> q;

public:

MazeSolver(char * const file) {

    ifstream inFile;
    inFile.open(file);

    int row = 0;
    string line;

    while(!inFile.eof()) {

        getline(inFile,line);

        if (line.size() > 0)
            cols = line.size();

        for(int x=0; x<line.size(); x++) {
            maze[row][x] = line[x];
        }

        row++;

    }

    rows = row-1;
    findEntryAndExit();

    solved = false;
}

void print() {
    for(int x=0; x<rows; x++) {
        for(int y=0; y<cols; y++) {
            cout << maze[x][y];
        }
        cout << '\n';
    }
}

void solveMaze() {

    square * s = new square;
    s->r = entrance_r;
    s->c = entrance_c;
    s->parent = NULL;
    q.push(s);

    int r,c;

    while(!q.empty() && !solved) {

        s = q.front();
        q.pop();

        r = s->r;
        c = s->c;

        if(r==exit_r && c==exit_c) {
            break;
        }

        else {
            enque(r-1,c,s);
            enque(r+1,c,s);
            enque(r,c-1,s);
            enque(r,c+1,s);
        }
    }

    clear();

    while(s != NULL) {
        maze[s->r][s->c] = '.';
        s = s->parent;
    }
}

private:

//returns true when end is found
```

```
void enque(int r, int c, square * parent) {
    if(maze[r][c] == ' ') {
        maze[r][c] ='.';

        square * s = new square;
        s->r = r;
        s->c = c;
        s->parent = parent;

        q.push(s);
    }
}

void findEntryAndExit() {
    for(int x=0; x<rows; x++) {

        if( maze[x][0] == ' ') {
            entrance_r = x;
            entrance_c = 0;
        }

        if( maze[x][cols-1] == ' ') {
            exit_r = x;
            exit_c = cols-1;
        }
    }
}

void clear() {
    for (int r = 0; r < cols; ++r)
        for (int c = 0; c < rows; ++c)
            if (maze[r][c] != '#')
                maze[r][c] = ' ';
}

};

int main(int argc, char * const argv[]) {

    MazeSolver maze(argv[1]);

    maze.solveMaze();

    maze.print();

    return 0;
}
```