AutoRob

Introduction to Autonomous Robotics

Michigan EECS 367

Robot Kinematics and Dynamics

Michigan ME 567 EECS 567 ROB 510

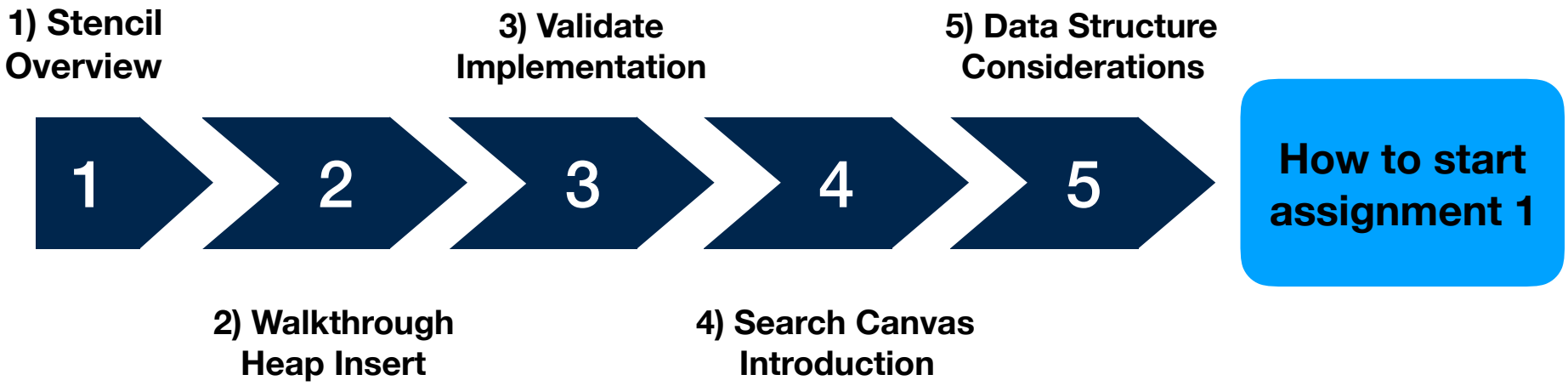
Fall 2019

EECS 367 Lab:
Assignment 1 Code Overview

Administrative

- Next **Wednesday, September 18**
  - Quiz 1 during regular lecture time and location
  - Assignment #1: Path Planning
    - Due 11:59pm, Wednesday, September 18
- Send Professor Jenkins your Git account/repo (through slack)

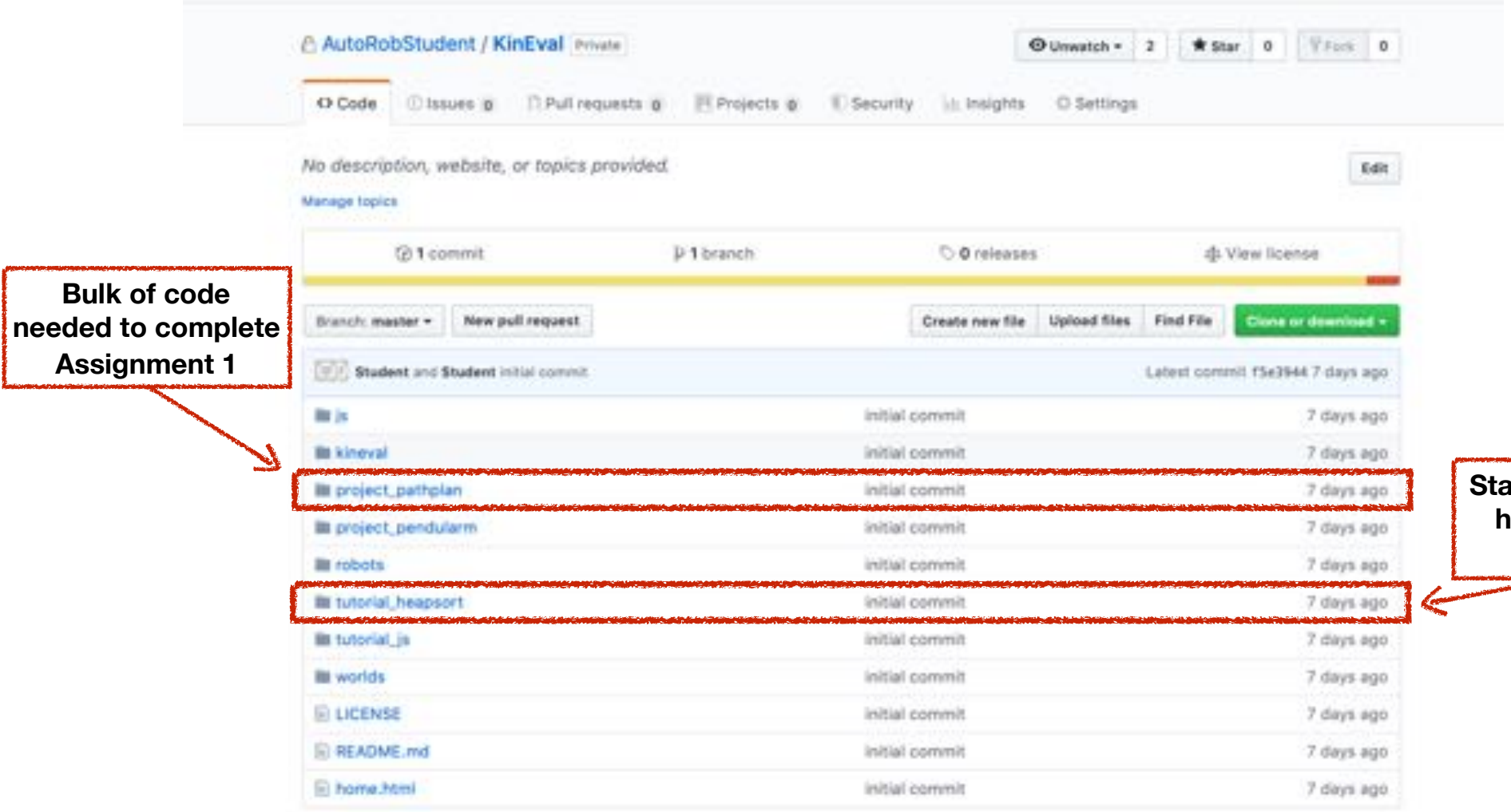
Lab Takeaways



Assignment 1

Points	Sections	Feature
		Assignment 1: 2D Path Planning
4	All	Heap implementation
8	All	A-star search
2	Grad	BFS
2	Grad	DFS
2	Grad	Greedy best-first

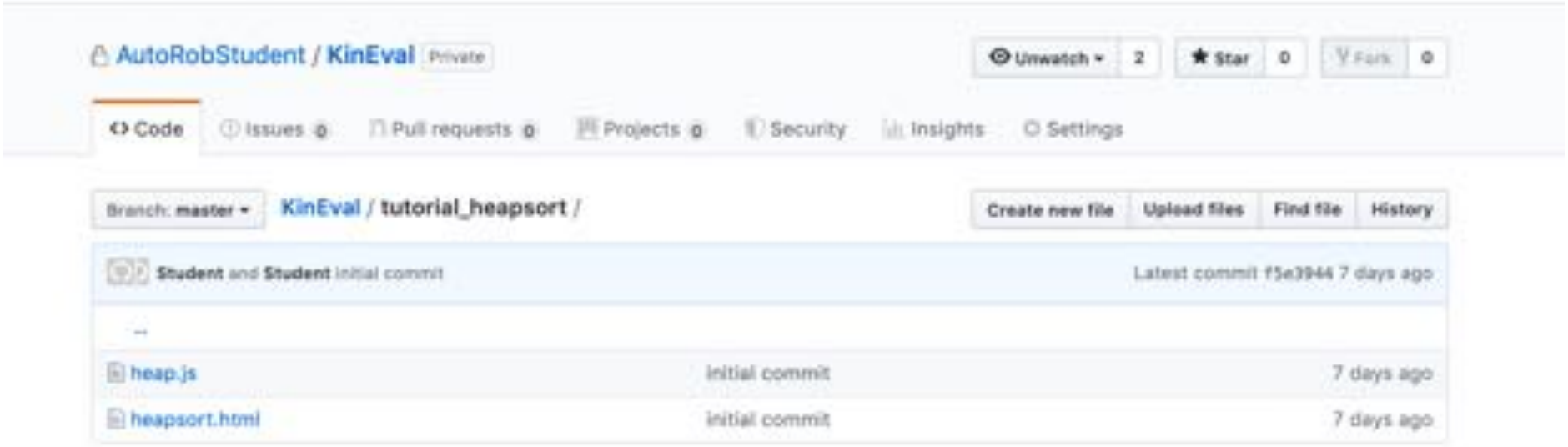
KinEval Stencil



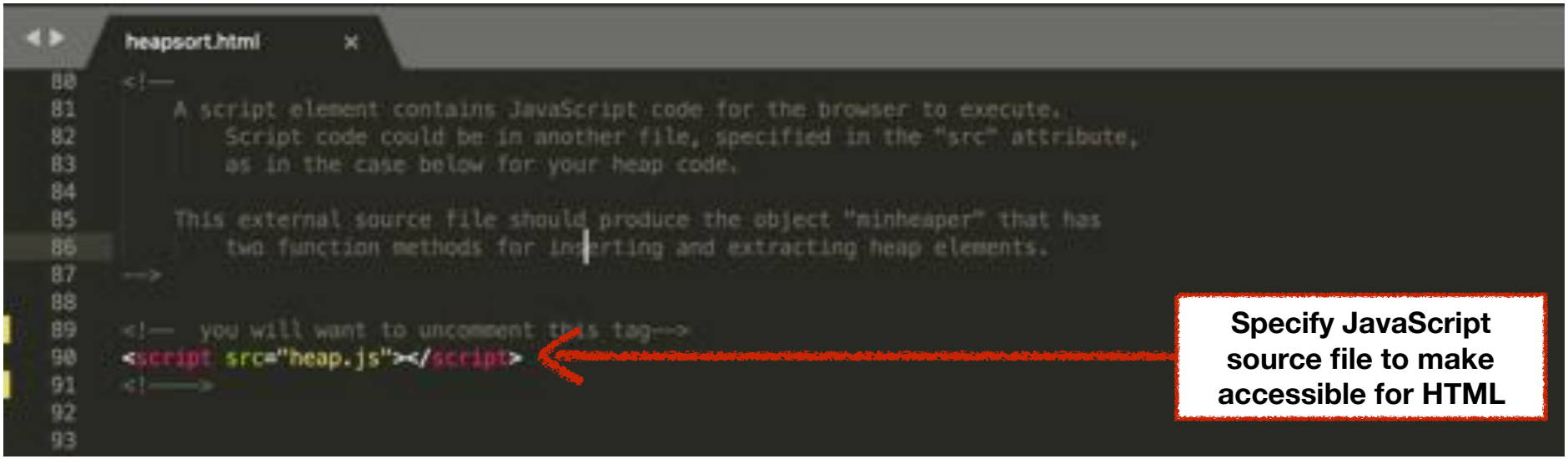
Bulk of code needed to complete Assignment 1

Starter code to help debug heap

KinEval Stencil



heapsort.html



heapsort.html

heapsort.html

```
167     console.log("building min binary heap from number array");
168     numbers_heap = []; // create array for heap
169     for (i=0;i<numbers.length;i++) {
170
171         console.log("inserting number "+numbers[i]+" into the heap");
172         minheaper.insert(numbers_heap,numbers[i]);
173
174         console.log("appending current heap state to output object");
175         output_string = "heap (insert " + numbers[i] + "): "; //
176         for (j=0;j<numbers_heap.length;j++) {
177             output_string += numbers_heap[j] + " "; //
178         }
179         addHTMLLine("output",output_string);
180     }
181
182     console.log("before sorting, crudely draw current heap state to canvas");
183     // grab canvas element and its drawing context
184     console.log("obtaining reference to canvas object in the document");
185     c = document.getElementById("myCanvas");
186     console.log("obtaining the drawing context of the canvas object");
187     ctx = c.getContext("2d");
188
```

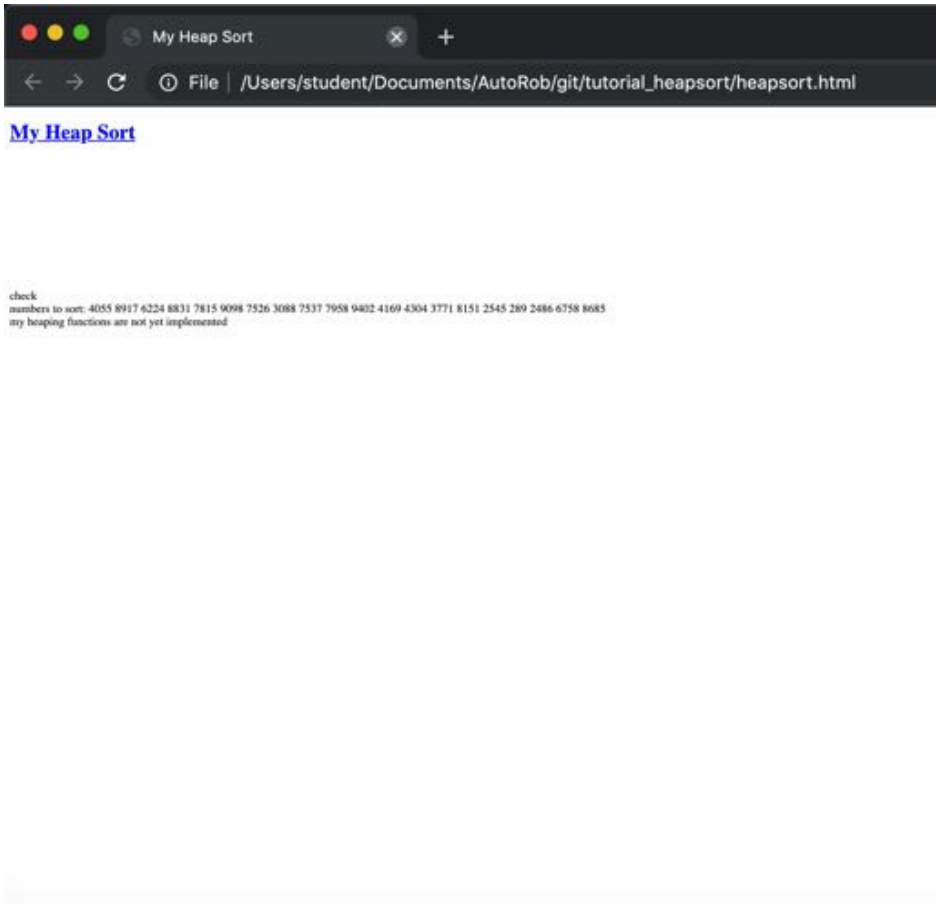
Represent heap as JS array

Repeatedly call heap insert method

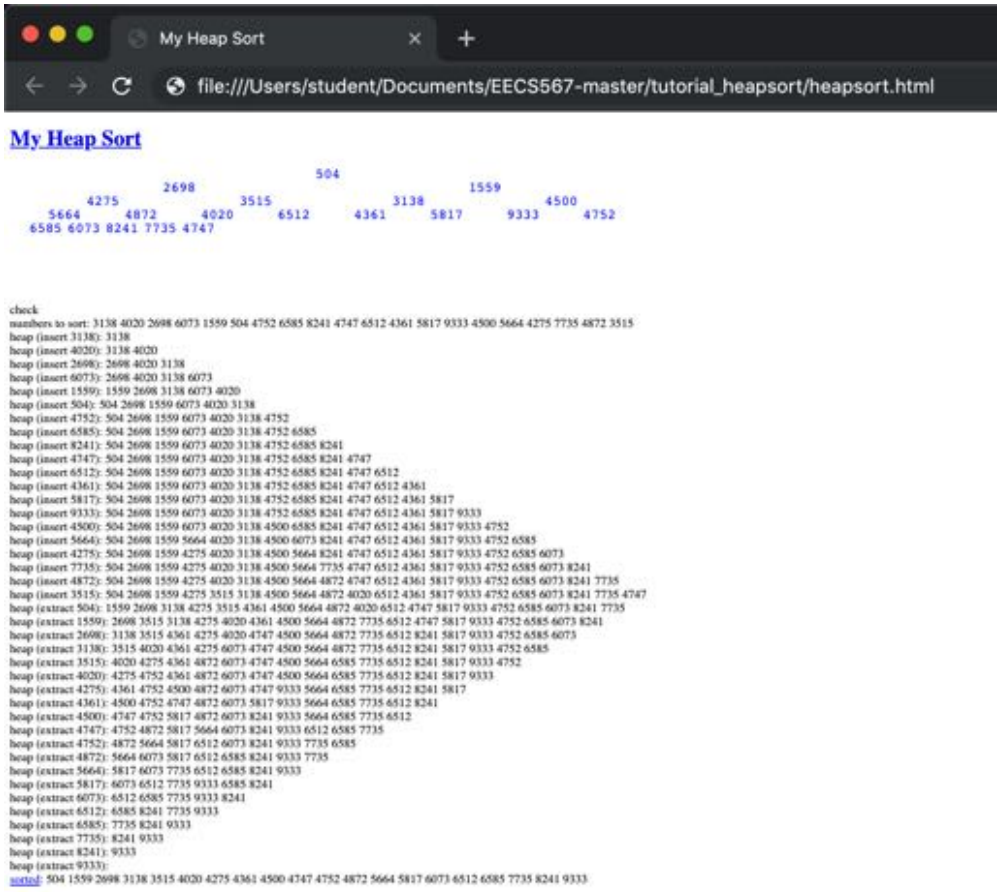
Print state of heap to screen

heapsort.html

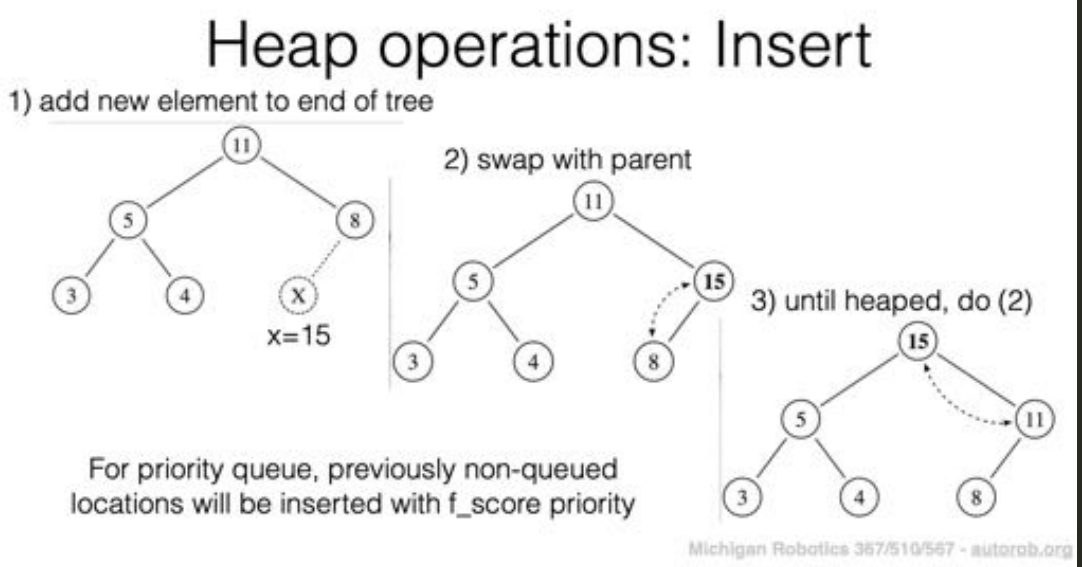
Without heap.js implementation



With heap.js implementation

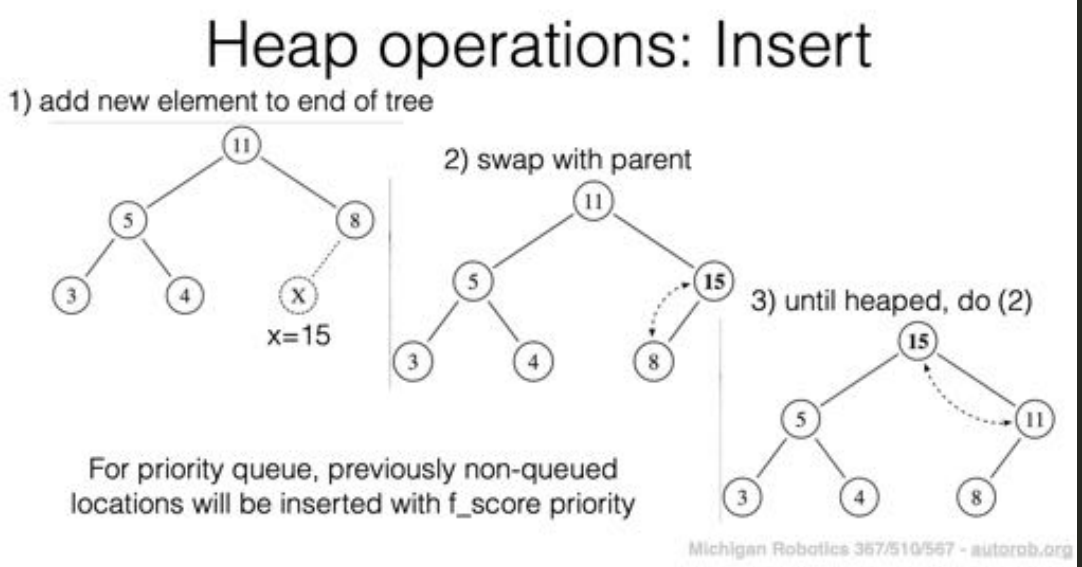


Heap Insert



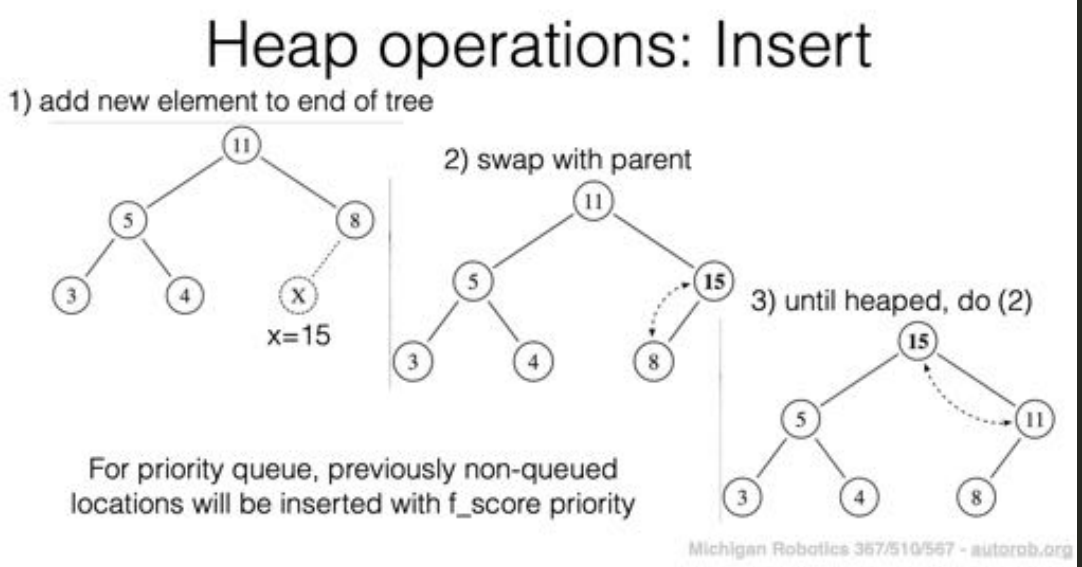

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    // TODO: Find index for new_element
    // TODO: Find index of new_element's parent

    // TODO: Add new_element to the heap array

    // TODO: Initialize heap condition check

    // TODO: As long as heap condition not satisfied
    //       TODO: Swap new_element with parent
    //
    //       TODO: Update index for new_element
    //       TODO: Update index for new_element's parent
    //
    //       TODO: Update heap condition check
}
```

Heap Insert



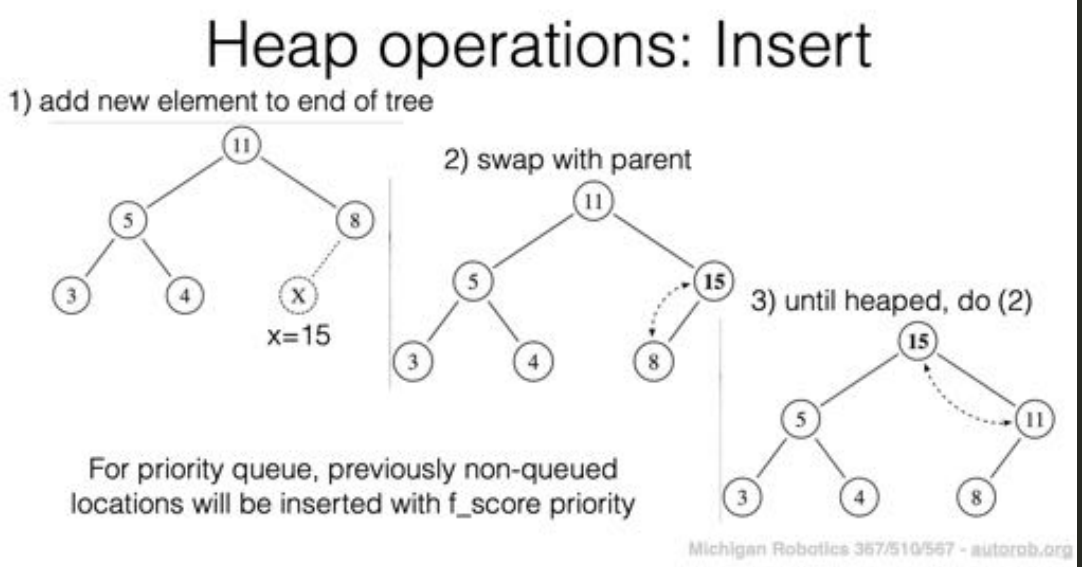

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    // TODO: Find index of new_element's parent

    // TODO: Add new_element to the heap array

    // TODO: Initialize heap condition check

    // TODO: As long as heap condition not satisfied
    //       TODO: Swap new_element with parent
    //
    //       TODO: Update index for new_element
    //       TODO: Update index for new_element's parent
    //
    //       TODO: Update heap condition check
}
```

Heap Insert



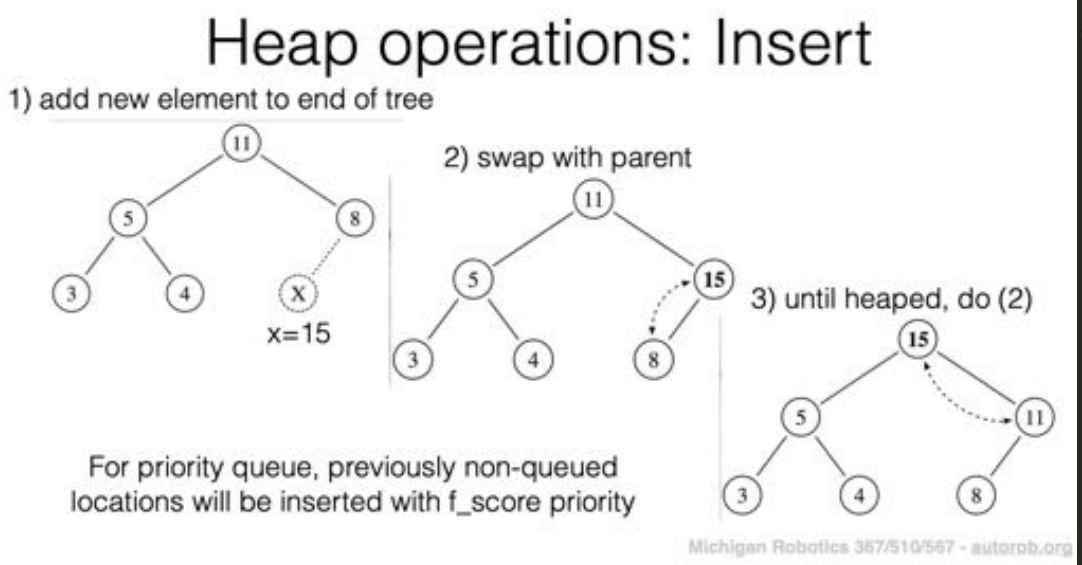

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    // TODO: Add new_element to the heap array

    // TODO: Initialize heap condition check

    // TODO: As long as heap condition not satisfied
    //       TODO: Swap new_element with parent
    //
    //       TODO: Update index for new_element
    //       TODO: Update index for new_element's parent
    //
    //       TODO: Update heap condition check
}
```

Heap Insert



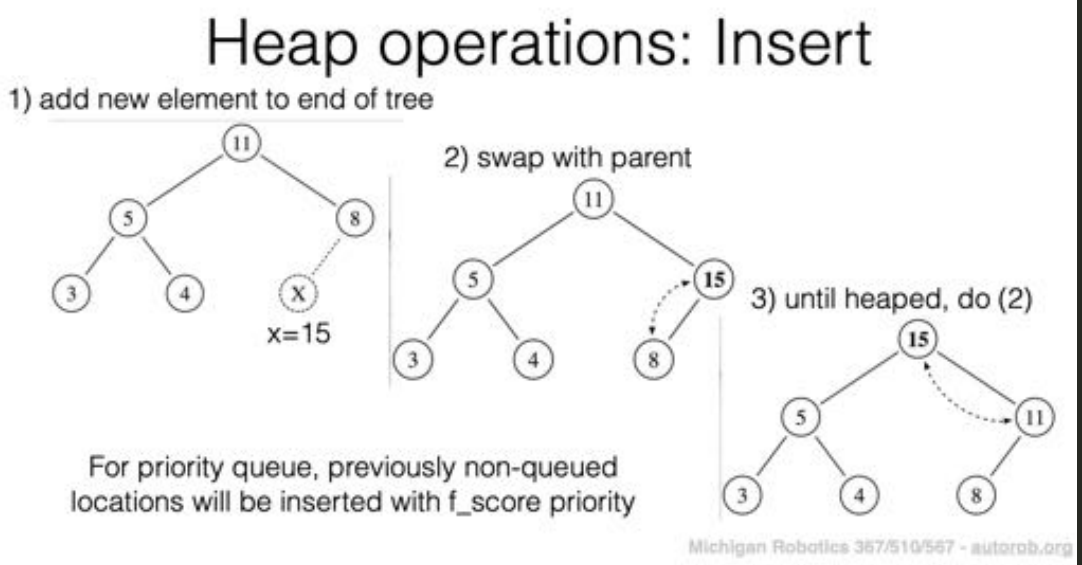

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    heap.push(new_element);

    // TODO: Initialize heap condition check

    // TODO: As long as heap condition not satisfied
    //       TODO: Swap new_element with parent
    //
    //       TODO: Update index for new_element
    //       TODO: Update index for new_element's parent
    //
    //       TODO: Update heap condition check
}
```

Heap Insert



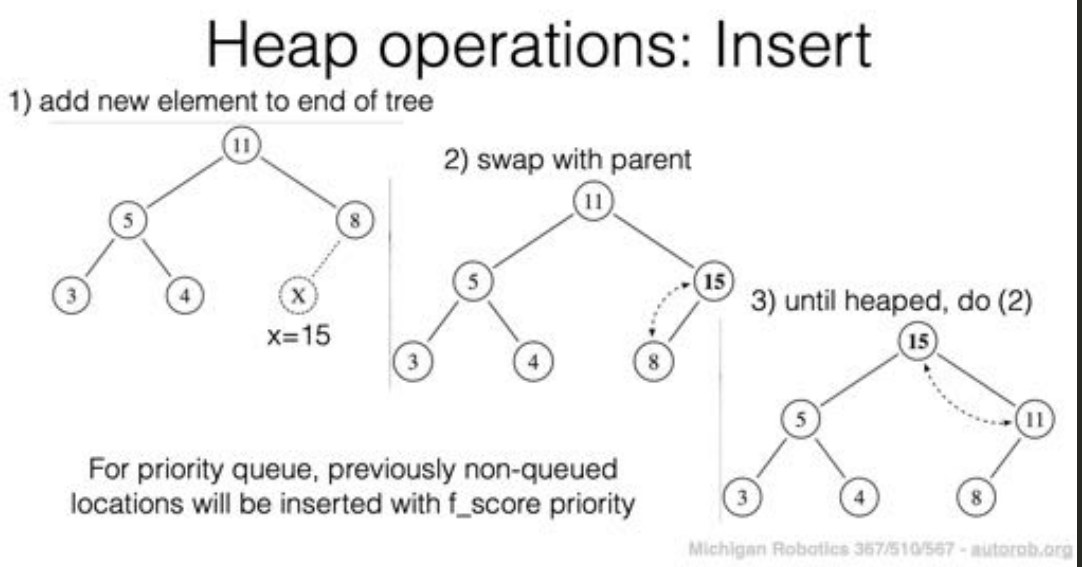

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    heap.push(new_element);

    // Heap condition is true if new element added as root, or if
    //  new element is less than or equal to its parent element
    var heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);

    // TODO: As long as heap condition not satisfied
    //       TODO: Swap new_element with parent
    //
    //       TODO: Update index for new_element
    //       TODO: Update index for new_element's parent
    //
    //       TODO: Update heap condition check
}
```

Heap Insert



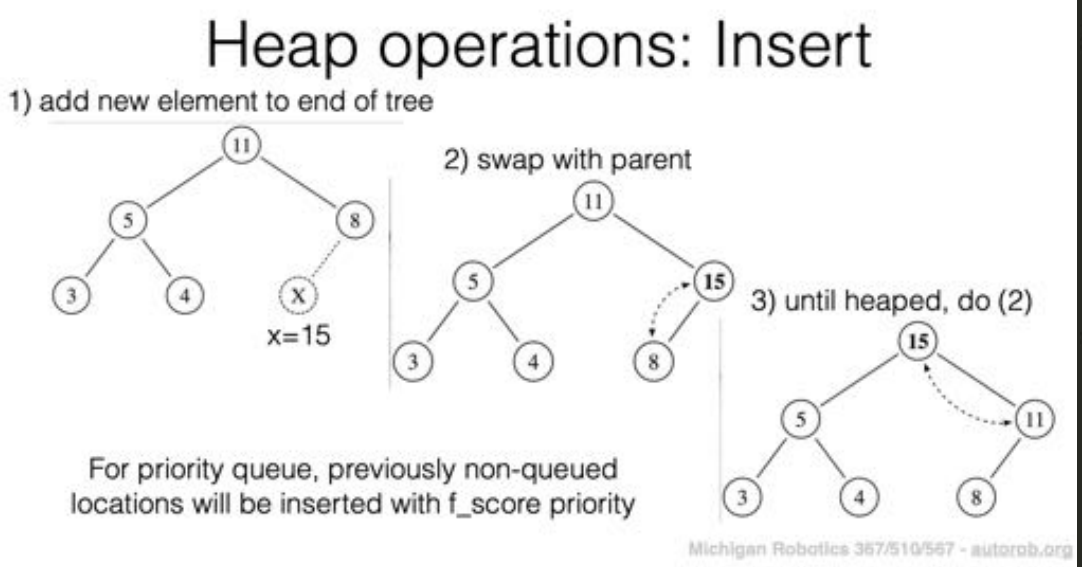

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    heap.push(new_element);

    // Heap condition is true if new element added as root, or if
    //  new element is less than or equal to its parent element
    var heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);

    while (!heaped) {
    //      TODO: Swap new_element with parent
    //
    //      TODO: Update index for new_element
    //      TODO: Update index for new_element's parent
    //
    //      TODO: Update heap condition check
    }
}
```

Heap Insert



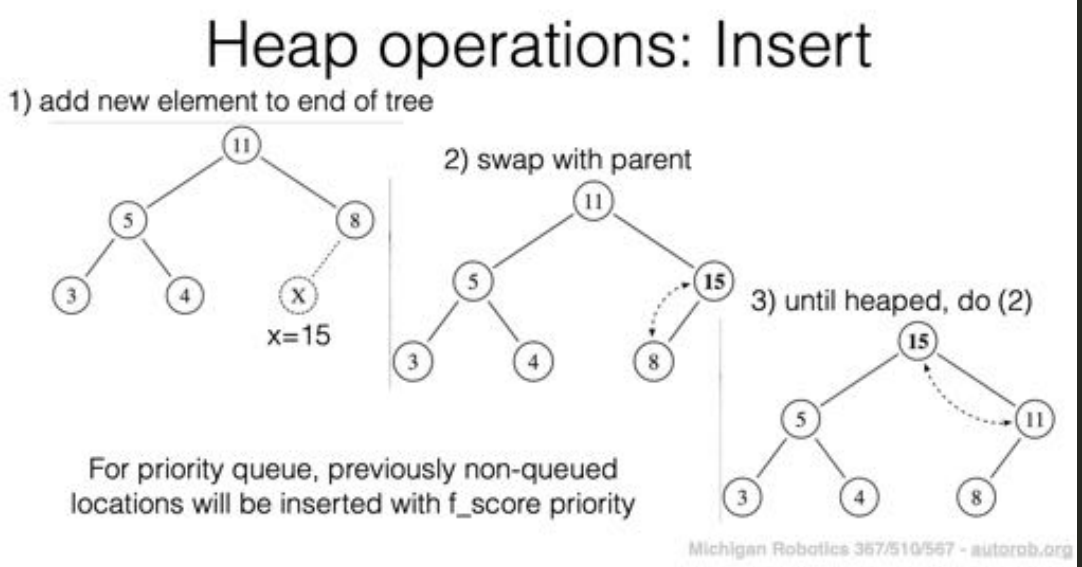

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    heap.push(new_element);

    // Heap condition is true if new element added as root, or if
    //  new element is less than or equal to its parent element
    var heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);

    while (!heaped) {
        // Swap element and parent
        var tmp = heap[prntIdx];
        heap[prntIdx] = heap[elntIdx];
        heap[elntIdx] = tmp;

    //      TODO: Update index for new_element
    //      TODO: Update index for new_element's parent
    //
    //      TODO: Update heap condition check
    }
}
```

Heap Insert



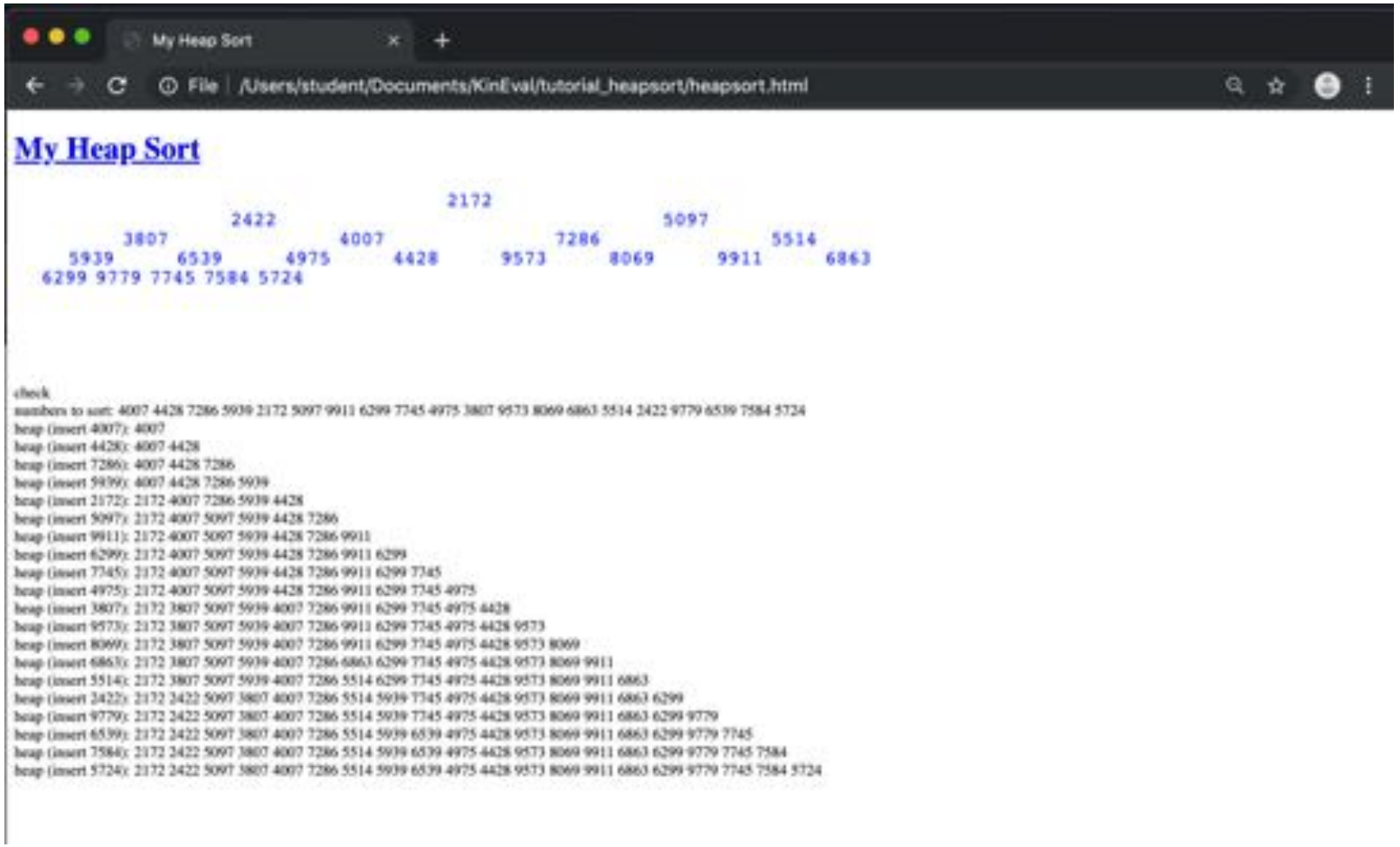

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    heap.push(new_element);

    // Heap condition is true if new element added as root, or if
    //  new element is less than or equal to its parent element
    var heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);

    while (!heaped) {
        // Swap element and parent
        var tmp = heap[prntIdx];
        heap[prntIdx] = heap[elntIdx];
        heap[elntIdx] = tmp;

        // Update element and parent index
        elntIdx = prntIdx;
        prntIdx = Math.floor( (elntIdx -1 ) / 2 );

    //
    //      TODO: Update heap condition check
    }
}
```

Heap Insert

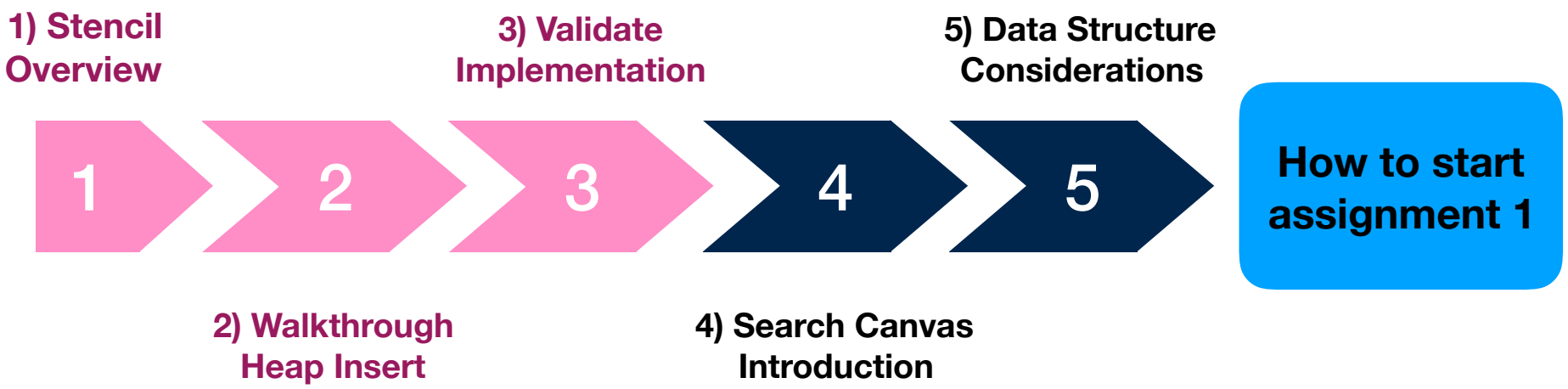



```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    heap.push(new_element);

    // Heap condition is true if new element added as root, or if
    //  new element is less than or equal to its parent element
    var heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);

    while (!heaped) {
        // Swap element and parent
        var tmp = heap[prntIdx];
        heap[prntIdx] = heap[elntIdx];
        heap[elntIdx] = tmp;

        // Update element and parent index
        elntIdx = prntIdx;
        prntIdx = Math.floor( (elntIdx -1 ) / 2 );

        // Re-evaluate heap condition
        heaped = (elntIdx <= 0) || (heap[prntIdx] <= heap[elntIdx]);
    }
}
```

heapsort.html



Lab Takeaways



search_canvas.html

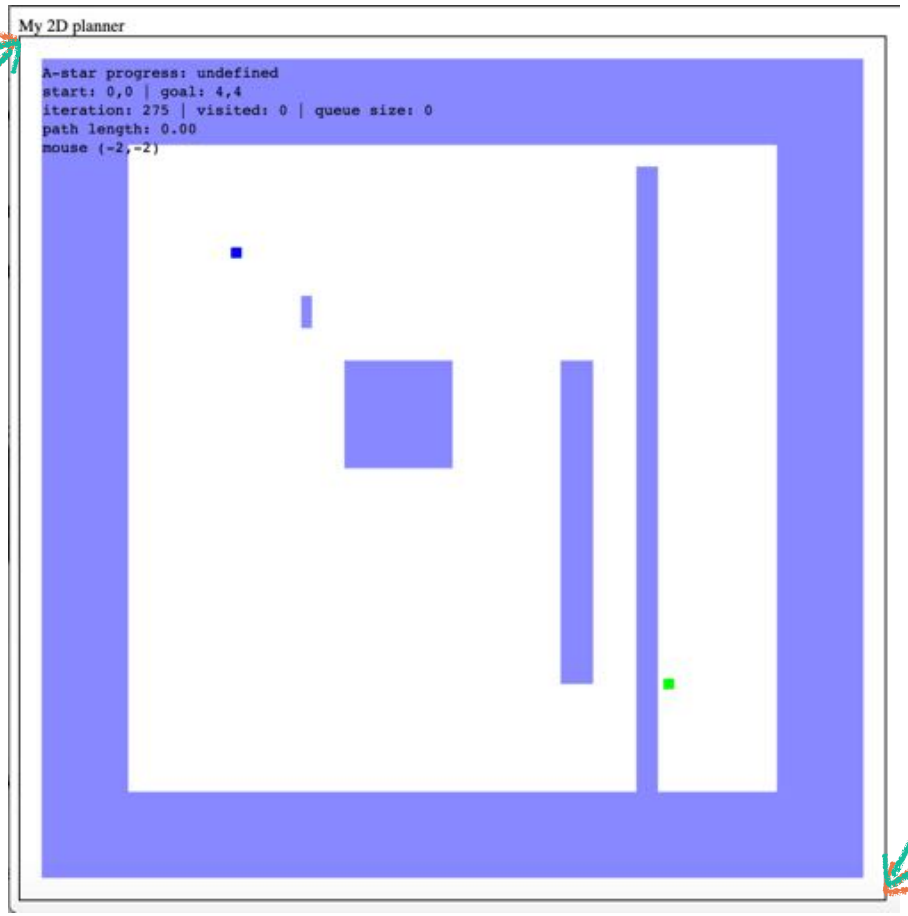
initSearch() function Instantiates global variables, commence algorithms

search_canvas.html

```
function initSearch() {

    // specify default search algorithm to use for planning
    //search_alg = "depth-first";
    //search_alg = "breadth-first";
    //search_alg = "greedy-best-first";
    search_alg = "A-star";
    //search_alg = "RRT";
    //search_alg = "RRT-connect";
    //search_alg = "RRT-star";

    // specify default the world for the planner
    //  (stored as "range" global variable with name "planning_scene")
    //planning_scene = "empty";
    planning_scene = "misc";
    //planning_scene = "narrow1";
    //planning_scene = "narrow2";
    //planning_scene = "three_sections";

    // specify default eps (epsilon) spatial resolution variable
    //   for RRT, specifies threshold radius for step size and reaching goal
    eps = 0.1;

    // create event handlers for the mouse
    canvas = document.getElementById("myCanvas");
    mouse_x = 0;
    mouse_y = 0;

    // when the mouse moves, update the mouse's location
    canvas.onmousemove = function handleMouseMove(event) {
        mouse_x = event.clientX;
        mouse_y = event.clientY;
    };

    // when the mouse button is pressed, update mouseDown
    canvas.onmousedown = function() {
        mouseDown = 1;
    };

    // when the mouse button is released, update mouseDown
    canvas.onmouseup = function() {
        mouseDown = 0;
        q_goal = [xformViewWorldX(mouse_x),xformViewWorldY(mouse_y)];

        // IMPORTANT: mouse_x and mouse_y are in absolute coordinates with
        //    respect to the browser window, and in units of pixels.  The
        //    xformViewWorldX only considers relative coordinates for the
        //    canvas element.
    };

    // specify start and goal configurations
    q_start_config = [0,0];
    q_goal_config = [4,4];
    q_init = q_start_config;
    q_goal = q_goal_config;

```

q_start_config = start cell in grid
q_goal_config = goal cell in grid
q_init

search_canvas.html



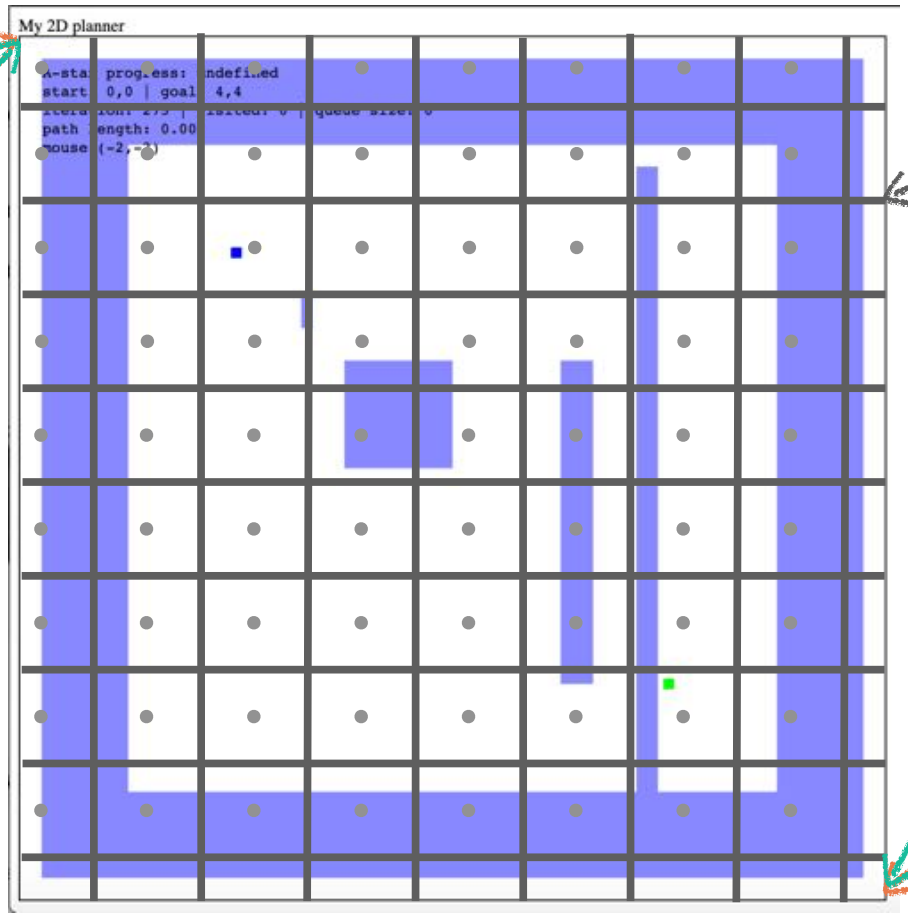
[-2, -2]
(world coordinates)

[0, 0]
(canvas coordinates)

[800, 800]
(canvas coordinates)

[7, 7]
(world coordinates)

search_canvas.html



[-2, -2]
(world coordinates)

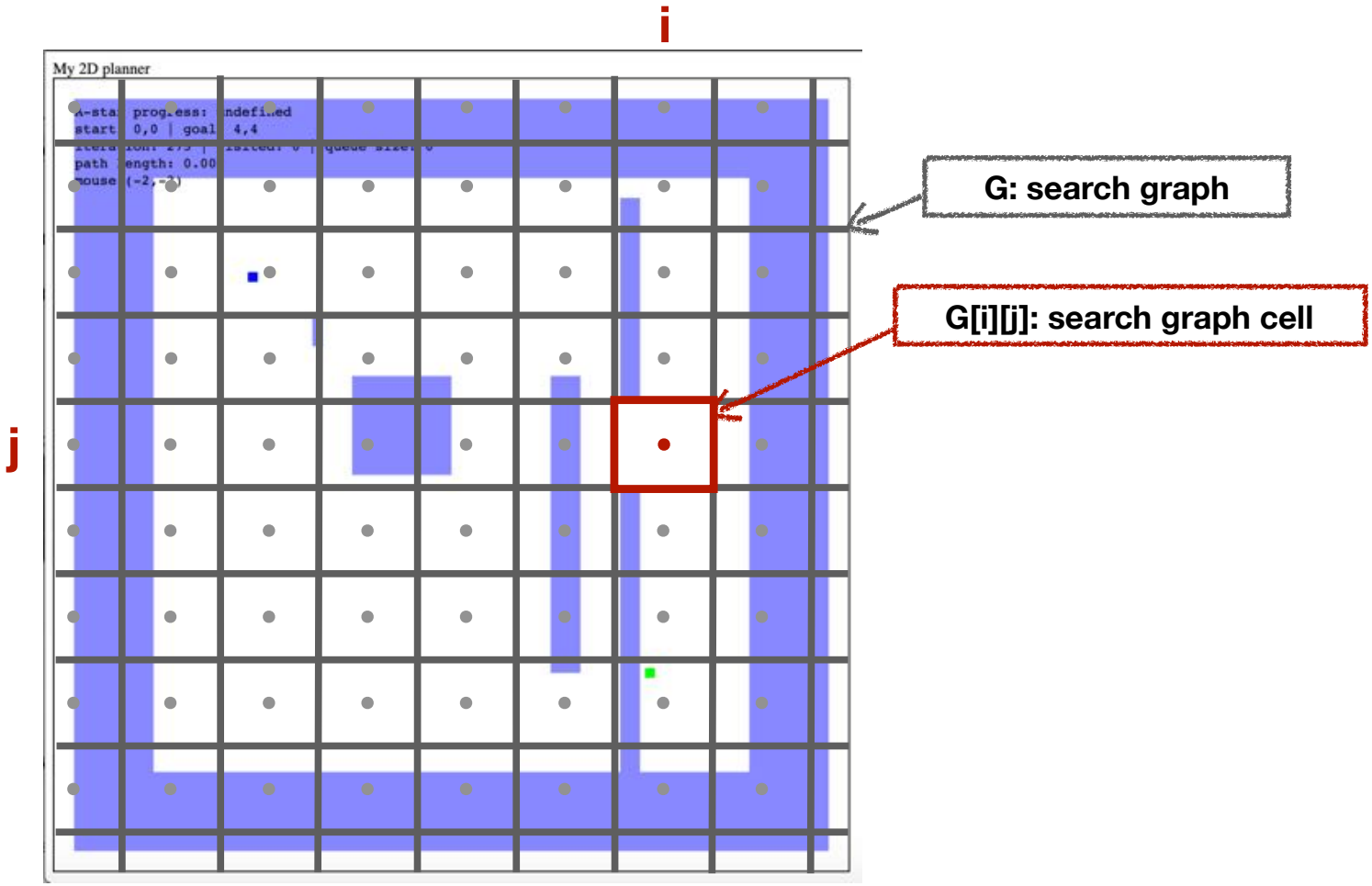
[0, 0]
(canvas coordinates)

G: search graph

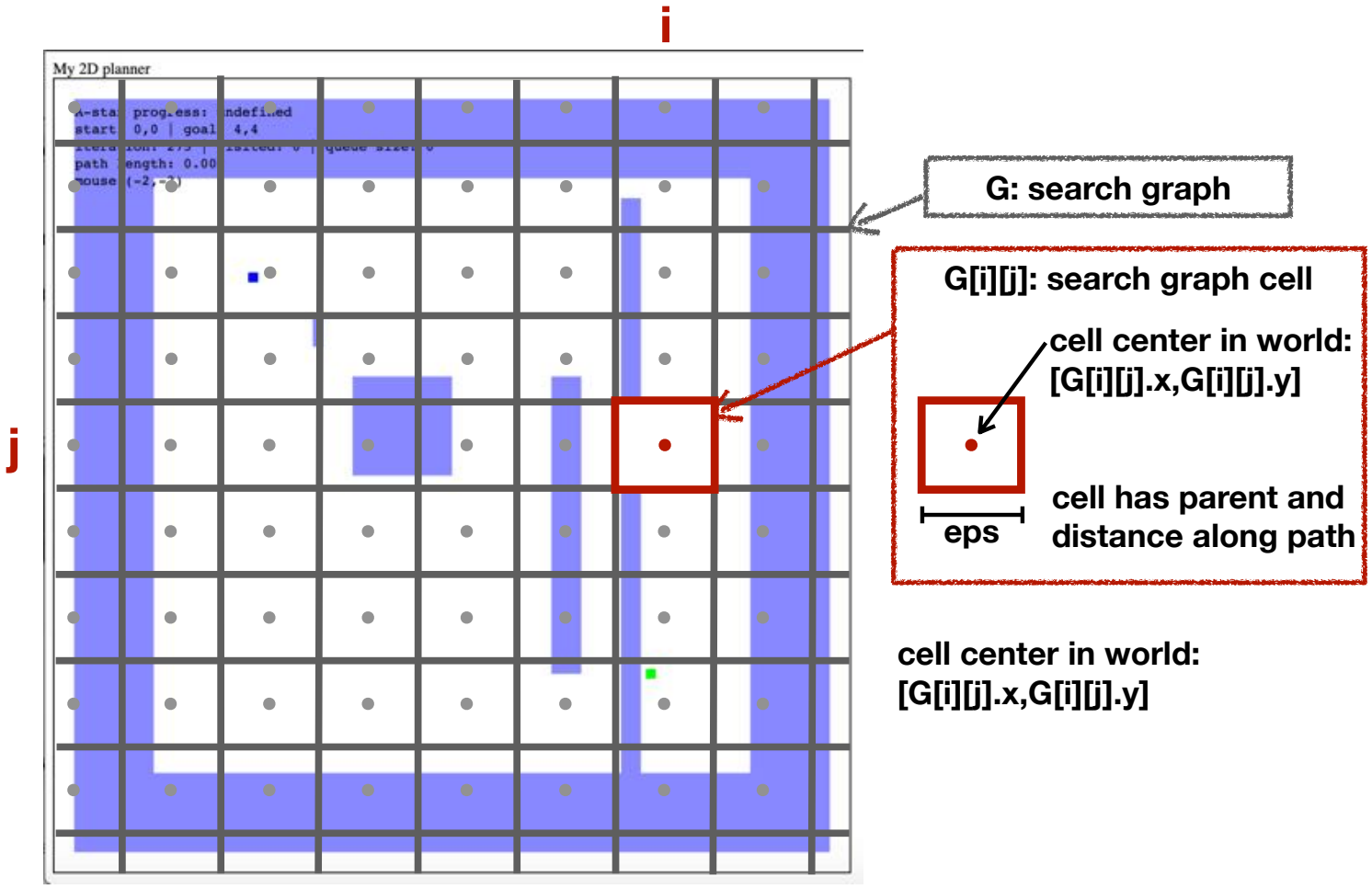
[800, 800]
(canvas coordinates)

[7, 7]
(world coordinates)

search_canvas.html

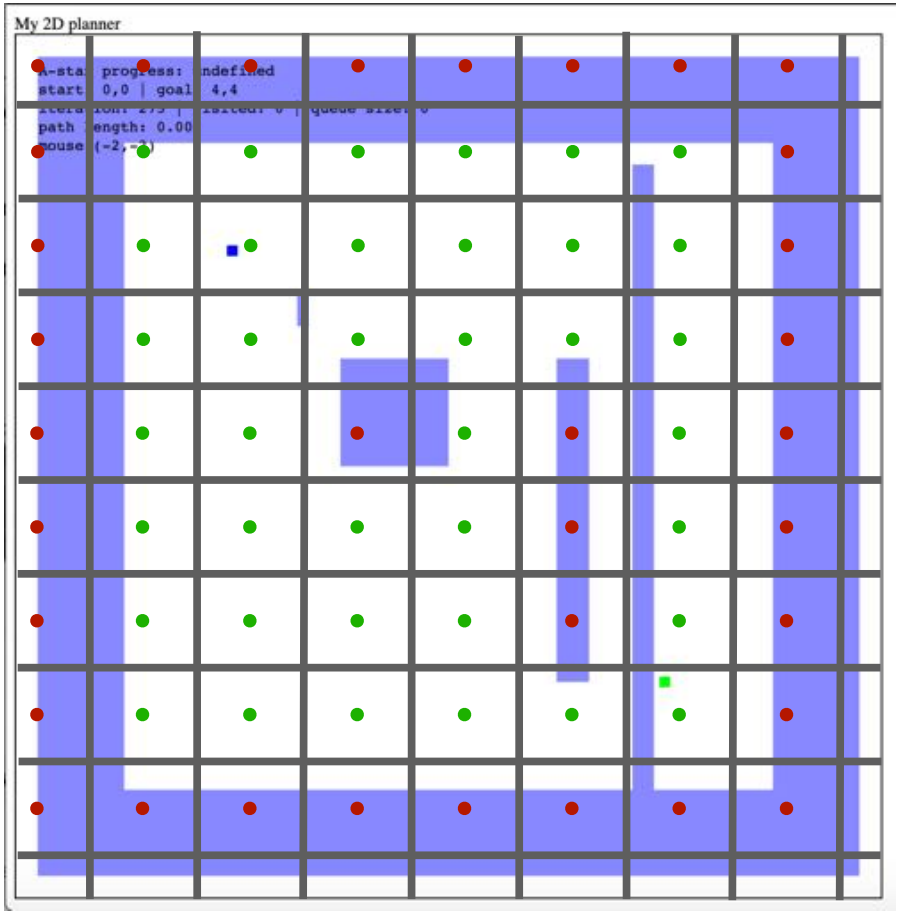


search_canvas.html

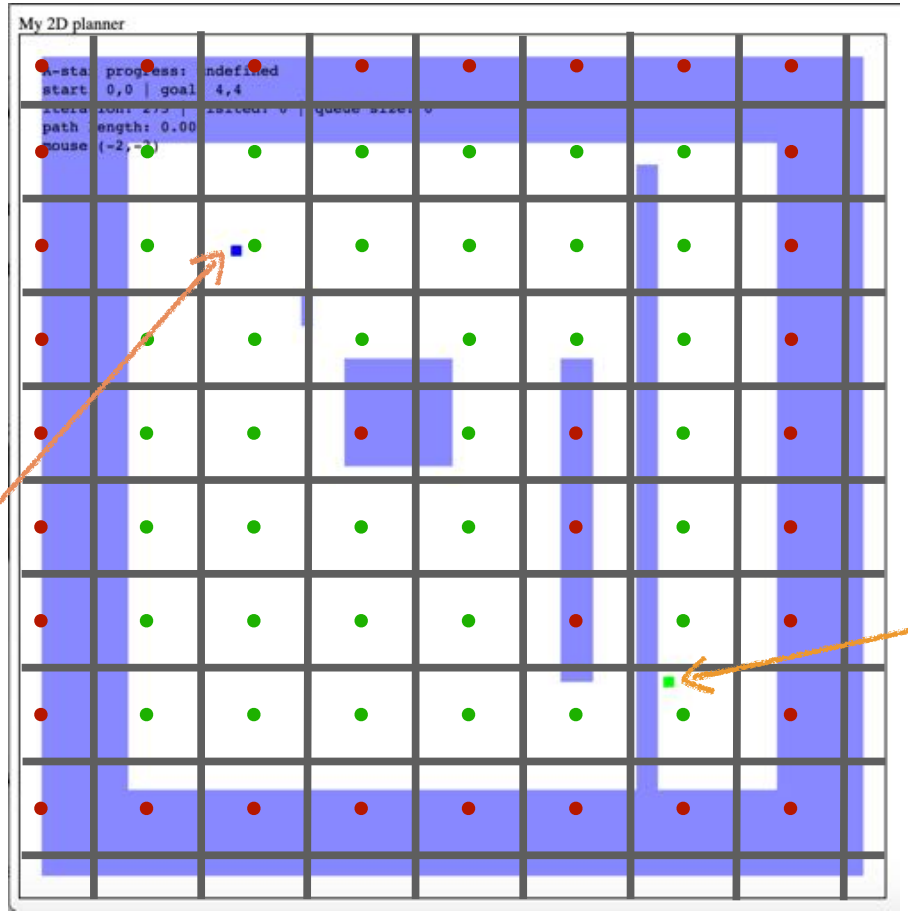


search_canvas.html

test configuration at visited cell center for collision: testCollision([G[i][j].x,G[i][j].y])



search_canvas.html



search starts from cell that contains configuration q_init

q_init=[0, 0]
(world coordinates)

search goal in cell that contains configuration q_goal

q_goal=[4, 4]
(world coordinates)

search_canvas.html

Important to identify discrete start indices within graph from continuous world position

```
function initSearchGraph() {

    // KE: visit queue not created for certain values of eps
    visit_queue = [];

    // initialize search graph as 2D array over configuration space
    //   of 2D locations with specified spatial resolution
    G = [];
    for (iind=0,xpos=-2;xpos<7;iind++,xpos+=eps) {
        G[iind] = [];
        for (jind=0,ypos=-2;ypos<7;jind++,ypos+=eps) {
            G[iind][jind] = {
                i:iind,j:jind, // mapping to graph array
                x:xpos,y:ypos, // mapping to map coordinates
                parent:null, // pointer to parent in graph along motion path
                distance:10000, // distance to start via path through parent
                visited:false, // flag for whether the node has been visited
                priority:null, // visit priority based on fscore
                queued:false // flag for whether the node has been queued for visiting
            };

            // STENCIL: determine whether this graph node should be the start
            //   point for the search
                visit_queue = [];
        }
    }
}
```

search_canvas.html

animate() function responsible for calling your iterate functions

Animation accomplished by iterative drawUpdate()->computeUpdate()

```
// render the world to the canvas element
drawRobotWorld();

// make sure the rrt iterations are not running faster than animation update
if (search_iterate && (Date.now() - cur_time > min_msec_between_iterations)) {

    // update time marker for last iteration update
    cur_time = Date.now();

    // update iteration count
    search_iter_count++;

    // call iteration for the selected search algorithm
    switch (search_alg) {
        case "depth-first":
        case "breadth-first":
        case "greedy-best-first":
        case "A-star":
            search_result = iterateGraphSearch();
            break;
        case "RRT":
            search_result = "failed";
            // (hack to speed viz)
            while (search_result == "failed")
                search_result = iterateRRT();
            break;
        case "RRT-connect":
            // (hack to speed viz) while (search_result == "failed")
                search_result = iterateRRTConnect();
            break;
        case "RRT-star":
            search_result = iterateRRTStar();
            break;
        default:
            console.warn('search_canvas: search algorithm not found, using rrt as default');
            search_result = iterateRRT();
            break;
    }
}
```

search_canvas.html

Ensure your implementations are isolated to single search steps

```
function iterateGraphSearch() {

    // STENCIL: implement a single iteration of a graph search algorithm
    //   for A-star (or DFS, BFS, Greedy Best-First)
    //   An asynch timing mechanism is used instead of a for loop to avoid
    //   blocking and non-responsiveness in the browser.
    //
    //   Return "failed" if the search fails on this iteration.
    //   Return "succeeded" if the search succeeds on this iteration.
    //   Return "iterating" otherwise.
    //
    //   Provided support functions:
    //
    //   testCollision - returns whether a given configuration is in collision
    //   drawHighlightedPathGraph - draws a path back to the start location
    //   draw_2D_configuration - draws a square at a given location
}
```

Including excessive loops may cause browser to become unresponsive