EECS 367 & ROB 320 Lab
KinEval Path Planning code overview

Administrative

- Let me know if you are not on the Slack workspace
- Assignment #0: ROS Pub/Sub
  - Due 11:59pm, Wednesday, January 18
- CI-Grader has run
  - Need: workflow survey (GitHub username, unique name), Github Classroom or specific private repo
- Next CI-Grader Run: Late Tonight(I will post on Slack when it’s done)

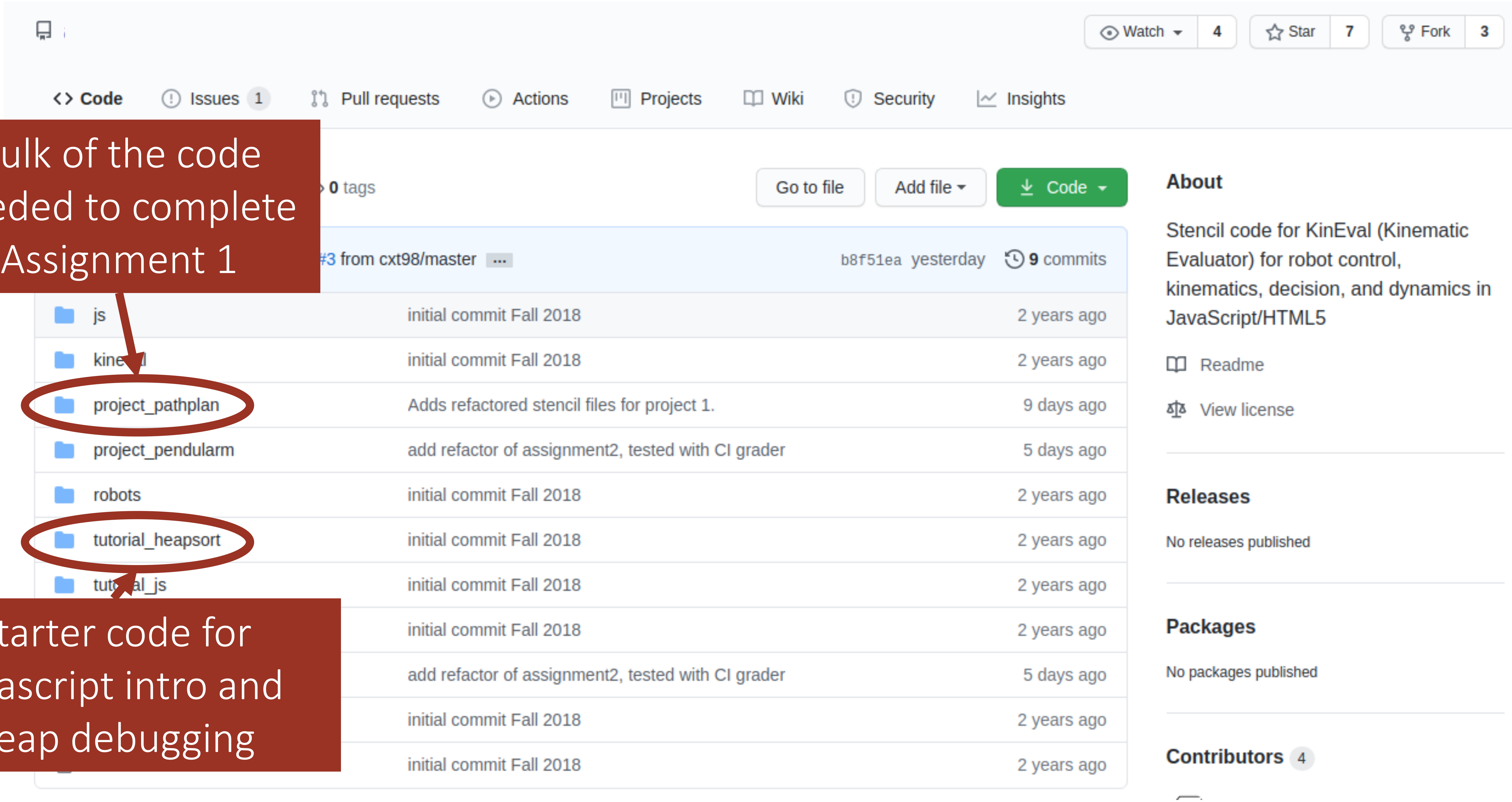
Lab Takeaways

1. Stencil overview
2. Walk through heap insert function
3. Validate implementation
4. Search canvas introduction
5. Data structure considerations

Assignment 1 Overview

		Assignment 1: 2D Path Planning
4	All	Heap implementation
6	All	A-star search
2	511	BFS
2	511	DFS
2	511	Greedy best-first

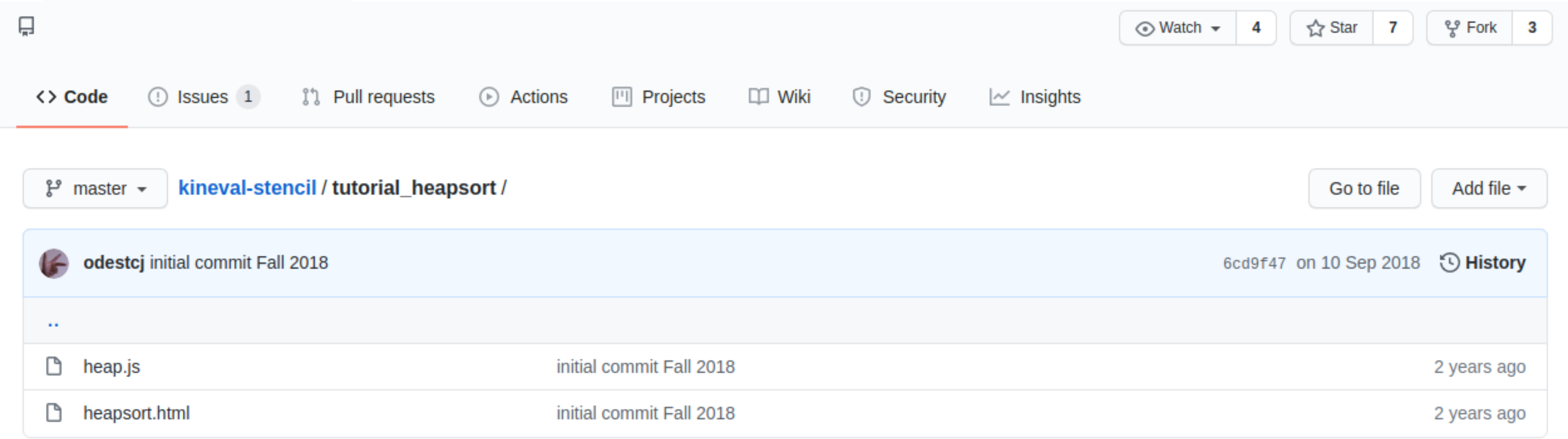
KinEval Stencil



Bulk of the code needed to complete Assignment 1

Starter code for Javascript intro and heap debugging

Heapsort Tutorial



Heapsort Tutorial HTML

heapsort.html

```
<!--
    A script element contains JavaScript code for the browser to execute.
        Script code could be in another file, specified in the "src" attribute,
        as in the case below for your heap code.

    This external source file should produce the object "minheaper" that has
        two function methods for inserting and extracting heap elements.
-->

<!--  you will want to uncomment this tag
<script src="heap.js"></script>
-->
```

Specifies JavaScript source file to make available in HTML. Uncomment this line!

Heapsort Tutorial HTML

heapsort.html

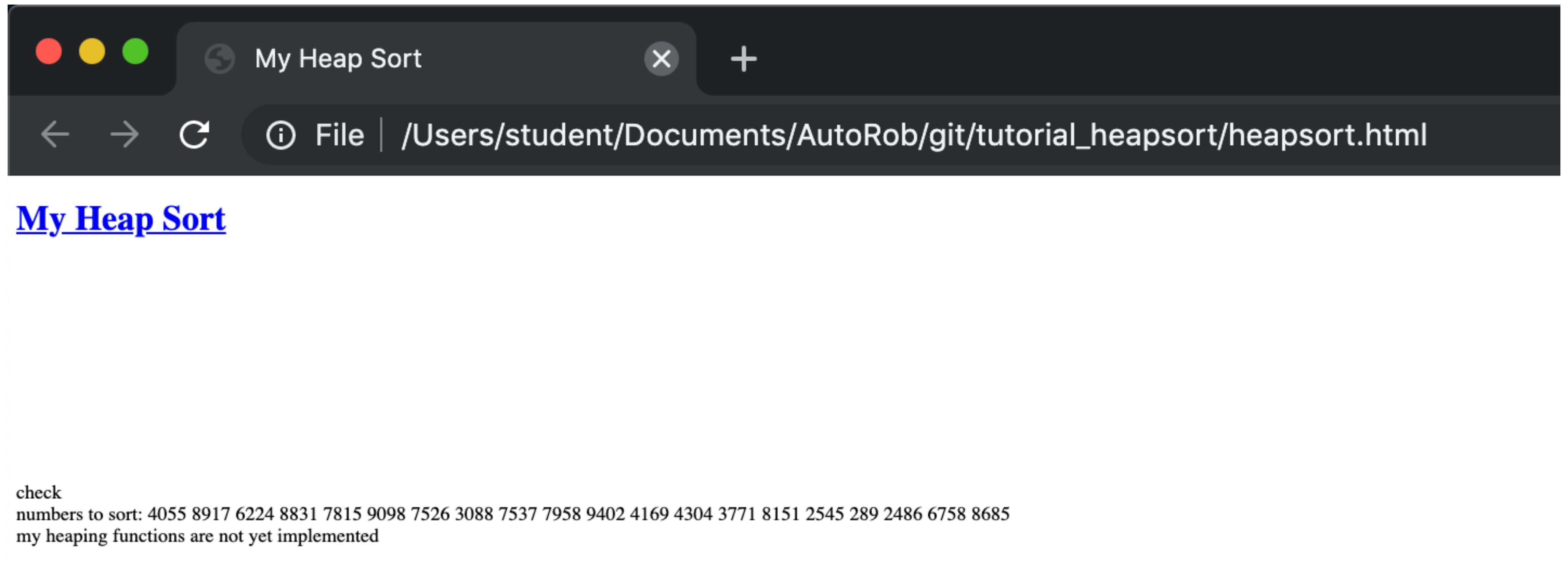
Represent heap as a JavaScript array

Repeatedly call heap insert method

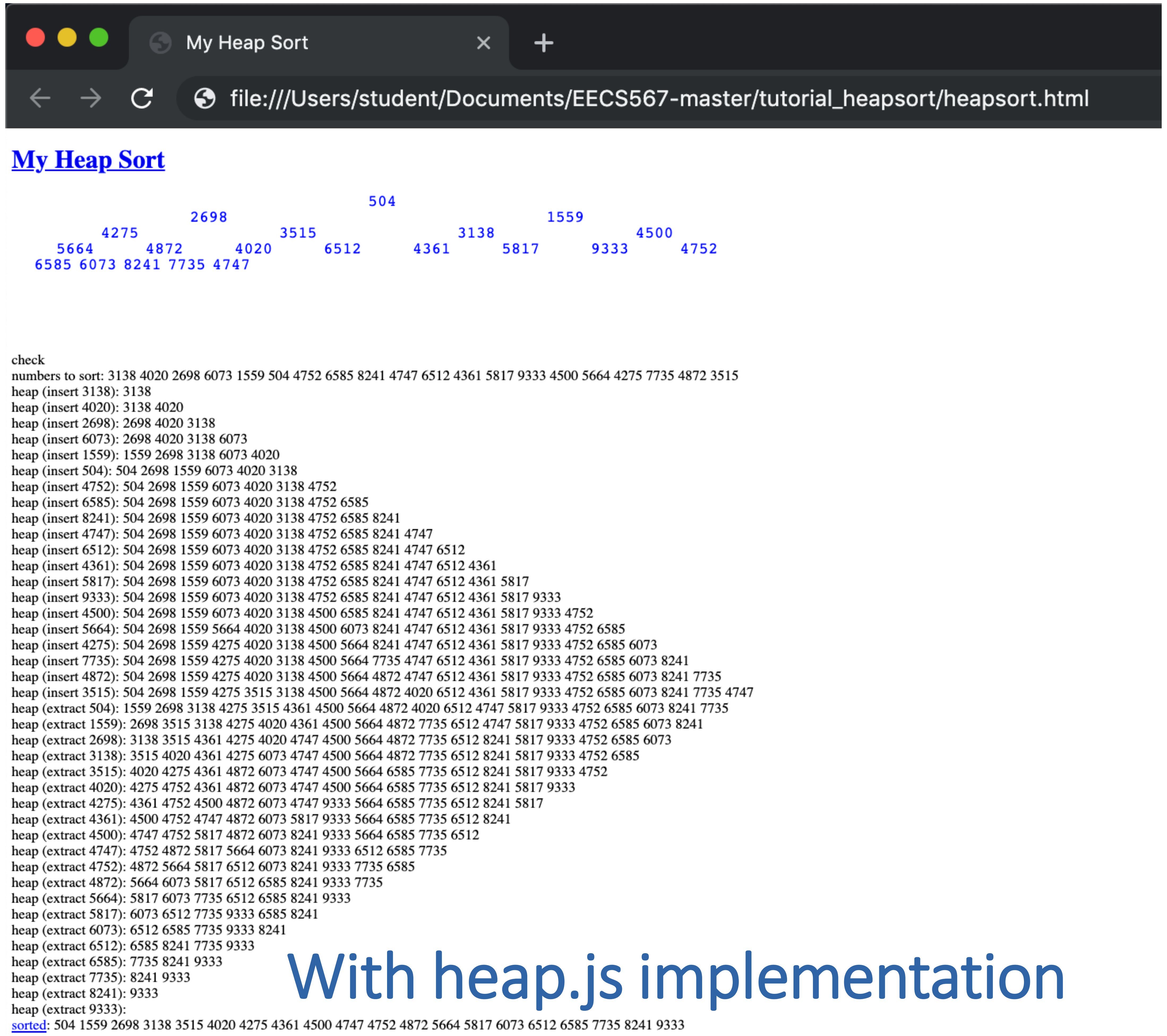
```
    console.log("building min binary heap from number array");
    numbers_heap = []; // create array for heap
    for (i=0;i<numbers.length;i++) {

        console.log("inserting number "+numbers[i]+" into the heap");
        minheaper.insert(numbers_heap,numbers[i]);

        console.log("appending current heap state to output object");
        output_string = "heap (insert " + numbers[i] + "): "; //
        for (j=0;j<numbers_heap.length;j++) {
            output_string += numbers_heap[j] + " "; //
        }
        addHTMLLine("output",output_string);
    }

```

Print state of heap to screen

Heapsort Tutorial Results



Without heap.js implementation



With heap.js implementation

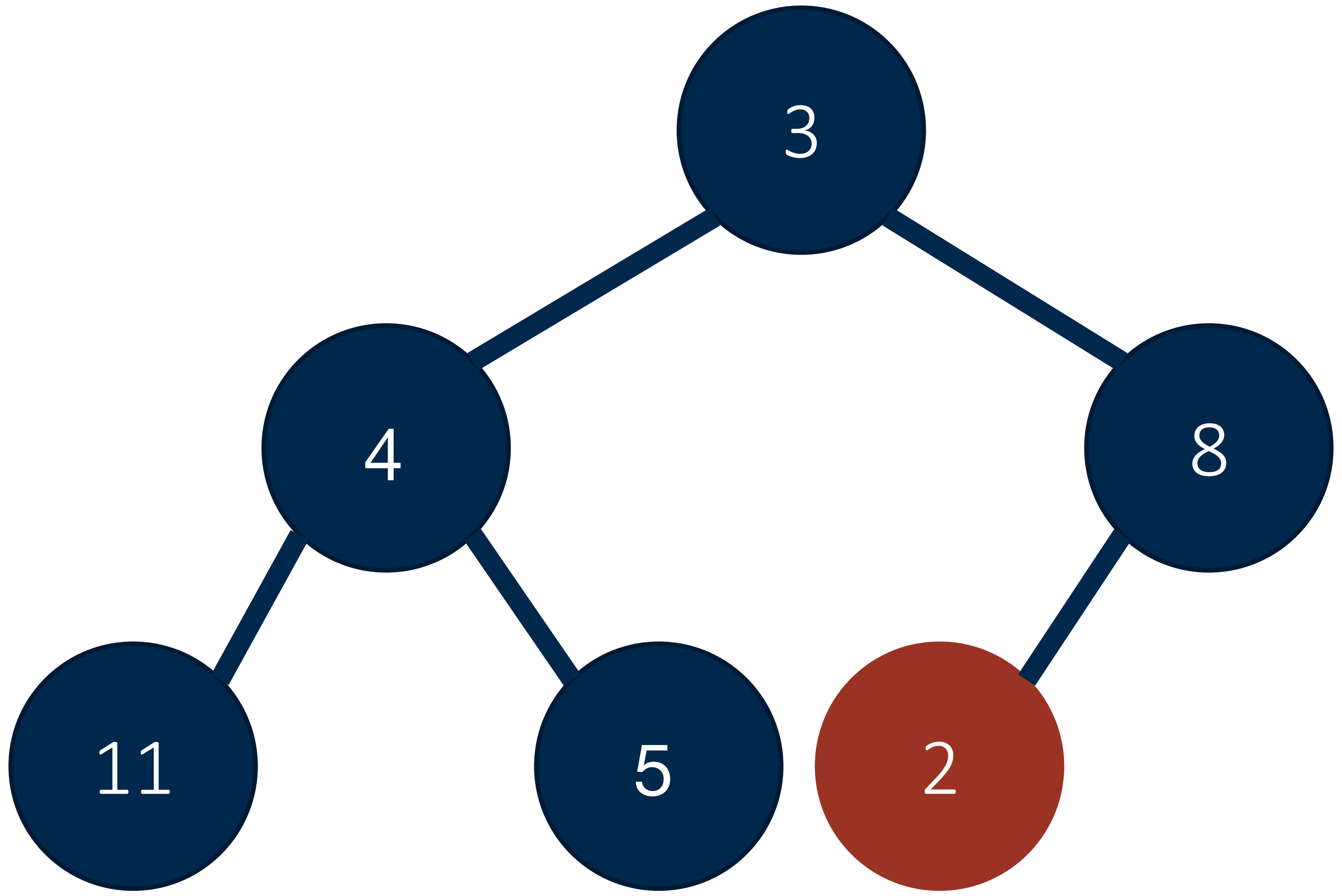
Heapsort Tutorial JavaScript

heap.js

```
// create empty object
minheaper = {};

// define insert function for min binary heap
function minheap_insert(heap, new_element) {

    // STENCIL: implement your min binary heap insert operation
}
```

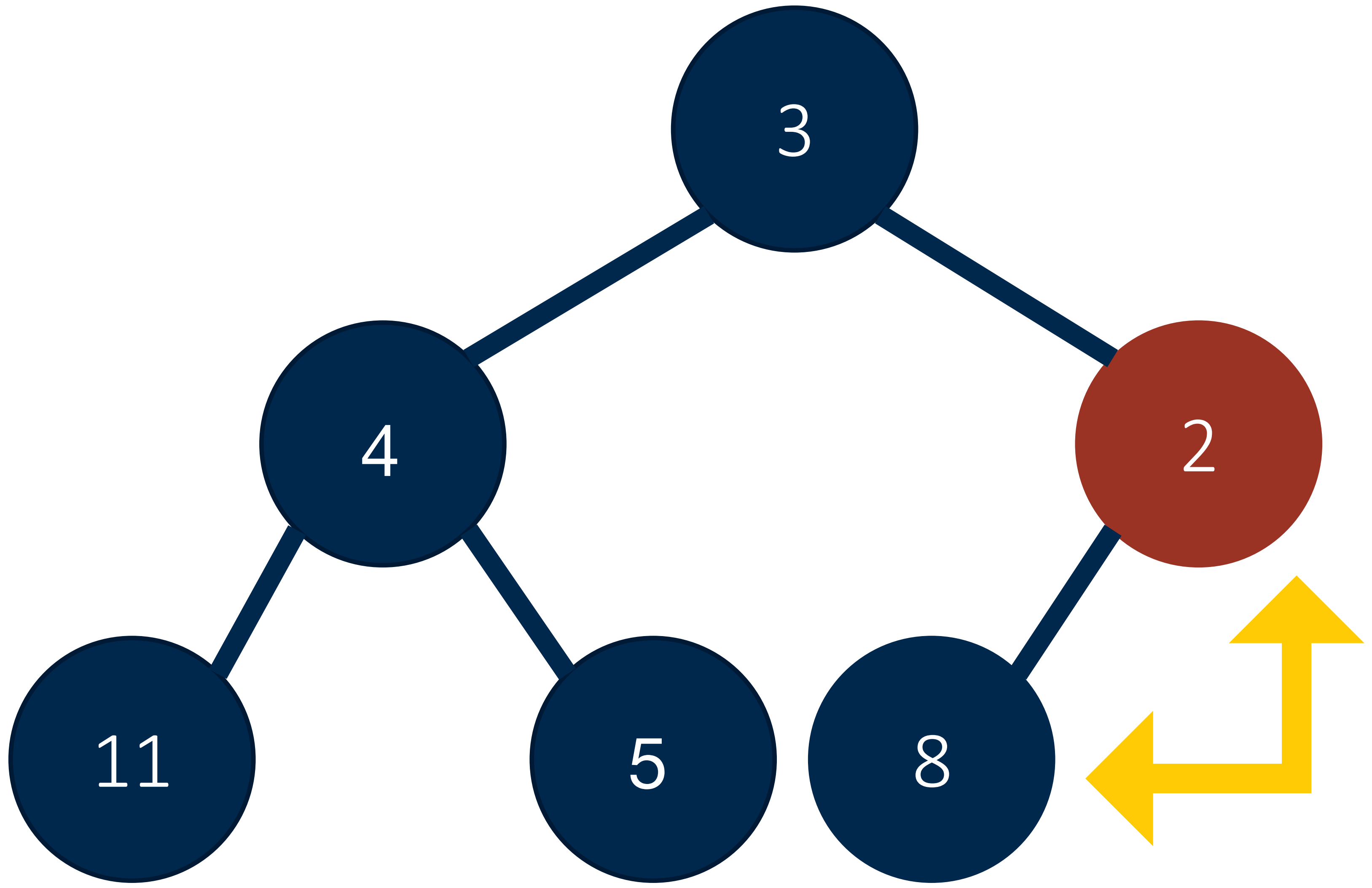
Heap Insert

1. Add new element into first open spot in tree (end of heap)



Heap Insert

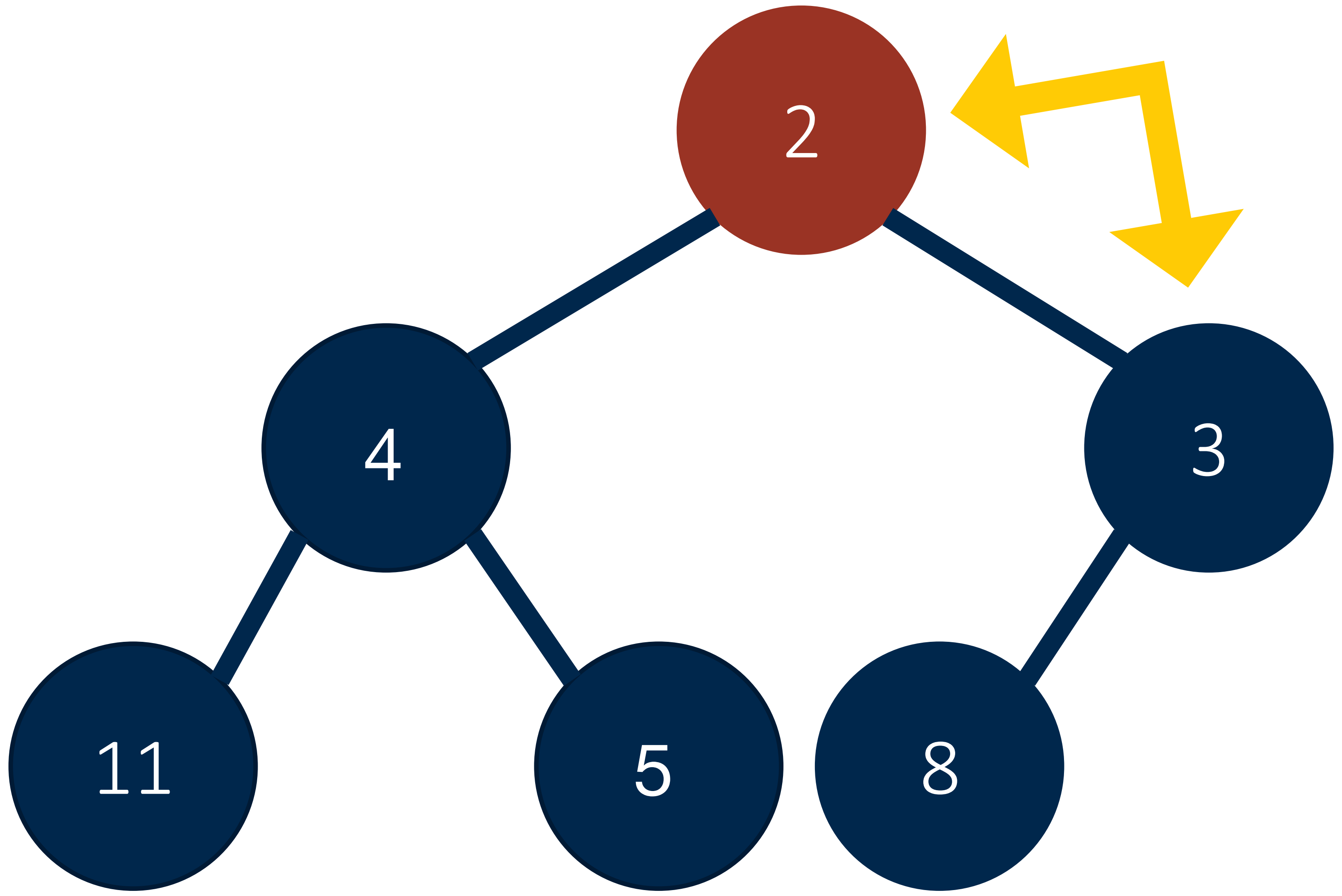
1. Add new element into first open spot in tree (end of heap)
2. If new element is smaller than parent, swap with parent



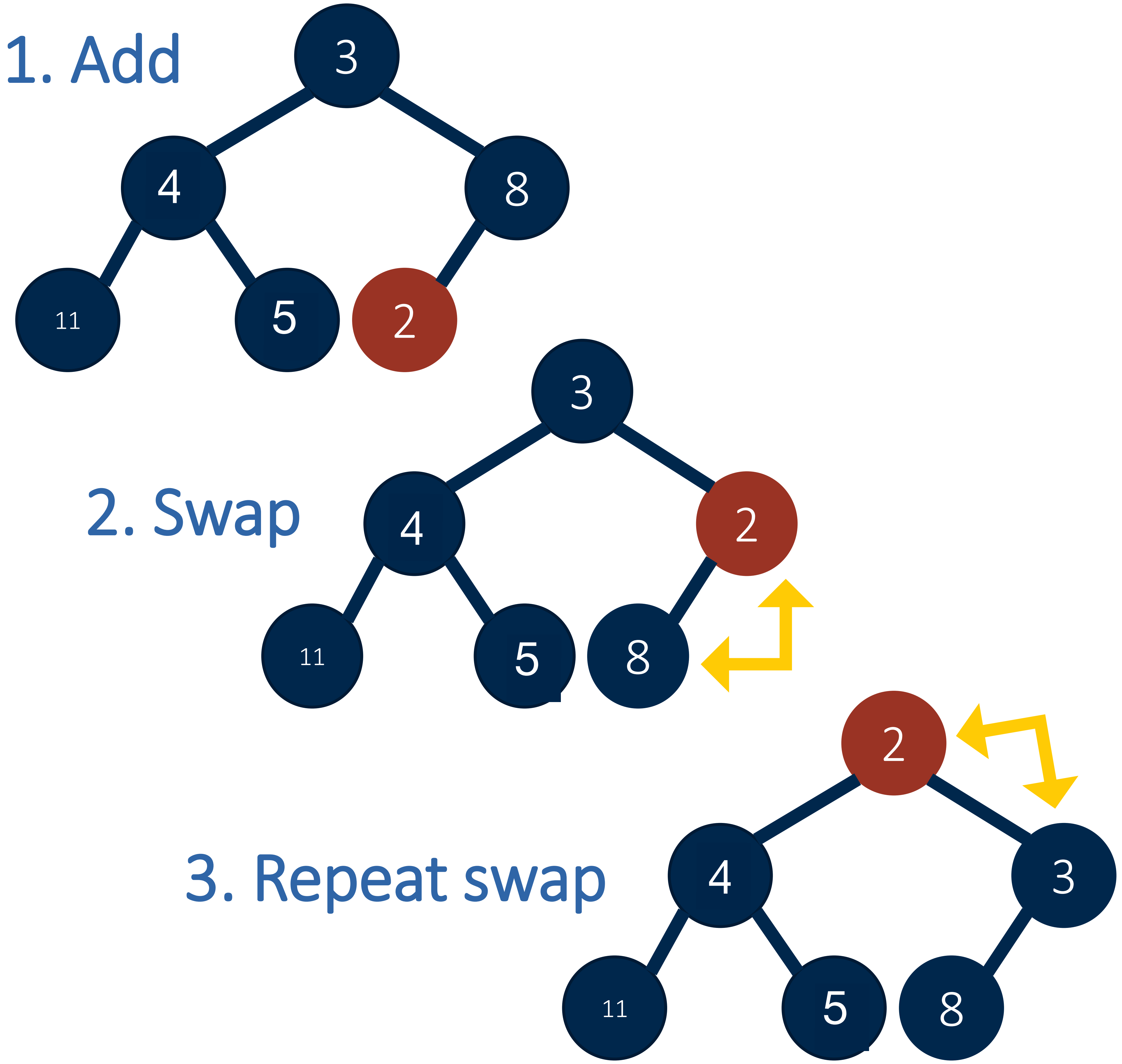
Heap Insert

1. Add new element into first open spot in tree (end of heap)
2. If new element is smaller than parent, swap with parent
3. Repeat step 2 until heap property holds

Heap property: For min heaps, the value of the parent node must always be less than or equal to the value of the child node

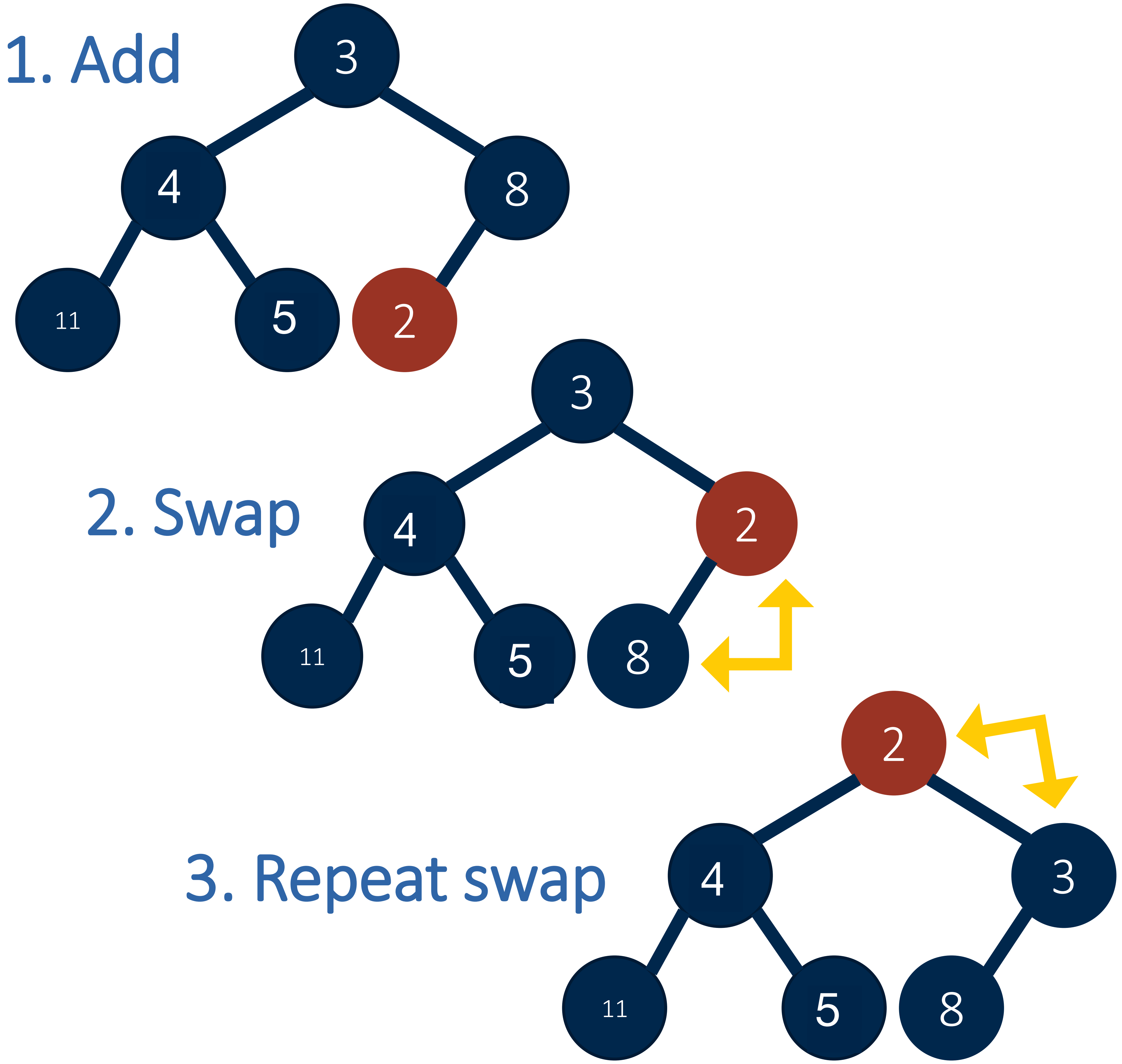


Heap Insert



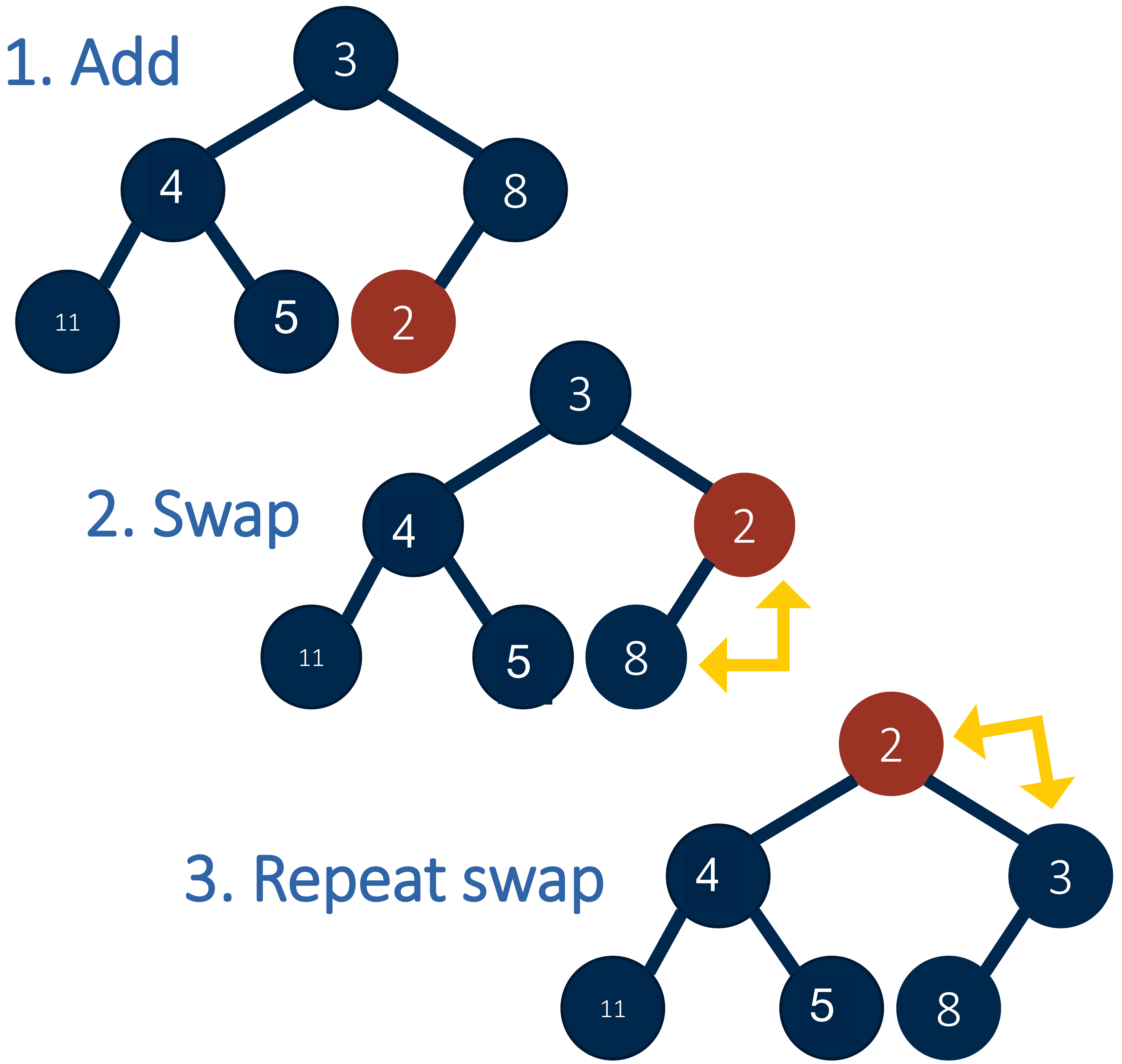

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    // TODO: Find index for new_element
    // TODO: Find index of new_element's parent

    // 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



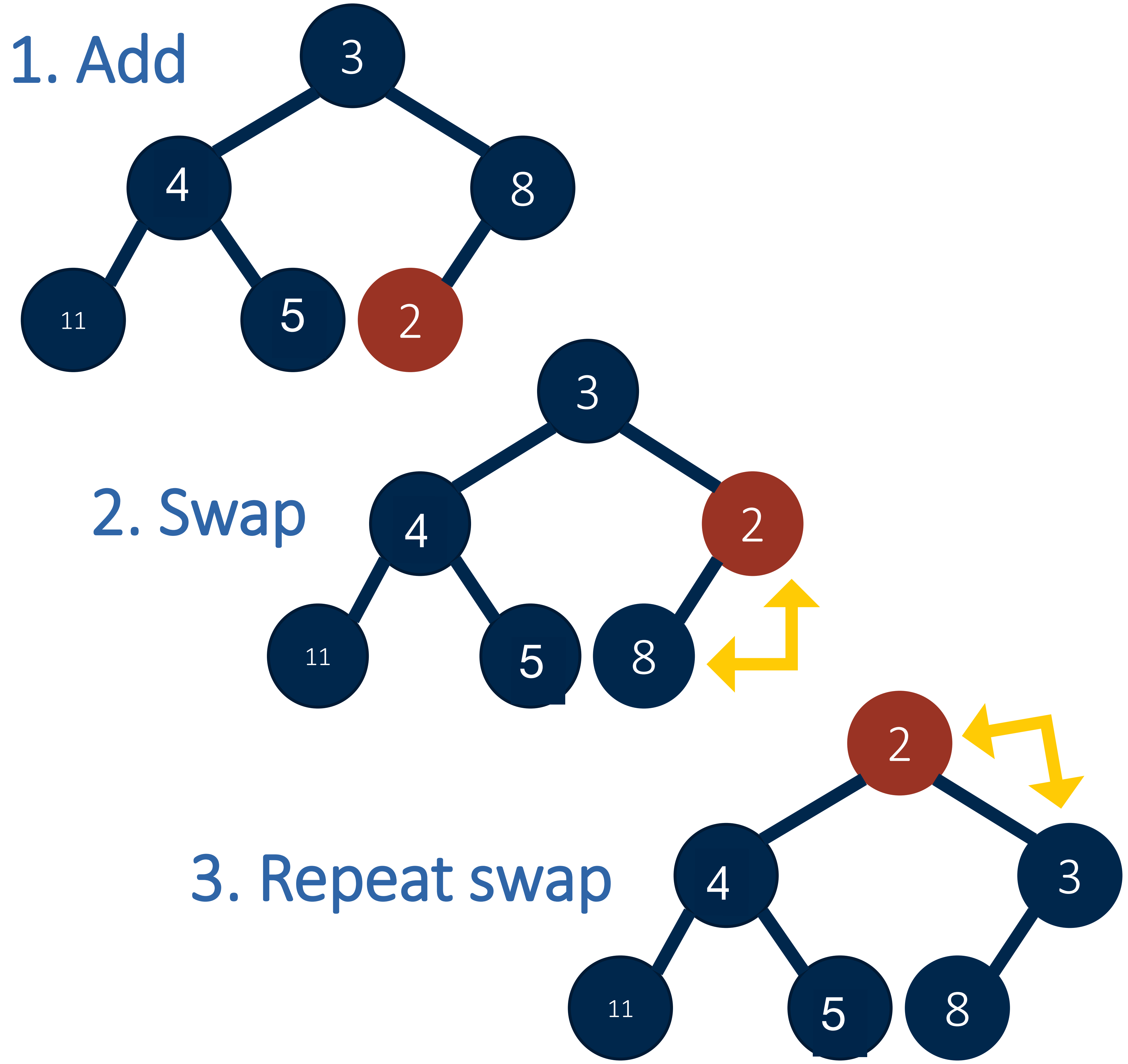

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    // TODO: Find index of new_element's parent

    // 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



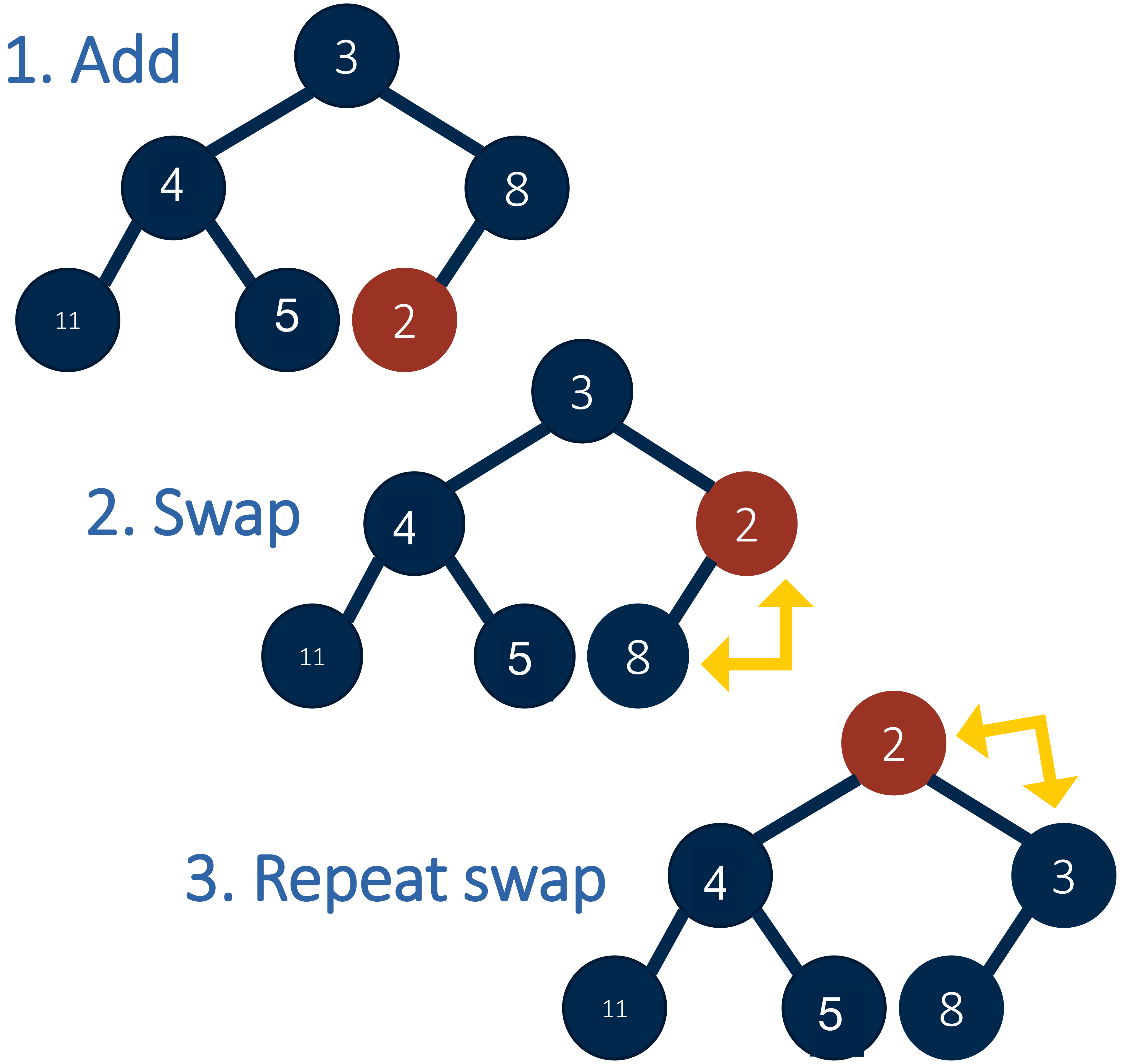

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    // 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



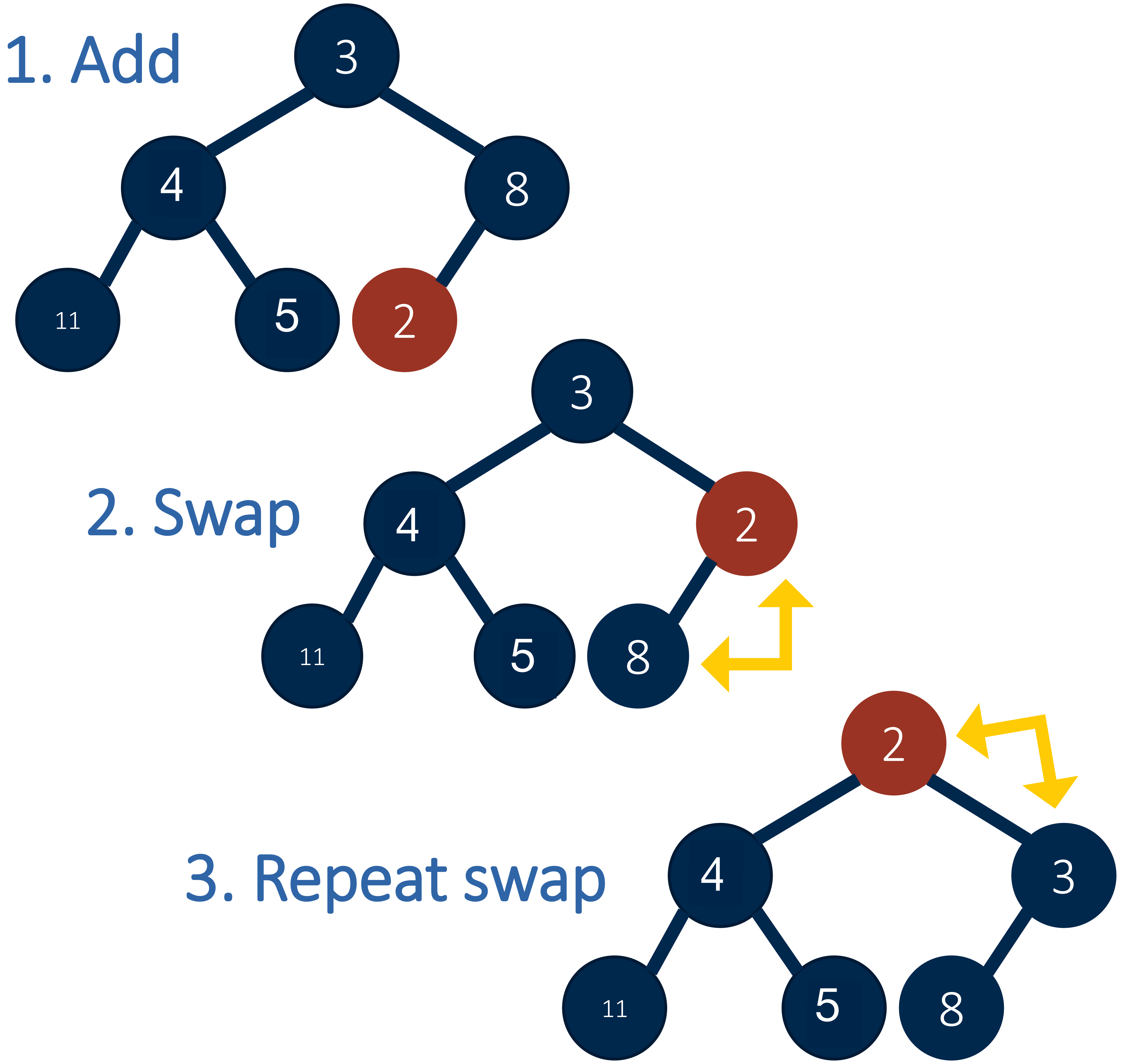

```javascript
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    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



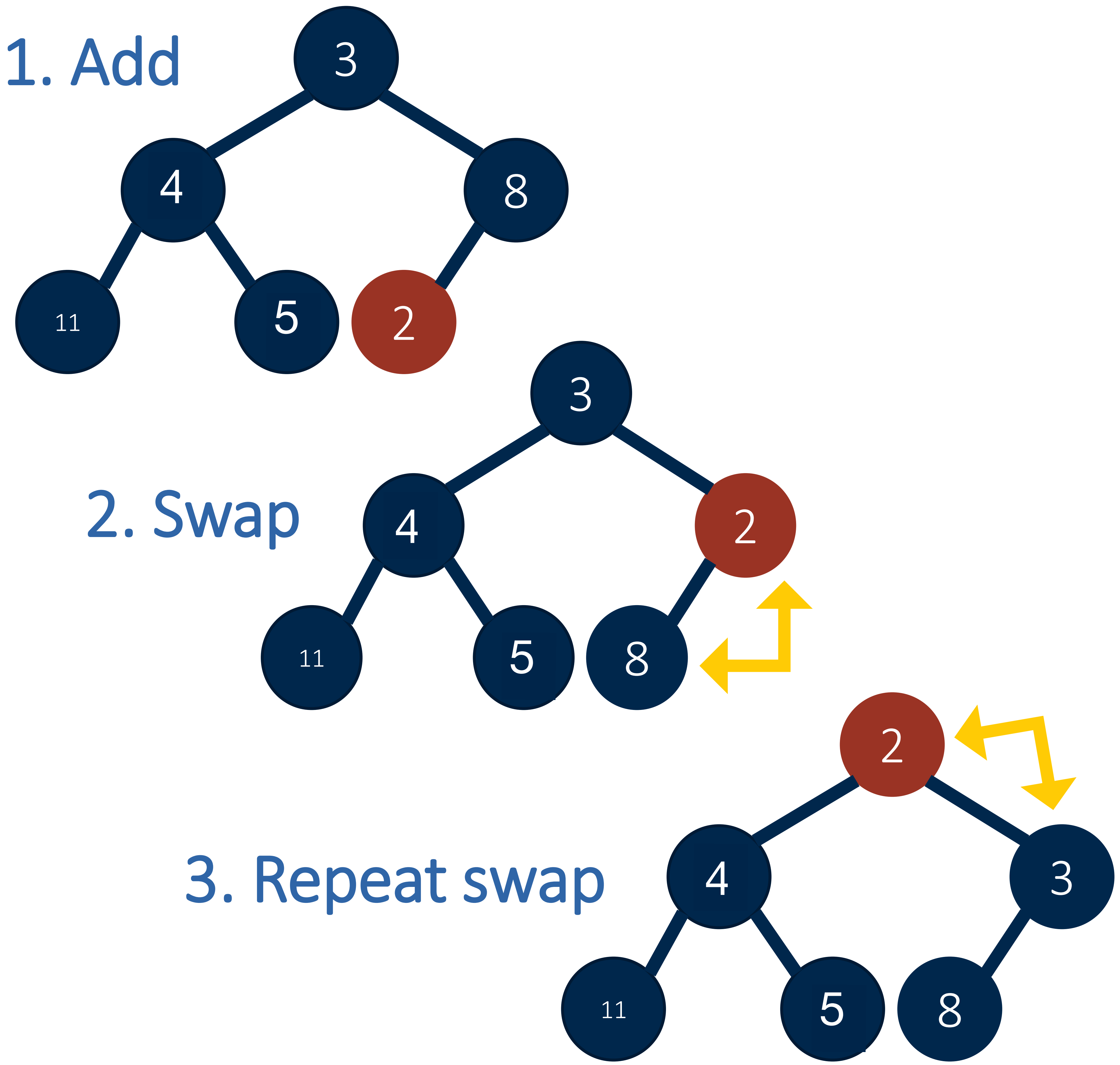

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    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



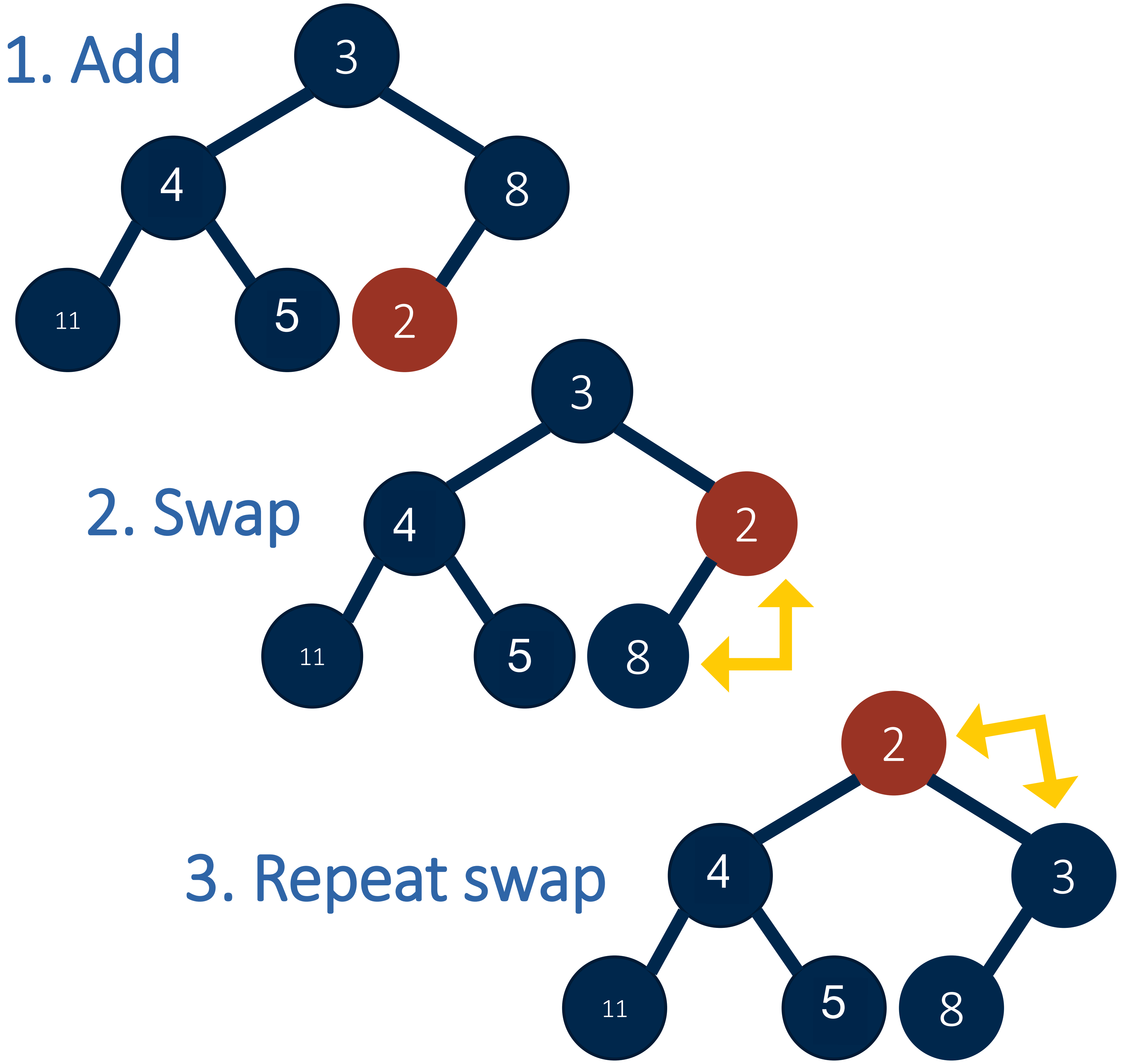

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    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



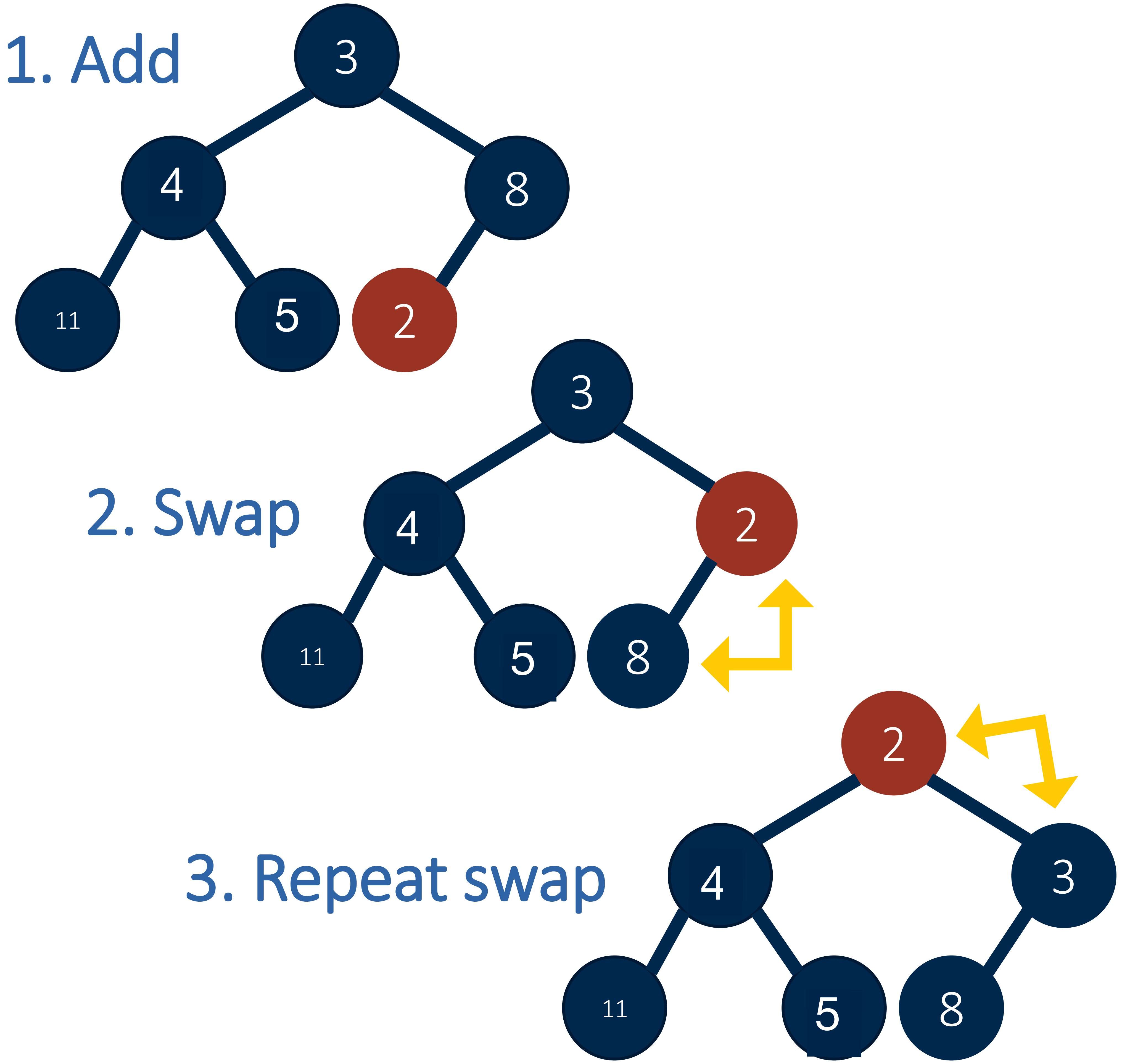

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    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



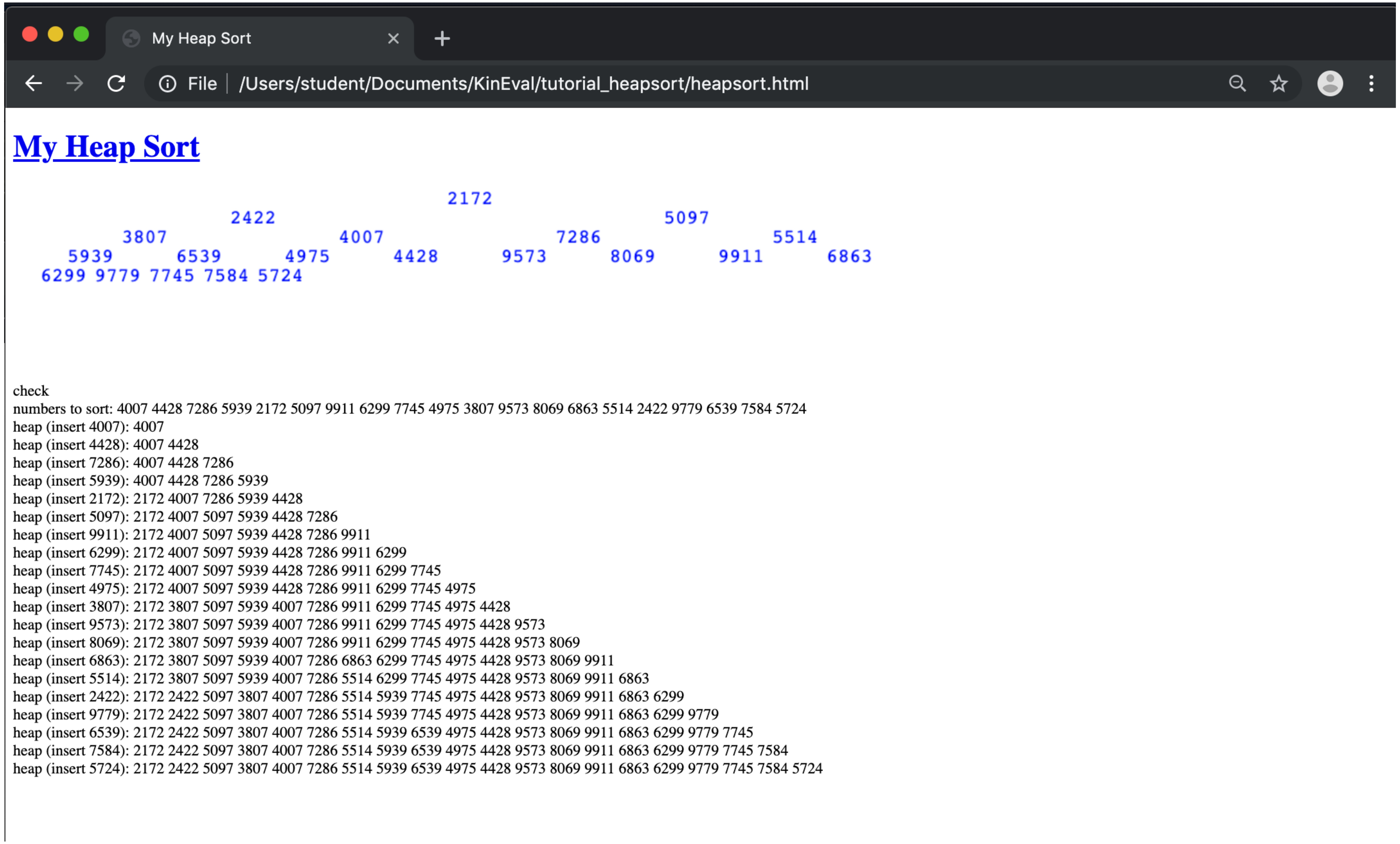

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    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]);
    }
}
```

Heap Insert Result



My Heap Sort

File | /Users/student/Documents/KinEval/tutorial_heapsort/heapsort.html

My Heap Sort

```
                              2172
              2422                            5097
       3807            4007           7286            5514
   5939    6539    4975    4428    9573    8069    9911    6863
 6299 9779 7745 7584 5724
```

check
numbers to sort: 4007 4428 7286 5939 2172 5097 9911 6299 7745 4975 3807 9573 8069 6863 5514 2422 9779 6539 7584 5724
heap (insert 4007): 4007
heap (insert 4428): 4007 4428
heap (insert 7286): 4007 4428 7286
heap (insert 5939): 4007 4428 7286 5939
heap (insert 2172): 2172 4007 7286 5939 4428
heap (insert 5097): 2172 4007 5097 5939 4428 7286
heap (insert 9911): 2172 4007 5097 5939 4428 7286 9911
heap (insert 6299): 2172 4007 5097 5939 4428 7286 9911 6299
heap (insert 7745): 2172 4007 5097 5939 4428 7286 9911 6299 7745
heap (insert 4975): 2172 4007 5097 5939 4428 7286 9911 6299 7745 4975
heap (insert 3807): 2172 3807 5097 5939 4007 7286 9911 6299 7745 4975 4428
heap (insert 9573): 2172 3807 5097 5939 4007 7286 9911 6299 7745 4975 4428 9573
heap (insert 8069): 2172 3807 5097 5939 4007 7286 9911 6299 7745 4975 4428 9573 8069
heap (insert 6863): 2172 3807 5097 5939 4007 7286 6863 6299 7745 4975 4428 9573 8069 9911
heap (insert 5514): 2172 3807 5097 5939 4007 7286 5514 6299 7745 4975 4428 9573 8069 9911 6863
heap (insert 2422): 2172 2422 5097 3807 4007 7286 5514 5939 7745 4975 4428 9573 8069 9911 6863 6299
heap (insert 9779): 2172 2422 5097 3807 4007 7286 5514 5939 7745 4975 4428 9573 8069 9911 6863 6299 9779
heap (insert 6539): 2172 2422 5097 3807 4007 7286 5514 5939 6539 4975 4428 9573 8069 9911 6863 6299 9779 7745
heap (insert 7584): 2172 2422 5097 3807 4007 7286 5514 5939 6539 4975 4428 9573 8069 9911 6863 6299 9779 7745 7584
heap (insert 5724): 2172 2422 5097 3807 4007 7286 5514 5939 6539 4975 4428 9573 8069 9911 6863 6299 9779 7745 7584 5724

Lab Takeaways

1. Stencil overview
2. Walk through heap insert function
3. Validate implementation
4. Search canvas introduction
5. Data structure considerations

Search Canvas Infrastructure

infrastructure.js

```
// 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;

// keep track of the last goal drawn on the canvas
goal = [10000,1000];

cheme = "default";
```

In the `initSearch()` function, which instantiates global variables and starts your algorithms

`q_start_config` = `q_init`
start location in world

`q_goal_config` = `q_goal`
goal location in world

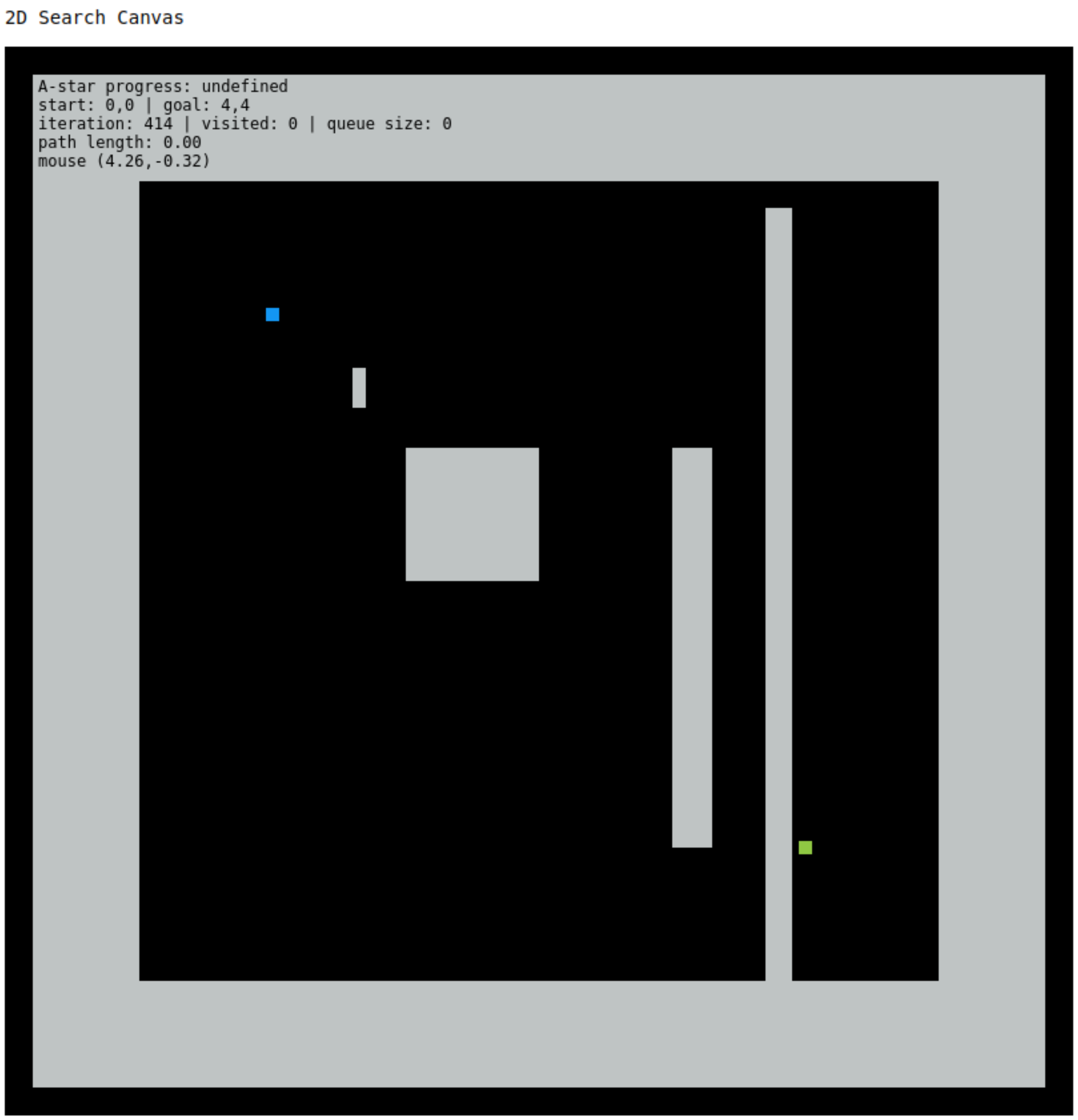
`q_init` and `q_goal` can be specified in URL

Search Canvas

Open search_canvas.html in your browser

Available URL parameters described in search_canvas.html file

World coordinates go from (-2, -2) to (7, 7)

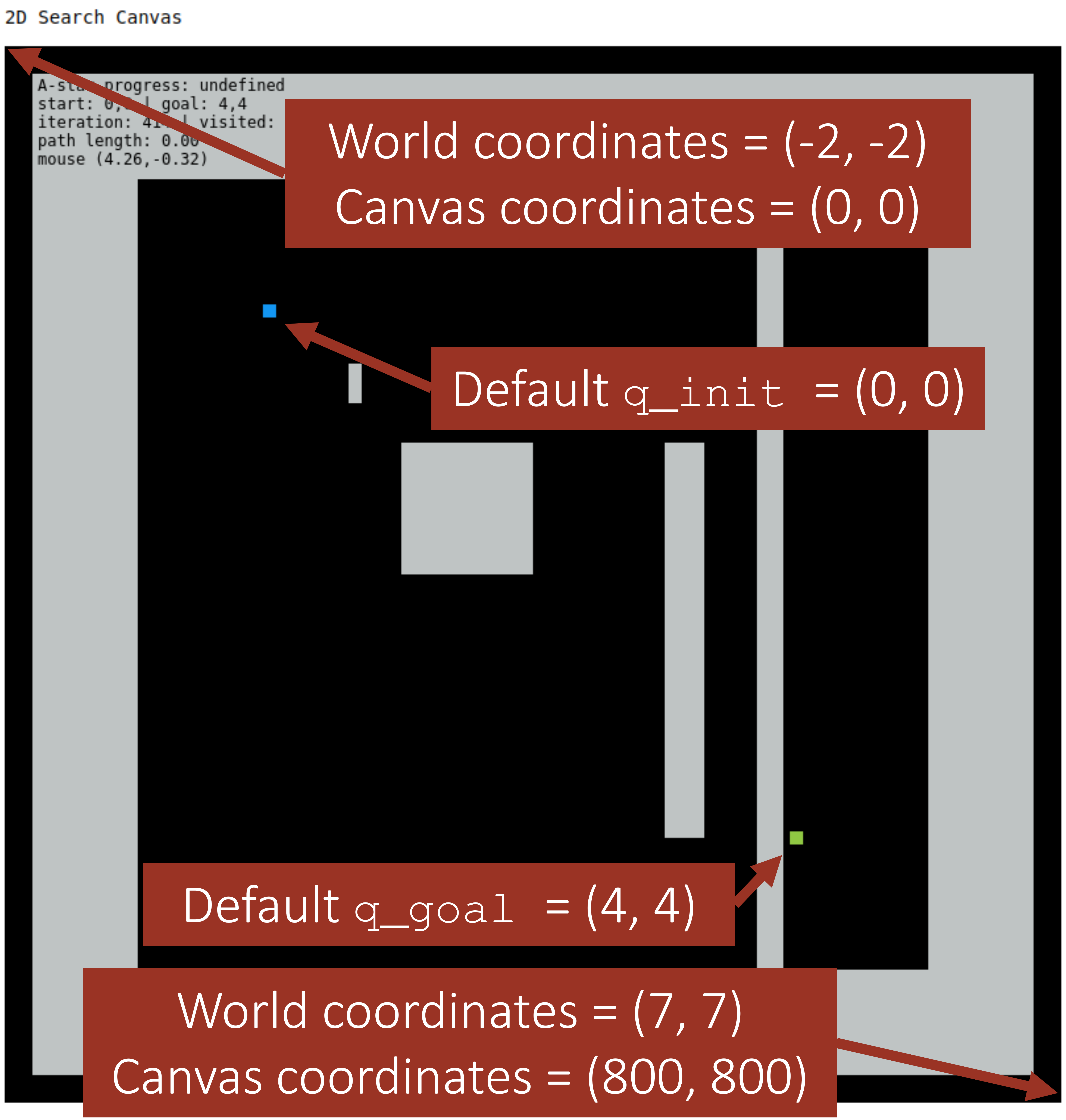


Search Canvas

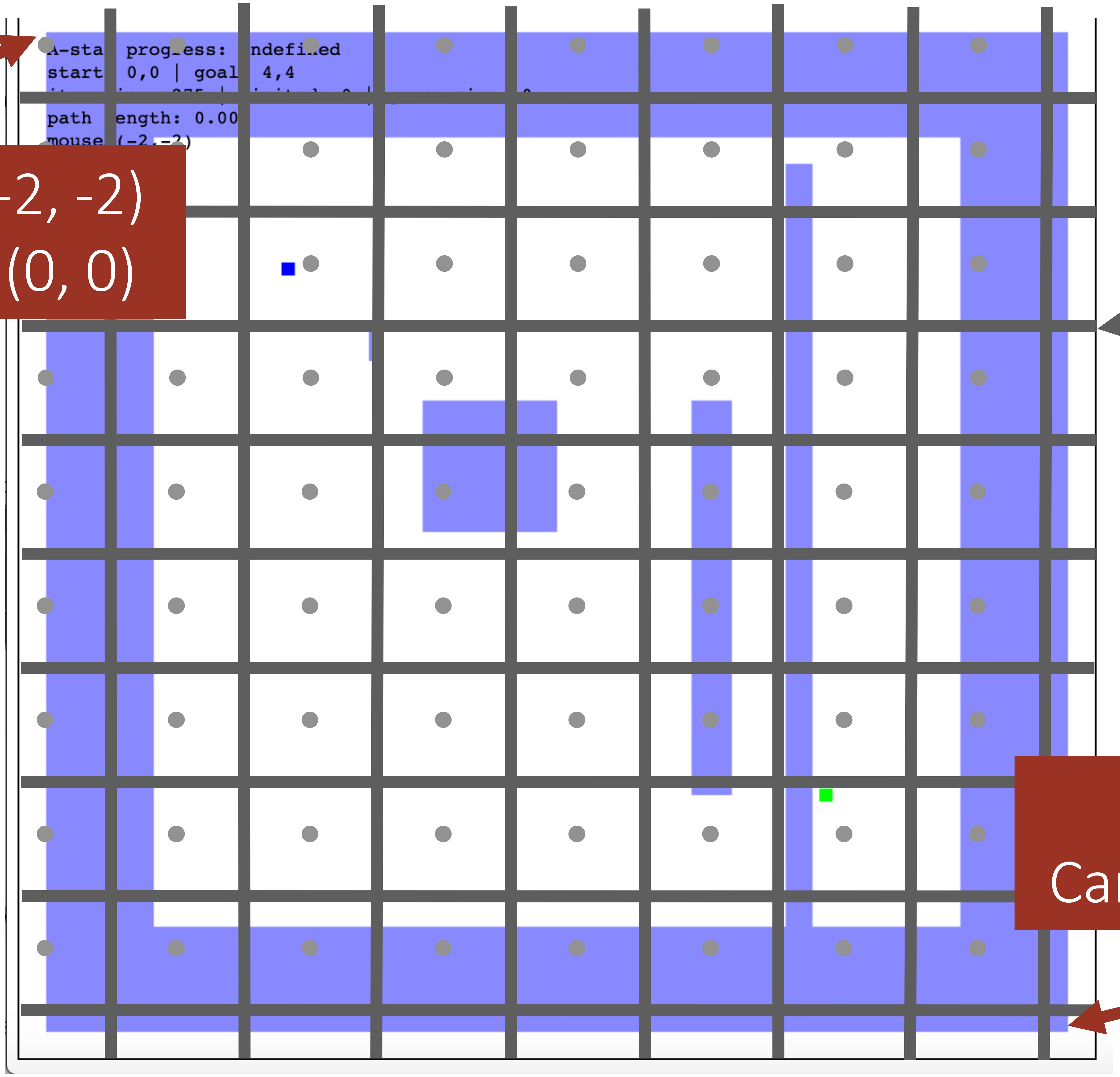
Open search_canvas.html in your browser

Available URL parameters described in search_canvas.html file

World coordinates go from (-2, -2) to (7, 7)



Search Graph

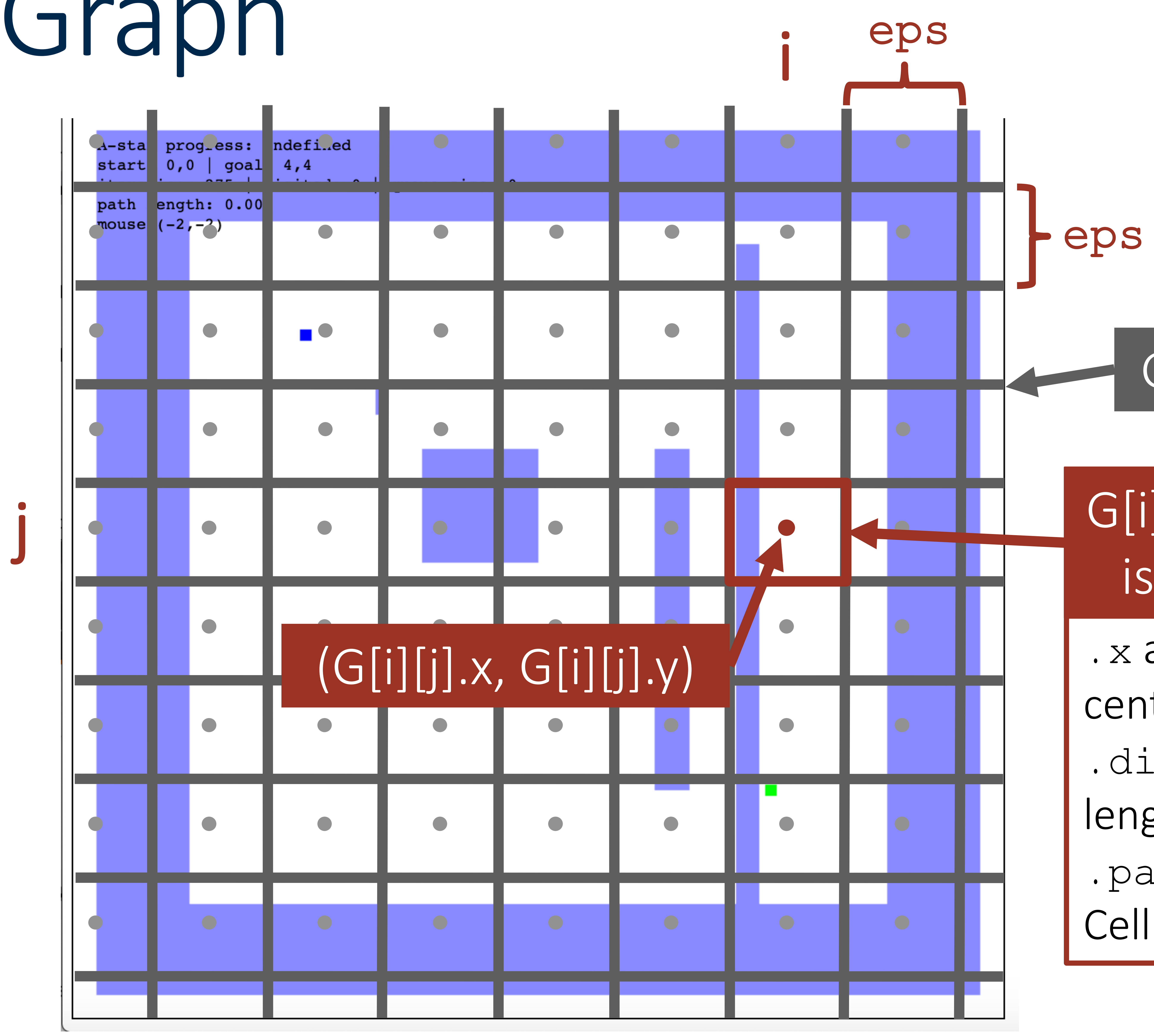


World coordinates = (-2, -2)
Canvas coordinates = (0, 0)

G: search graph

World coordinates = (7, 7)
Canvas coordinates = (800, 800)

Search Graph



G: search graph

G[i][j]: search graph cell is a JavaScript object

`.x` and `.y` specify cell center in world coordinates

`.distance` specifies path length to start through

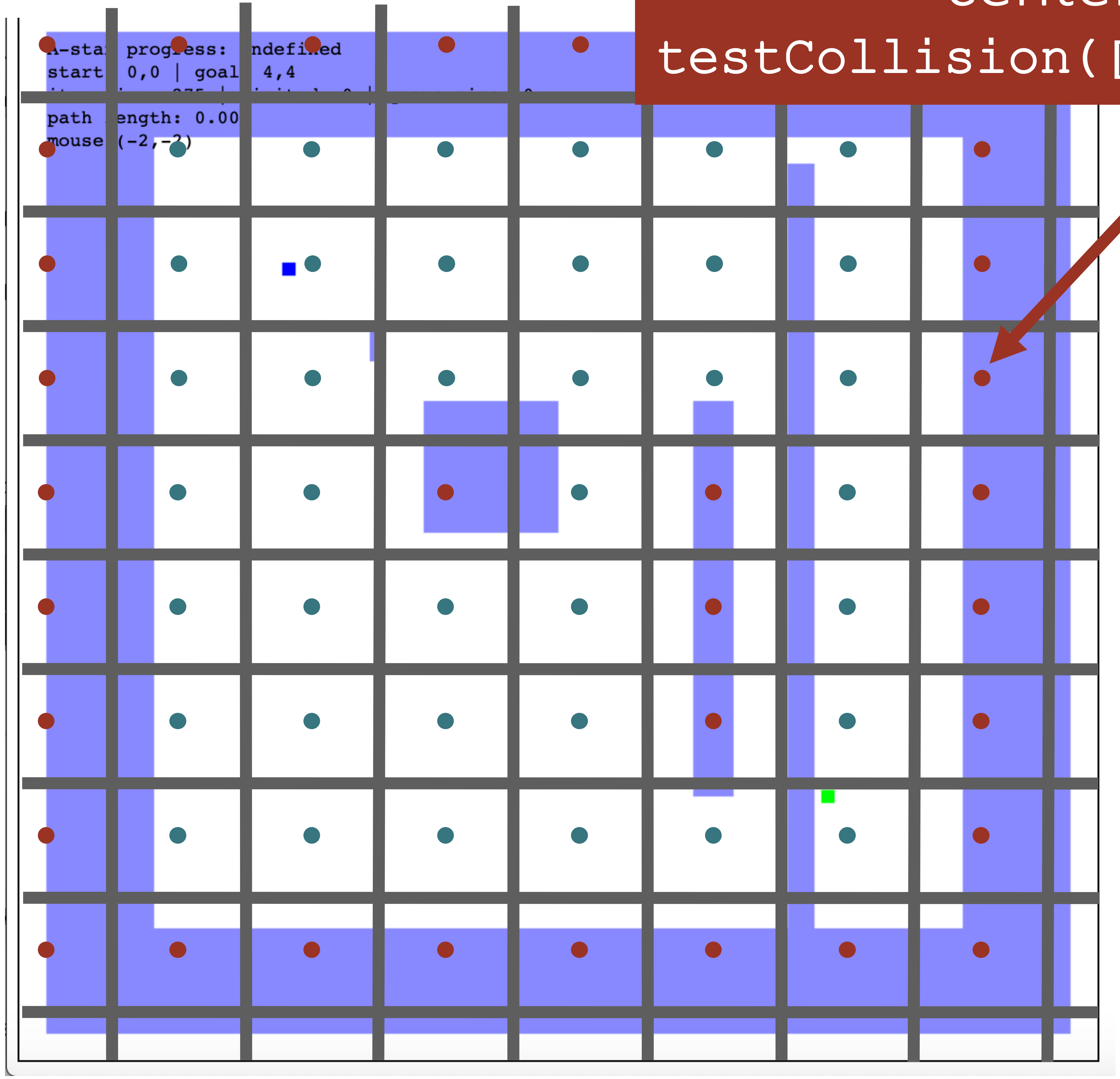
`.parent` node

Cell height and width = `eps`

Collisions

Test configuration at visited cell center for collision:

`testCollision([G[i][j].x,G[i][j].y])`



Graph Search Initialization

graph_search.js

```
function initSearchGraph() {

    // create the search queue
    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
        }
```

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

Graph Search Iteration

In `animate()` function, which is responsible for calling your iterate functions

draw.js

```
    // 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":
```

Graph Search Iteration

Ensure your implementations are isolated to **single search steps**! Including excessive loops may cause browser to become unresponsive.

graph_search.js

```
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
}
```

Once search has completed, turn off iteration: `search_iterate = false;`

Lab Takeaways

1. Stencil overview
2. Walk through heap insert function
3. Validate implementation
4. Search canvas introduction
5. Data structure considerations