EECS 367 & ROB 320 Lab
KinEval Path Planning code overview

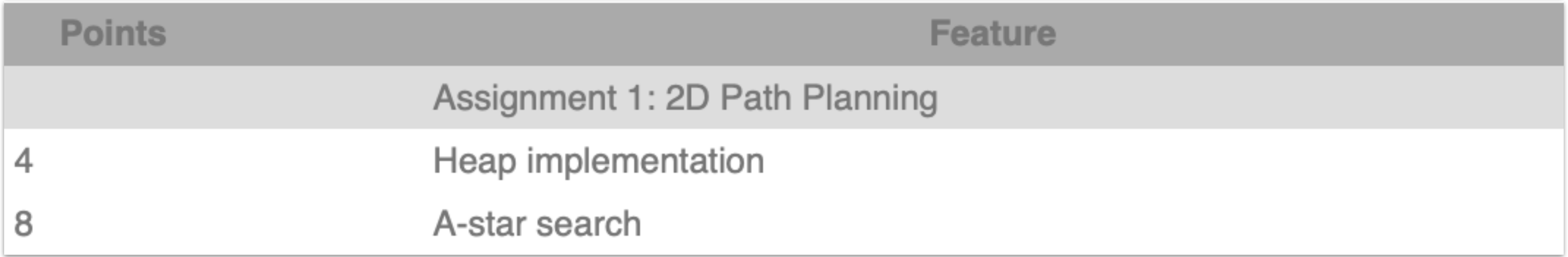
Administrative

- Assignment #1: Path Planning
    - Due 11:59pm, Friday, January 21
- Send Anthony (topipari) your Git repo (through slack)

Lab Takeaways

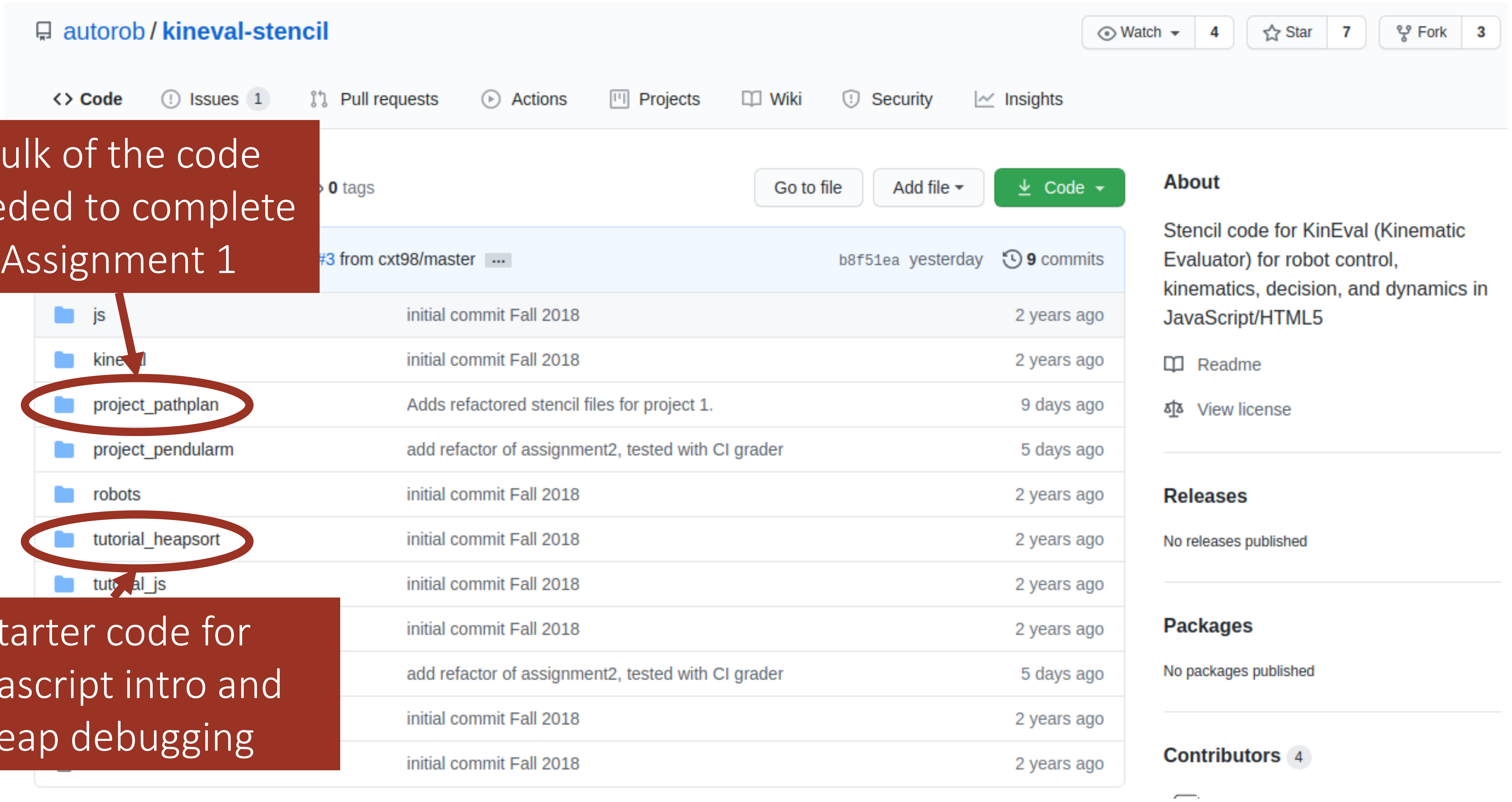
1. Stencil overview
2. Walk through heap insert function
3. Validate implementation
4. Search canvas introduction
5. Data structure considerations

Assignment 1 Overview



Points	Feature
	Assignment 1: 2D Path Planning
4	Heap implementation
8	A-star search

KinEval Stencil



autorob / kineval-stencil

Watch 4 | Star 7 | Fork 3

Code | Issues 1 | Pull requests | Actions | Projects | Wiki | Security | Insights

0 tags | Go to file | Add file | Code

#3 from cxt98/master ... b8f51ea yesterday 9 commits

js	initial commit Fall 2018	2 years ago
kine[illegible]l	initial commit Fall 2018	2 years ago
project_pathplan	Adds refactored stencil files for project 1.	9 days ago
project_pendularm	add refactor of assignment2, tested with CI grader	5 days ago
robots	initial commit Fall 2018	2 years ago
tutorial_heapsort	initial commit Fall 2018	2 years ago
tut[illegible]al_js	initial commit Fall 2018	2 years ago
	initial commit Fall 2018	2 years ago
	add refactor of assignment2, tested with CI grader	5 days ago
	initial commit Fall 2018	2 years ago
	initial commit Fall 2018	2 years ago

About

Stencil code for KinEval (Kinematic Evaluator) for robot control, kinematics, decision, and dynamics in JavaScript/HTML5

Readme

View license

Releases

No releases published

Packages

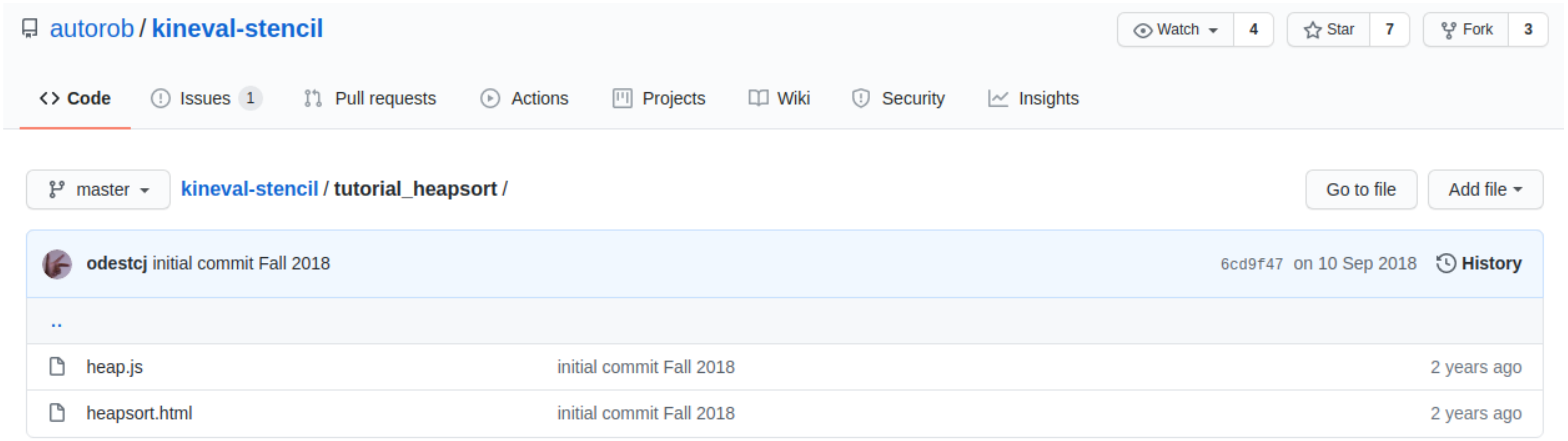
No packages published

Contributors 4

Bulk of the code needed to complete Assignment 1

Starter code for Javascript intro and heap debugging

Heapsort Tutorial



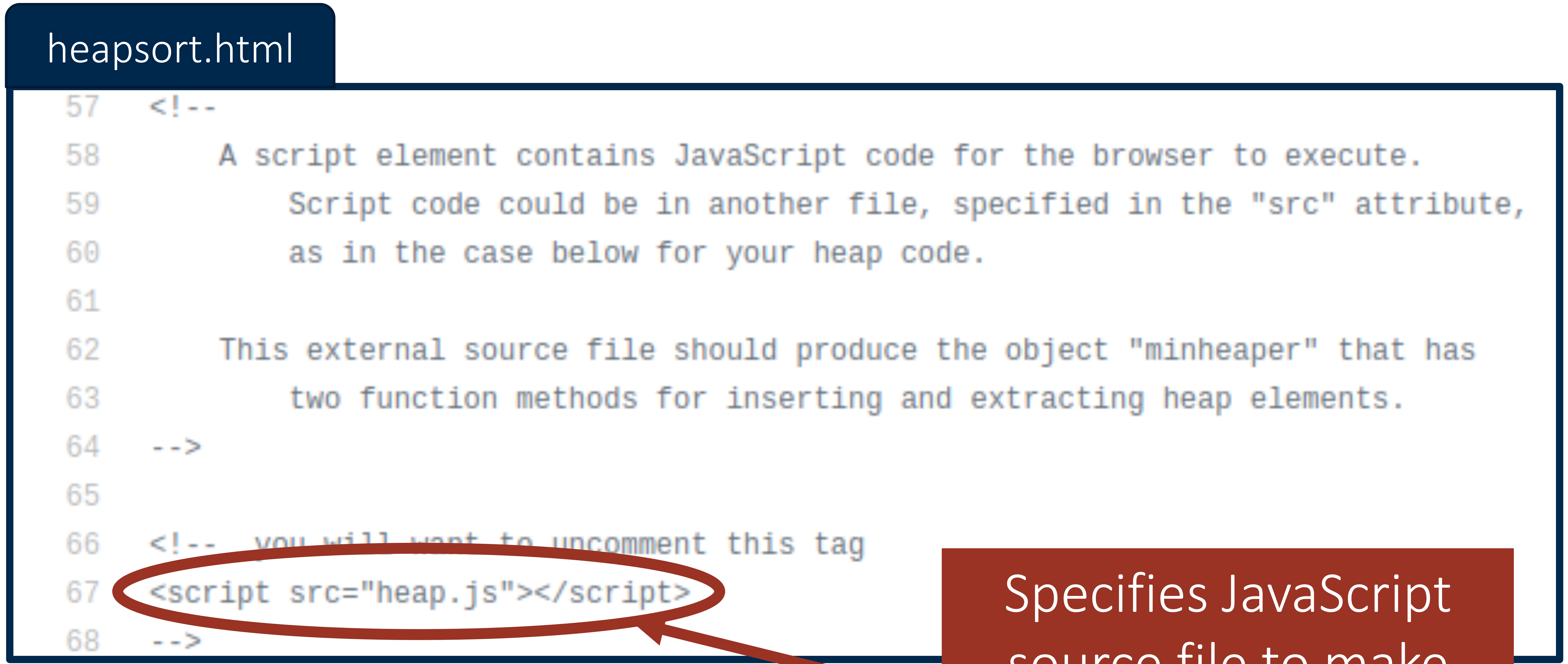
kineval-stencil / tutorial_heapsort /

odestcj initial commit Fall 2018 6cd9f47 on 10 Sep 2018 History

..

heap.js	initial commit Fall 2018	2 years ago
heapsort.html	initial commit Fall 2018	2 years ago

Heapsort Tutorial HTML

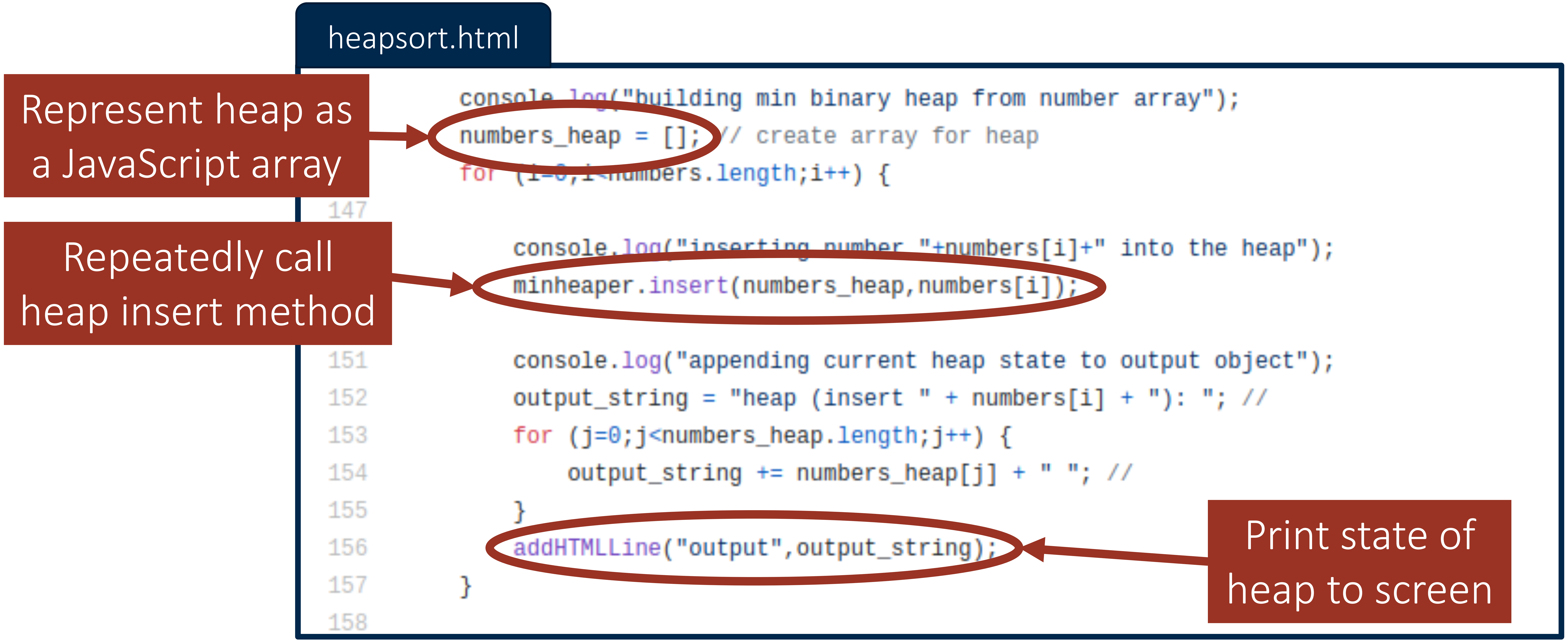


heapsort.html

```
<!--
    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

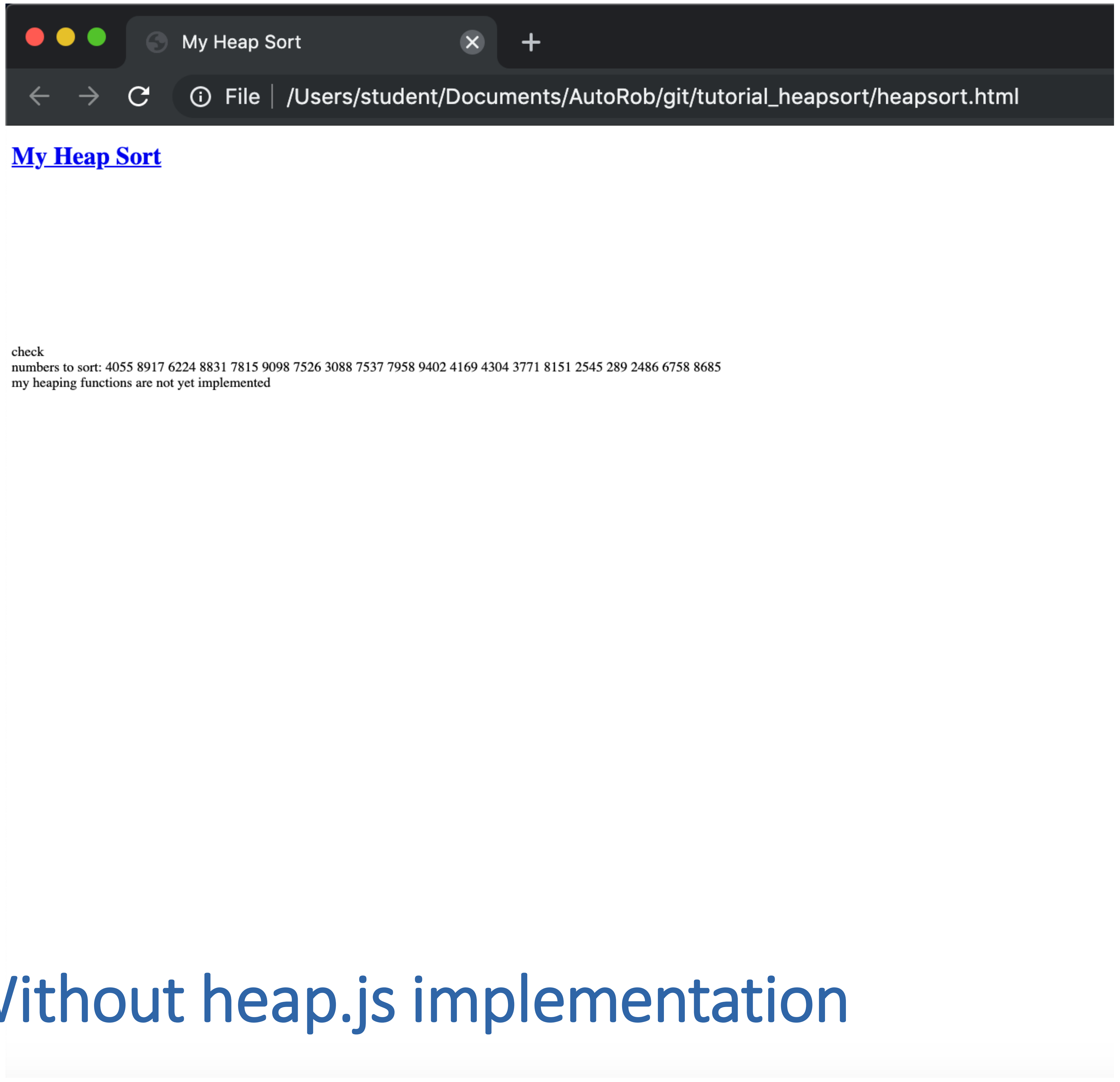
```
console.log("building min binary heap from number array");
numbers_heap = []; // create array for heap
for (i=0;i<numbers.length;i++) {

    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);
}
```

Represent heap as a JavaScript array

Repeatedly call heap insert method

Print state of heap to screen

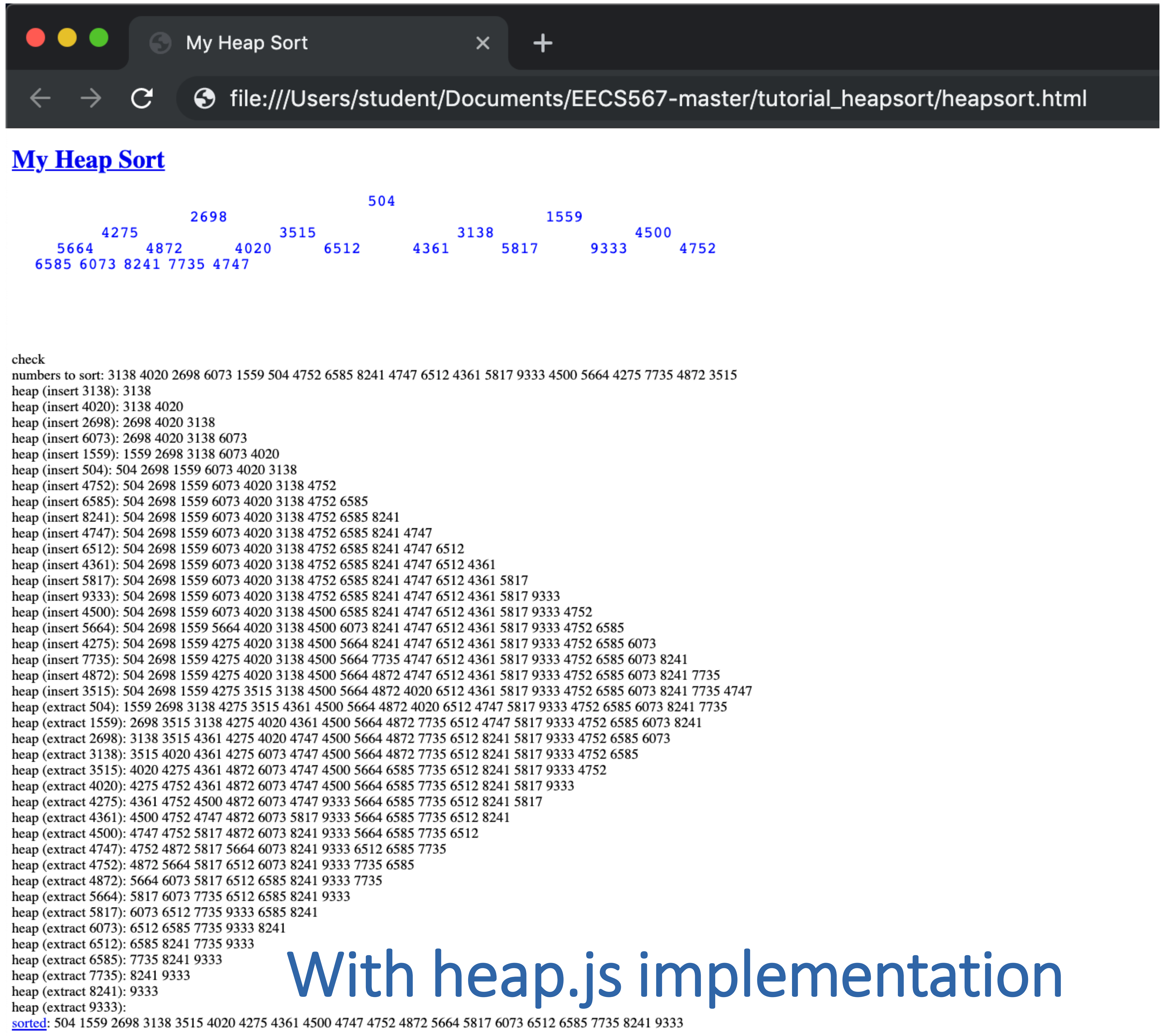
Heapsort Tutorial Results


My Heap Sort
File | /Users/student/Documents/AutoRob/git/tutorial_heapsort/heapsort.html

[My Heap Sort]

```
check
numbers to sort: 4055 8917 6224 8831 7815 9098 7526 3088 7537 7958 9402 4169 4304 3771 8151 2545 289 2486 6758 8685
my heaping functions are not yet implemented
```

Without heap.js implementation


My Heap Sort
file:///Users/student/Documents/EECS567-master/tutorial_heapsort/heapsort.html

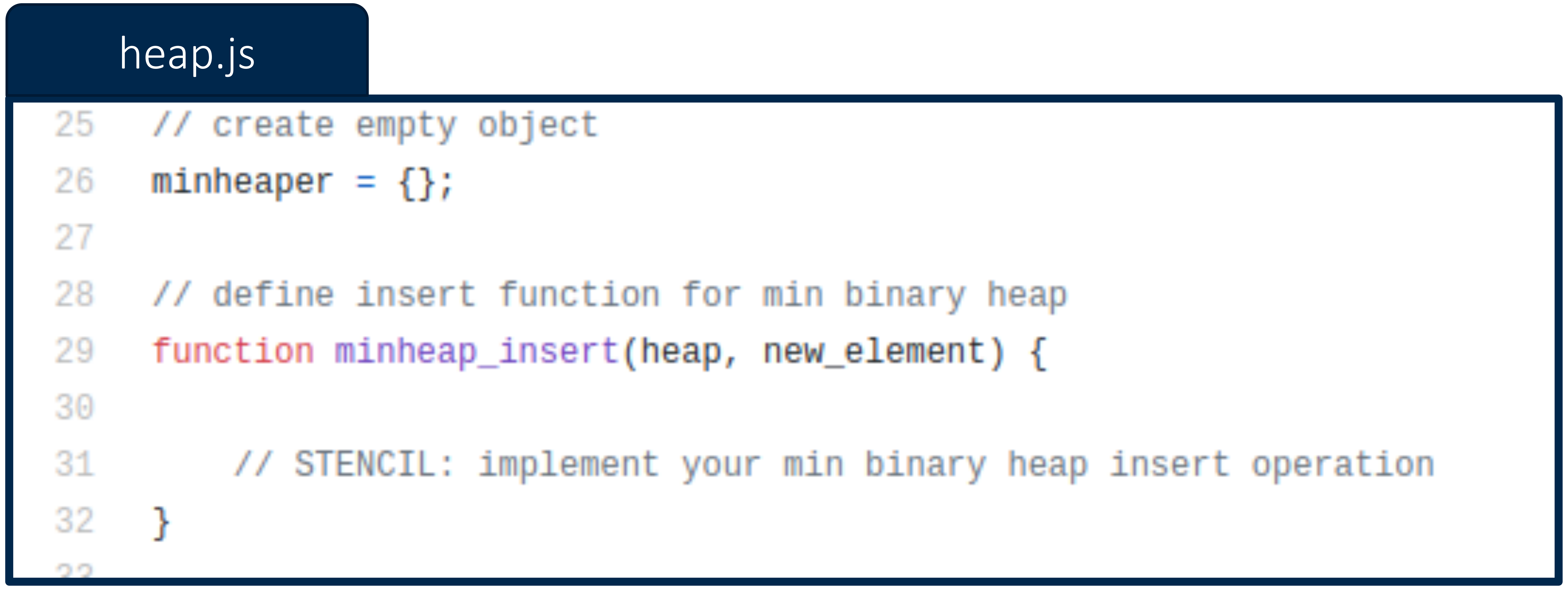
[My Heap Sort]

```
                                     504
                 2698                                    1559
       4275                3515                3138                4500
  5664      4872      4020      6512      4361      5817      9333      4752
6585 6073 8241 7735 4747
```

```
check
numbers to sort: 3138 4020 2698 6073 1559 504 4752 6585 8241 4747 6512 4361 5817 9333 4500 5664 4275 7735 4872 3515
heap (insert 3138): 3138
heap (insert 4020): 3138 4020
heap (insert 2698): 2698 4020 3138
heap (insert 6073): 2698 4020 3138 6073
heap (insert 1559): 1559 2698 3138 6073 4020
heap (insert 504): 504 2698 1559 6073 4020 3138
heap (insert 4752): 504 2698 1559 6073 4020 3138 4752
heap (insert 6585): 504 2698 1559 6073 4020 3138 4752 6585
heap (insert 8241): 504 2698 1559 6073 4020 3138 4752 6585 8241
heap (insert 4747): 504 2698 1559 6073 4020 3138 4752 6585 8241 4747
heap (insert 6512): 504 2698 1559 6073 4020 3138 4752 6585 8241 4747 6512
heap (insert 4361): 504 2698 1559 6073 4020 3138 4752 6585 8241 4747 6512 4361
heap (insert 5817): 504 2698 1559 6073 4020 3138 4752 6585 8241 4747 6512 4361 5817
heap (insert 9333): 504 2698 1559 6073 4020 3138 4752 6585 8241 4747 6512 4361 5817 9333
heap (insert 4500): 504 2698 1559 6073 4020 3138 4500 6585 8241 4747 6512 4361 5817 9333 4752
heap (insert 5664): 504 2698 1559 5664 4020 3138 4500 6073 8241 4747 6512 4361 5817 9333 4752 6585
heap (insert 4275): 504 2698 1559 4275 4020 3138 4500 5664 8241 4747 6512 4361 5817 9333 4752 6585 6073
heap (insert 7735): 504 2698 1559 4275 4020 3138 4500 5664 7735 4747 6512 4361 5817 9333 4752 6585 6073 8241
heap (insert 4872): 504 2698 1559 4275 4020 3138 4500 5664 4872 4747 6512 4361 5817 9333 4752 6585 6073 8241 7735
heap (insert 3515): 504 2698 1559 4275 3515 3138 4500 5664 4872 4020 6512 4361 5817 9333 4752 6585 6073 8241 7735 4747
heap (extract 504): 1559 2698 3138 4275 3515 4361 4500 5664 4872 4020 6512 4747 5817 9333 4752 6585 6073 8241 7735
heap (extract 1559): 2698 3515 3138 4275 4020 4361 4500 5664 4872 7735 6512 4747 5817 9333 4752 6585 6073 8241
heap (extract 2698): 3138 3515 4361 4275 4020 4747 4500 5664 4872 7735 6512 8241 5817 9333 4752 6585 6073
heap (extract 3138): 3515 4020 4361 4275 6073 4747 4500 5664 4872 7735 6512 8241 5817 9333 4752 6585
heap (extract 3515): 4020 4275 4361 4872 6073 4747 4500 5664 6585 7735 6512 8241 5817 9333 4752
heap (extract 4020): 4275 4752 4361 4872 6073 4747 4500 5664 6585 7735 6512 8241 5817 9333
heap (extract 4275): 4361 4752 4500 4872 6073 4747 9333 5664 6585 7735 6512 8241 5817
heap (extract 4361): 4500 4752 4747 4872 6073 5817 9333 5664 6585 7735 6512 8241
heap (extract 4500): 4747 4752 5817 4872 6073 8241 9333 5664 6585 7735 6512
heap (extract 4747): 4752 4872 5817 5664 6073 8241 9333 6512 6585 7735
heap (extract 4752): 4872 5664 5817 6512 6073 8241 9333 7735 6585
heap (extract 4872): 5664 6073 5817 6512 6585 8241 9333 7735
heap (extract 5664): 5817 6073 7735 6512 6585 8241 9333
heap (extract 5817): 6073 6512 7735 9333 6585 8241
heap (extract 6073): 6512 6585 7735 9333 8241
heap (extract 6512): 6585 8241 7735 9333
heap (extract 6585): 7735 8241 9333
heap (extract 7735): 8241 9333
heap (extract 8241): 9333
heap (extract 9333):
sorted: 504 1559 2698 3138 3515 4020 4275 4361 4500 4747 4752 4872 5664 5817 6073 6512 6585 7735 8241 9333
```

With heap.js implementation

Heapsort Tutorial JavaScript

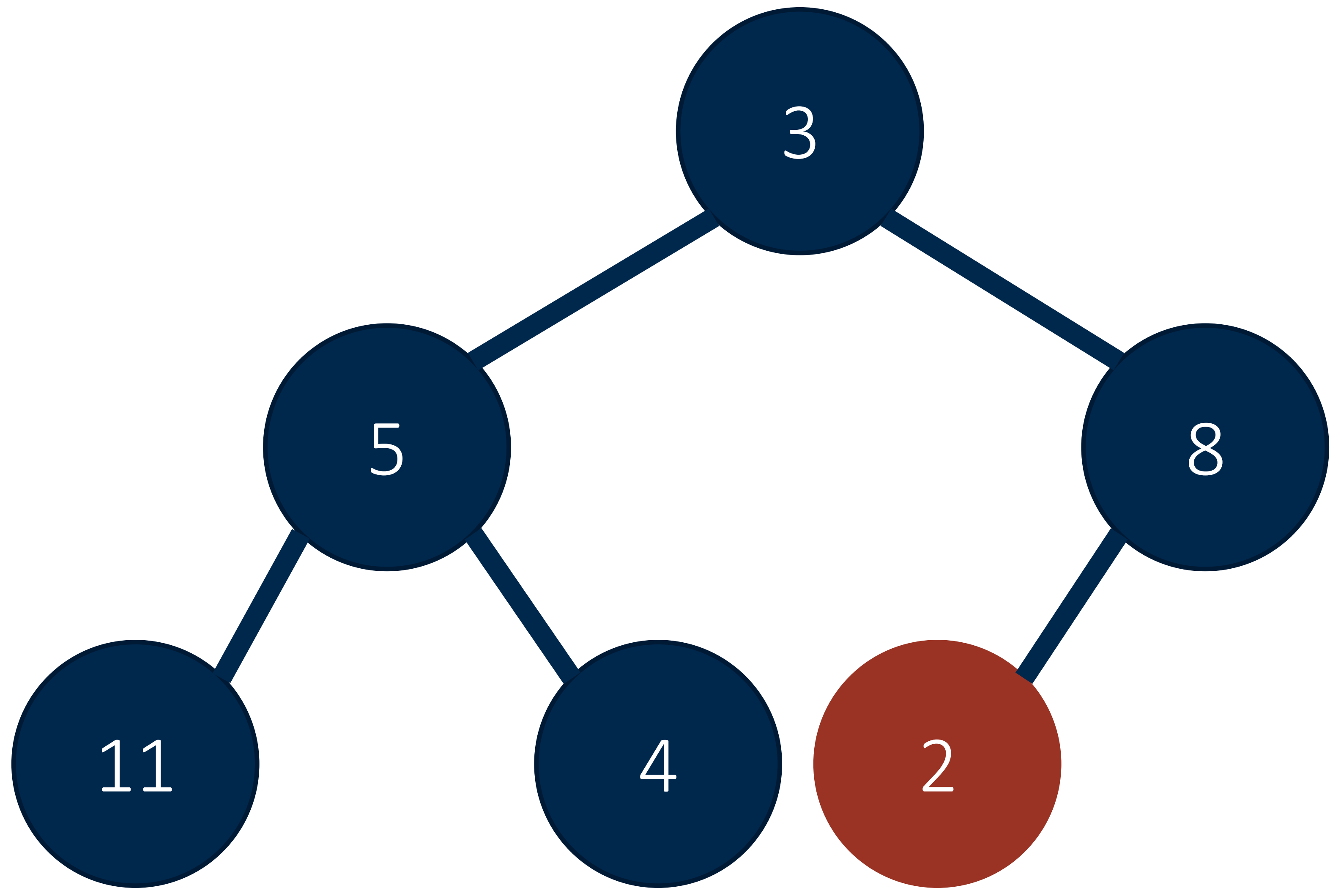


heap.js

```
// 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
}
```

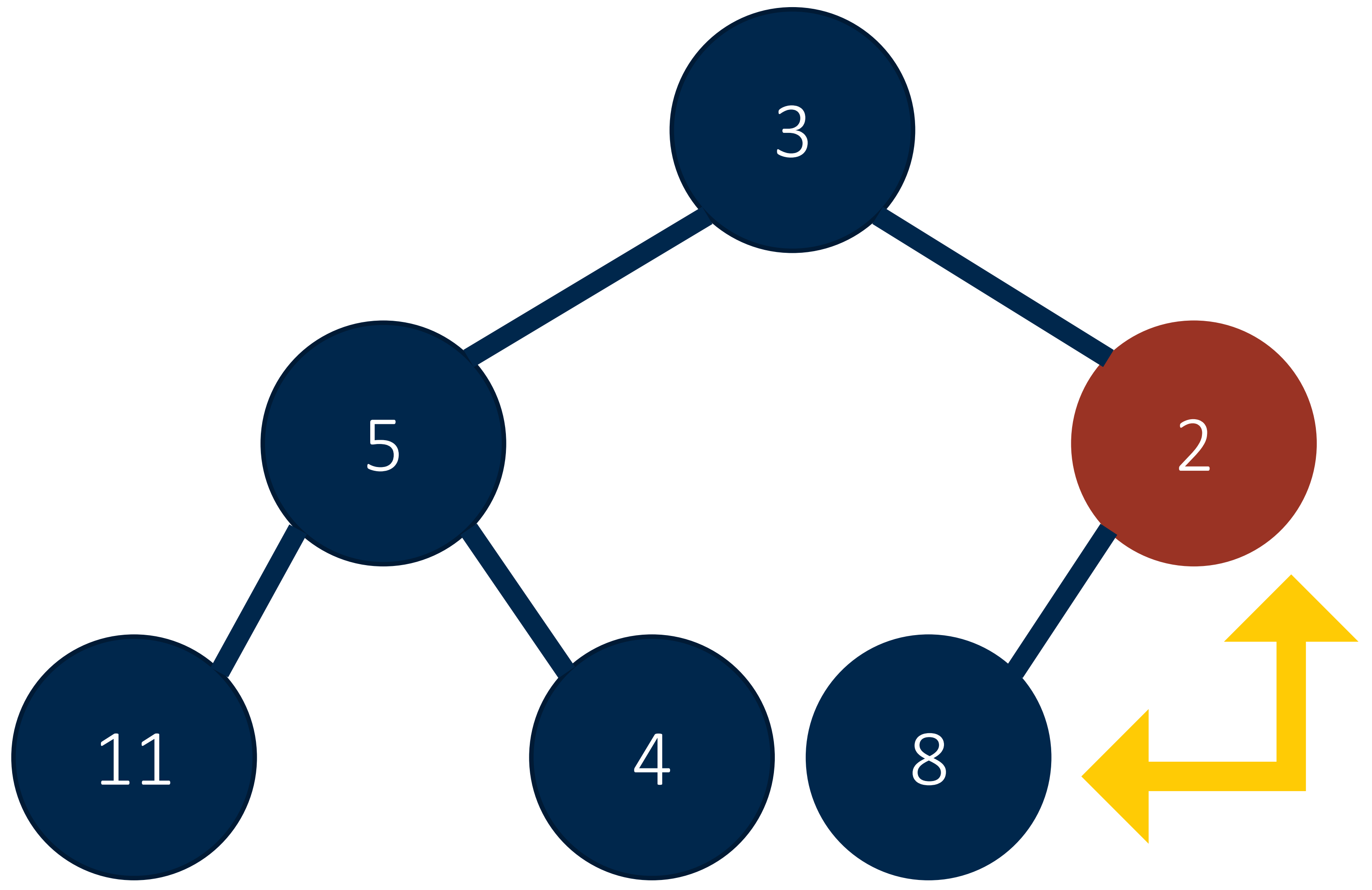
Heap Insert

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



Heap Insert

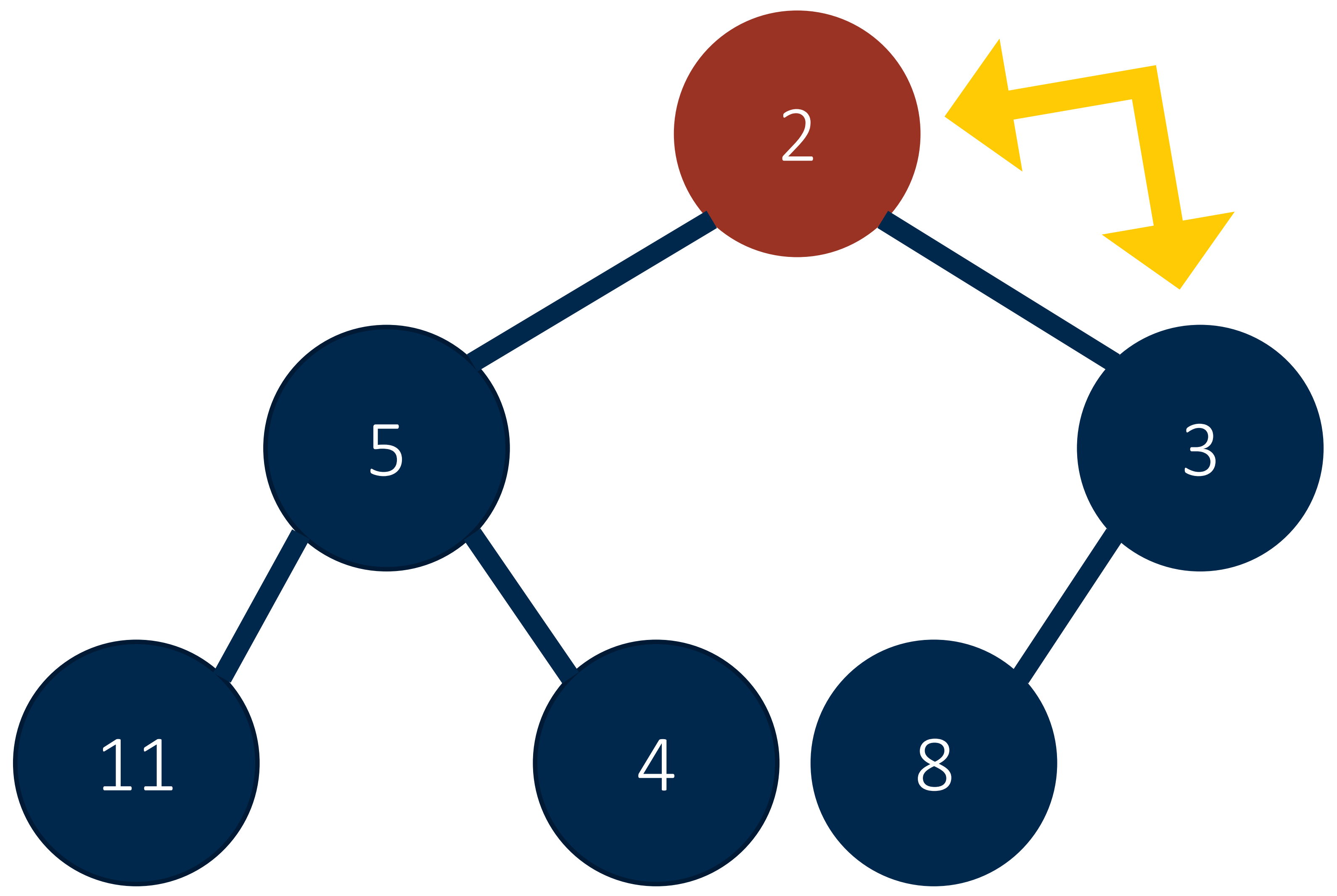
1. Add new element into first open spot in tree (end of heap)
2. If new element is smaller than parent, swap with parent



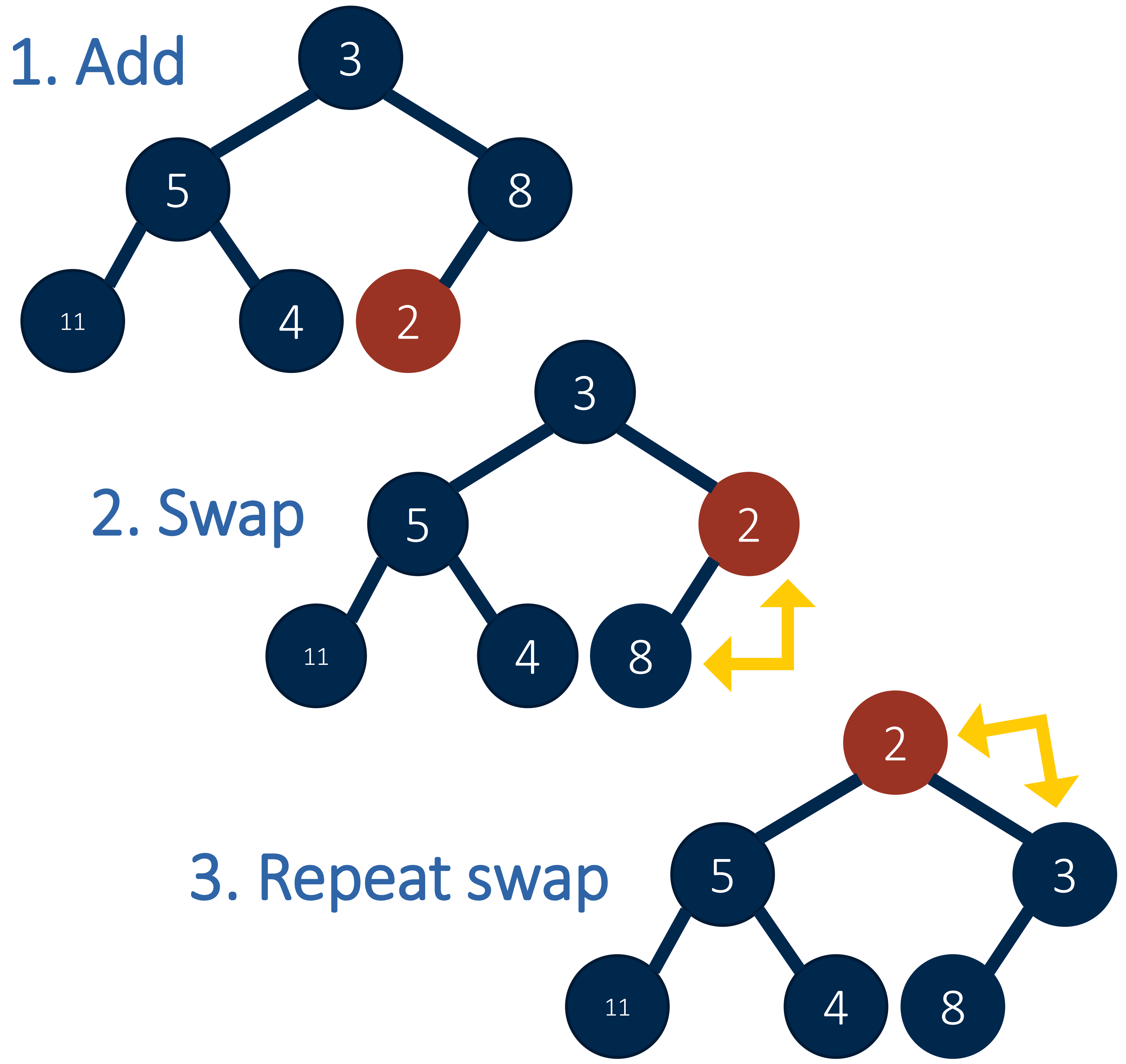
Heap Insert

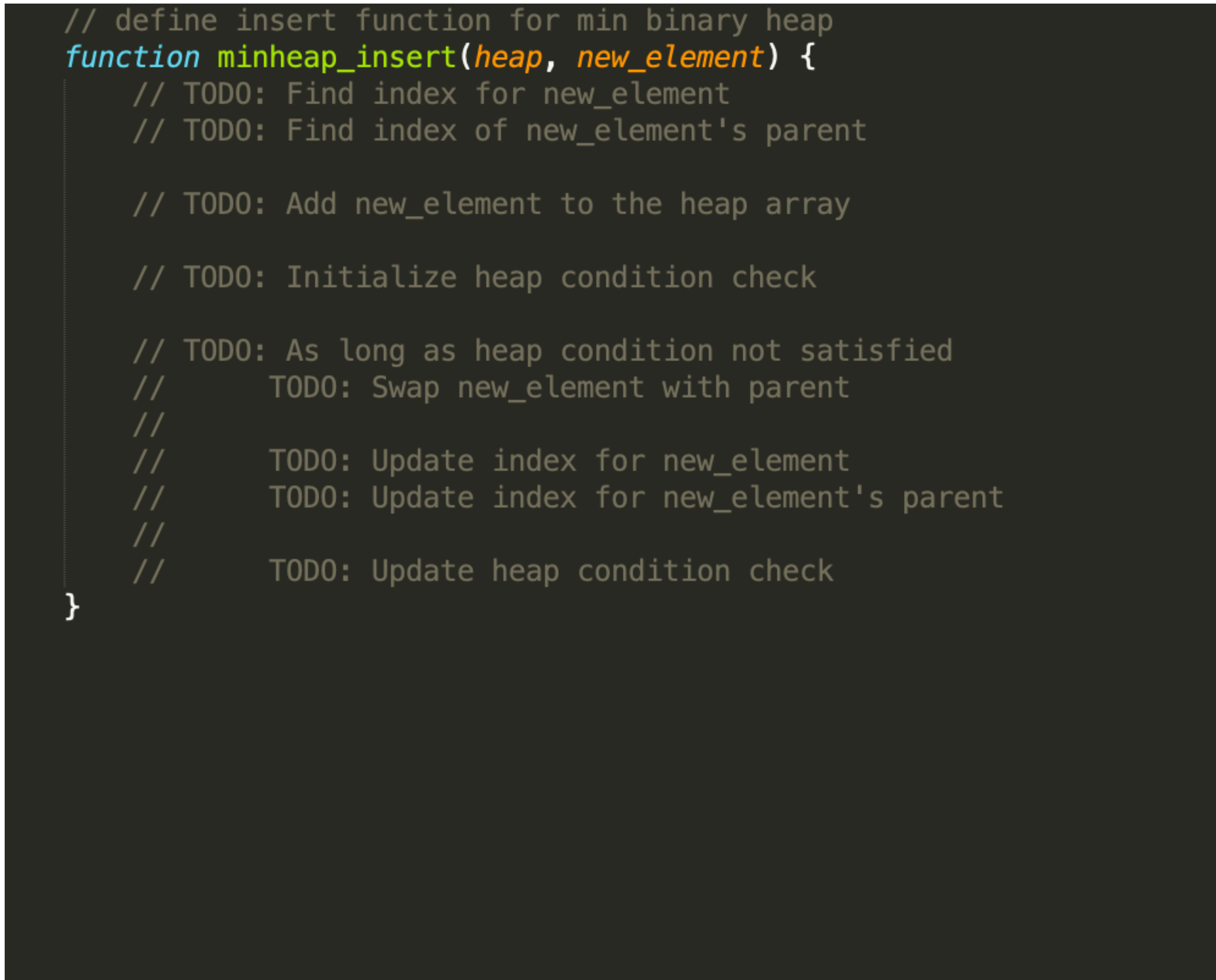
1. Add new element into first open spot in tree (end of heap)
2. If new element is smaller than parent, swap with parent
3. Repeat step 2 until heap property holds

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





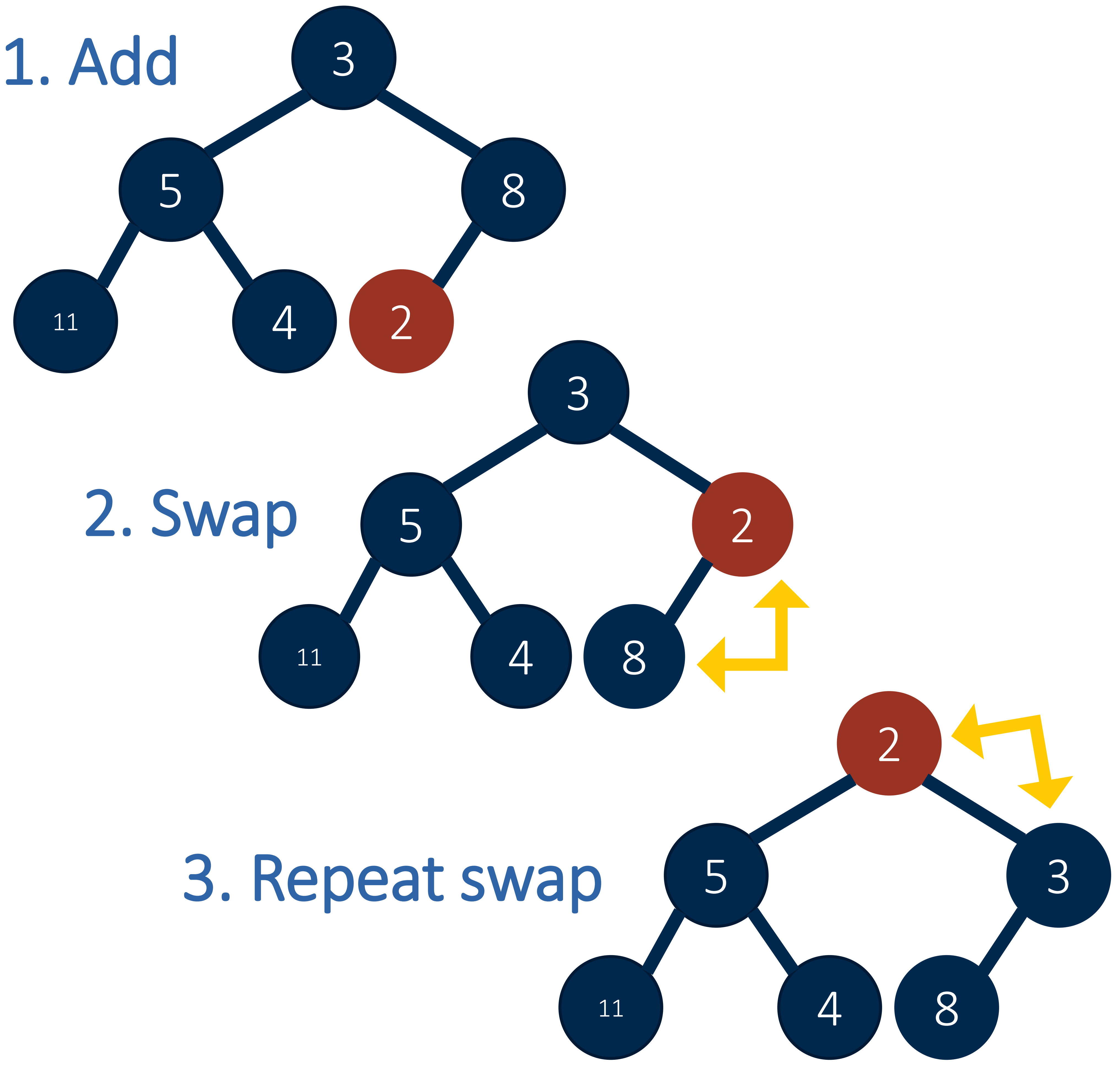
```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    // TODO: Find index for new_element
    // TODO: Find index of new_element's parent

    // 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
}
```

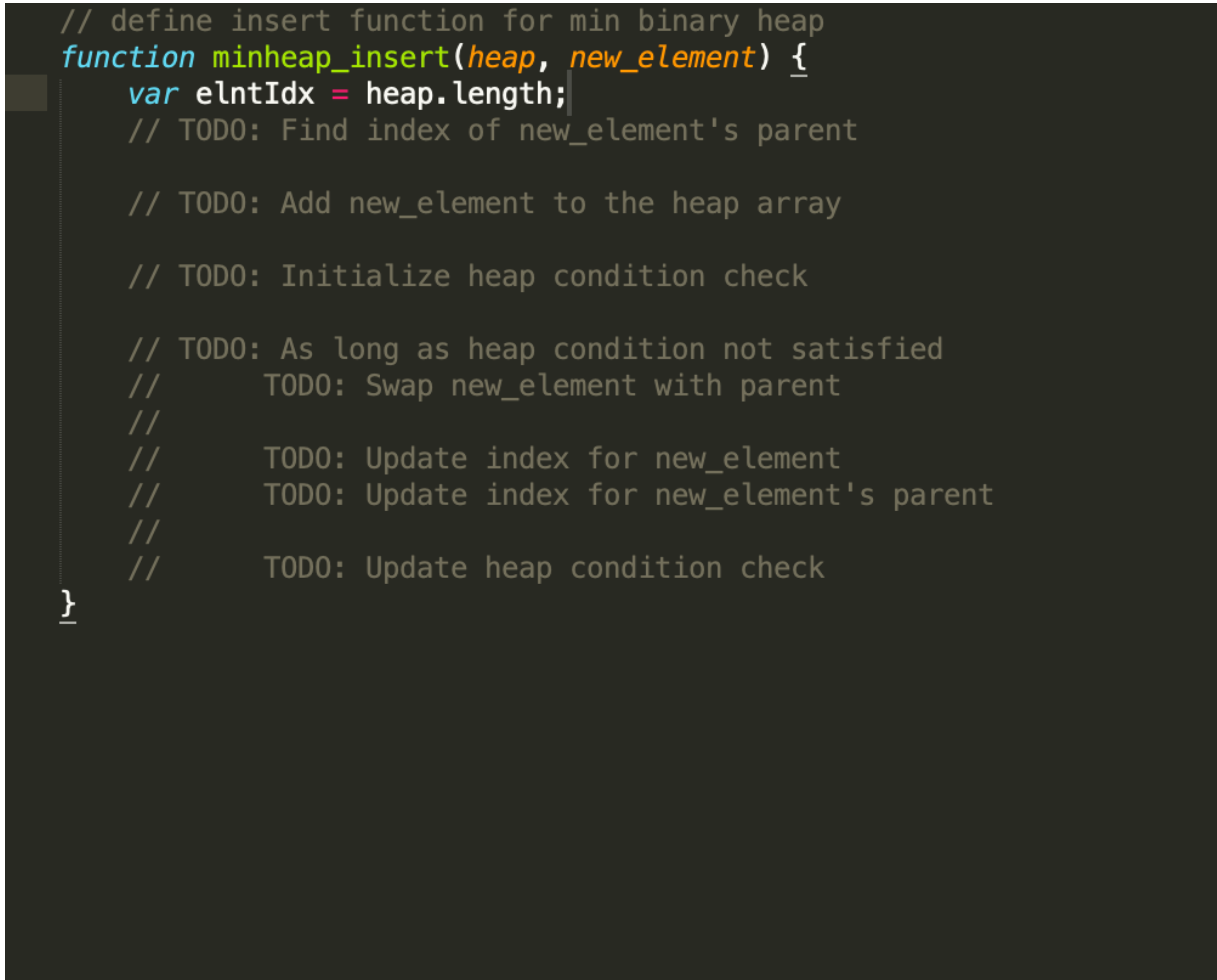
Heap Insert

1. Add

2. Swap

3. Repeat swap

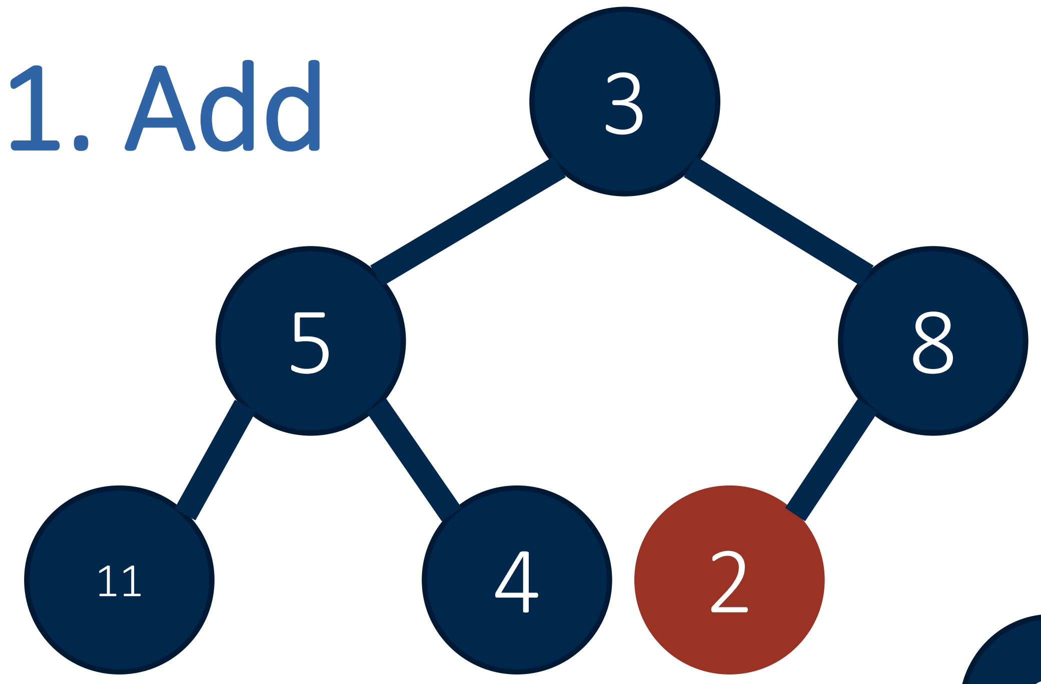




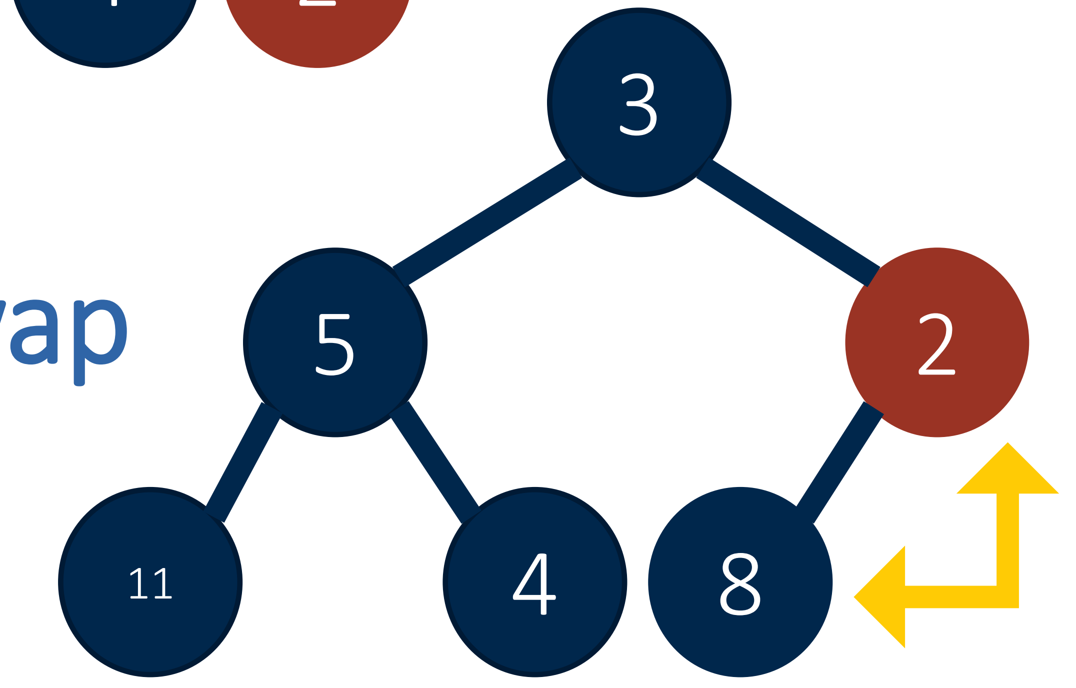
```javascript
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    // TODO: Find index of new_element's parent

    // 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

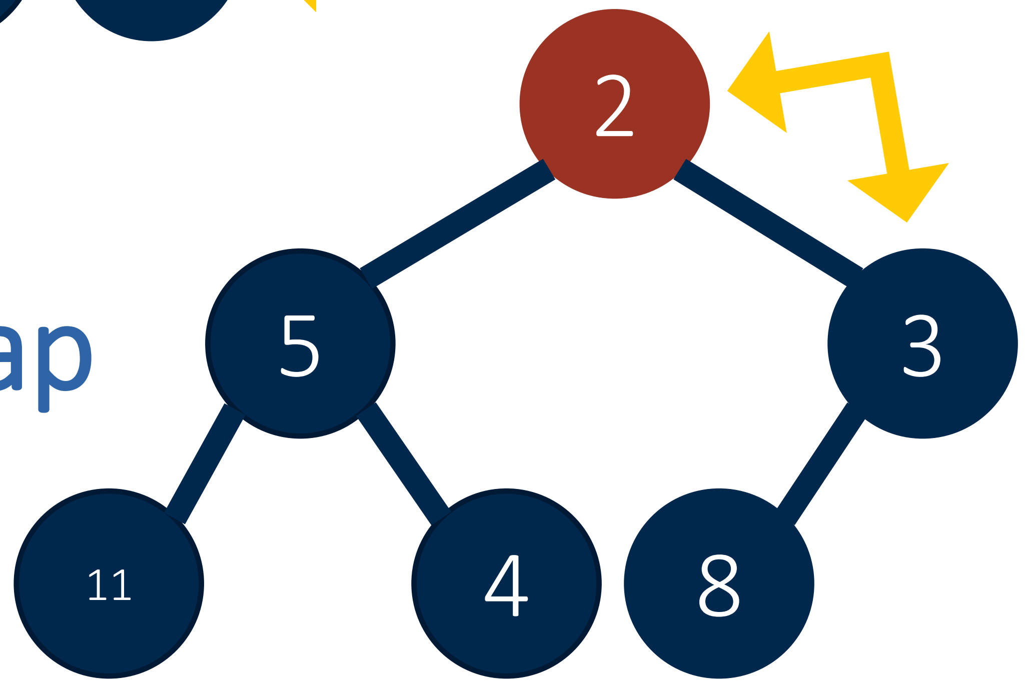
1. Add

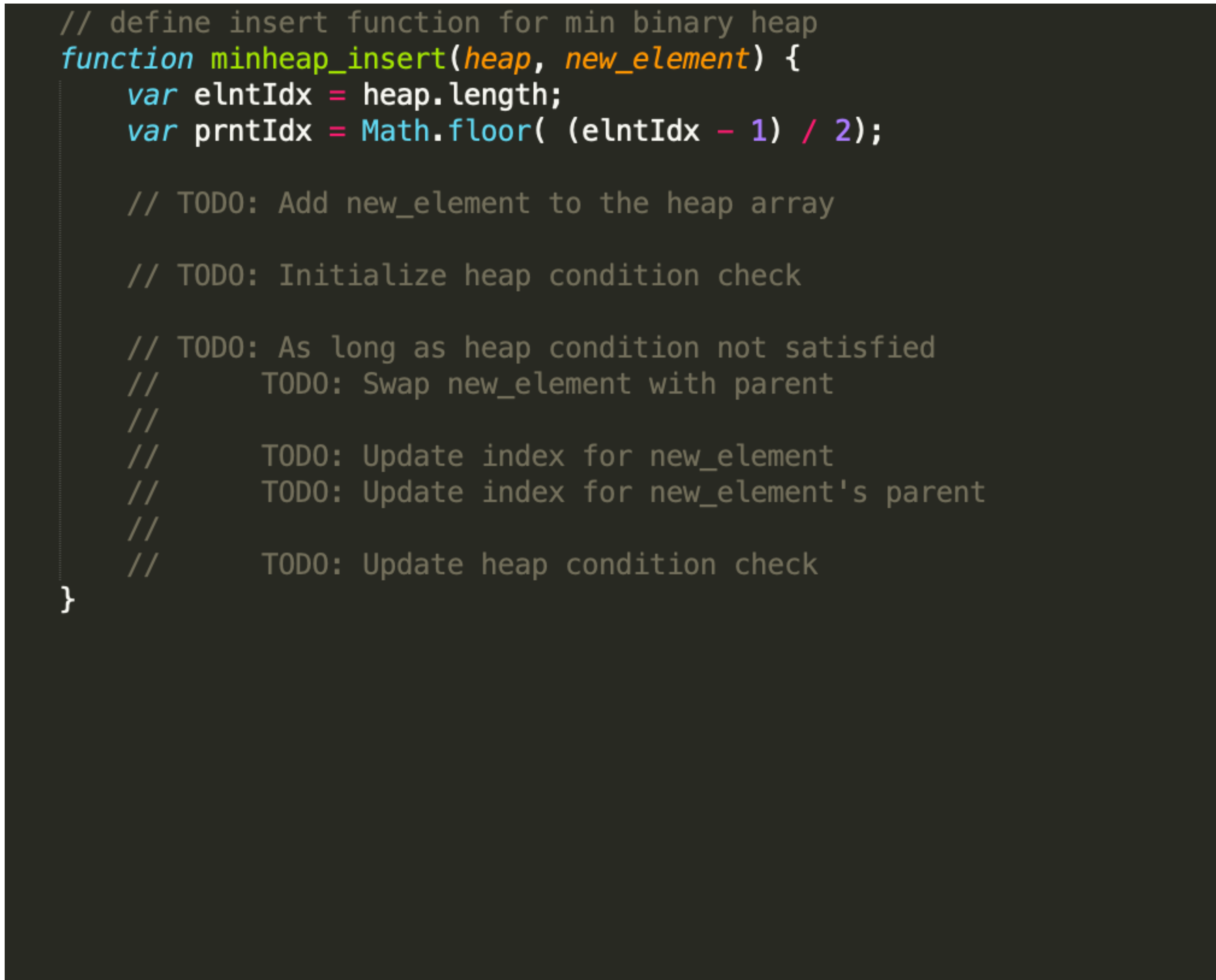


2. Swap



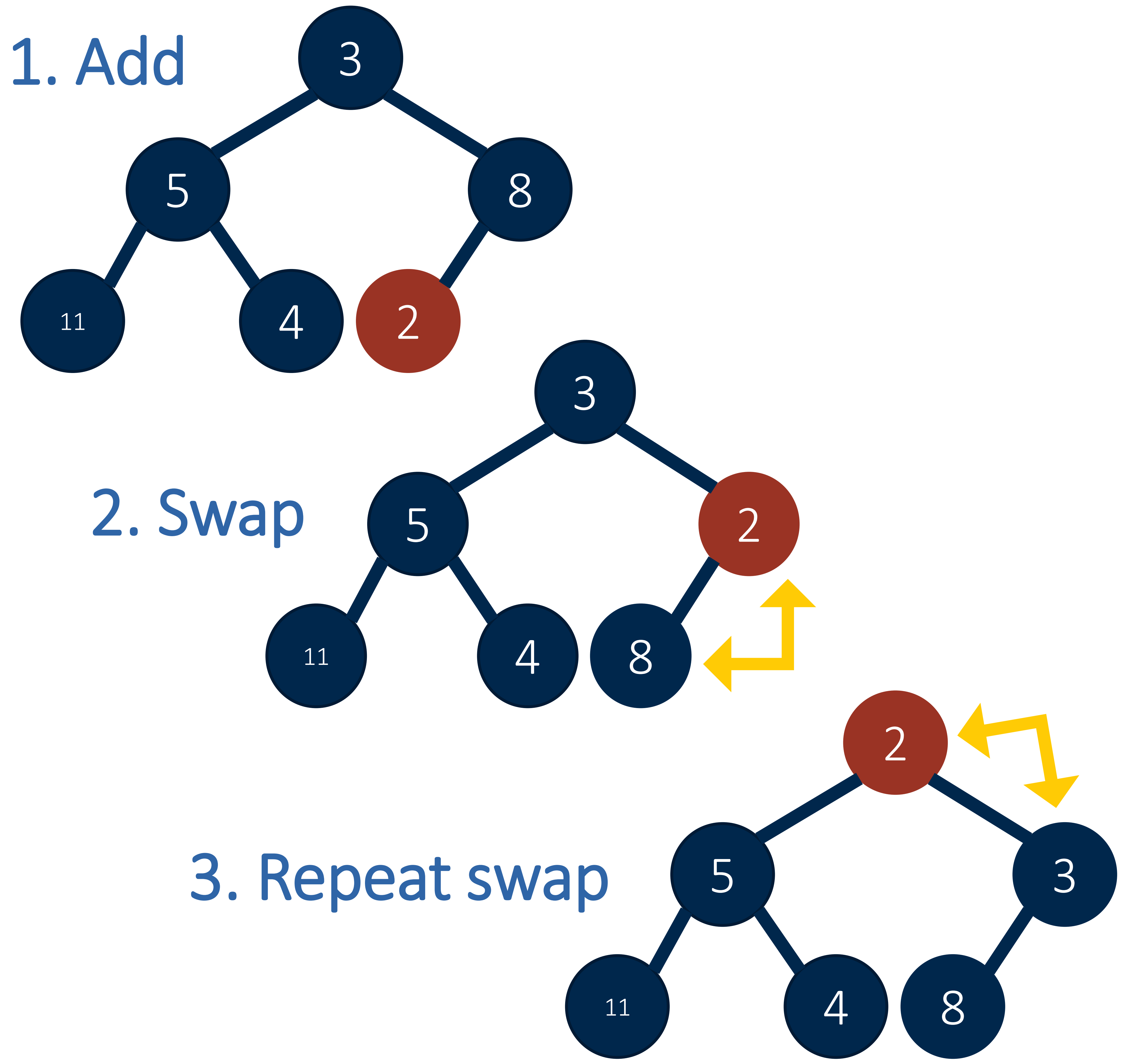
3. Repeat swap

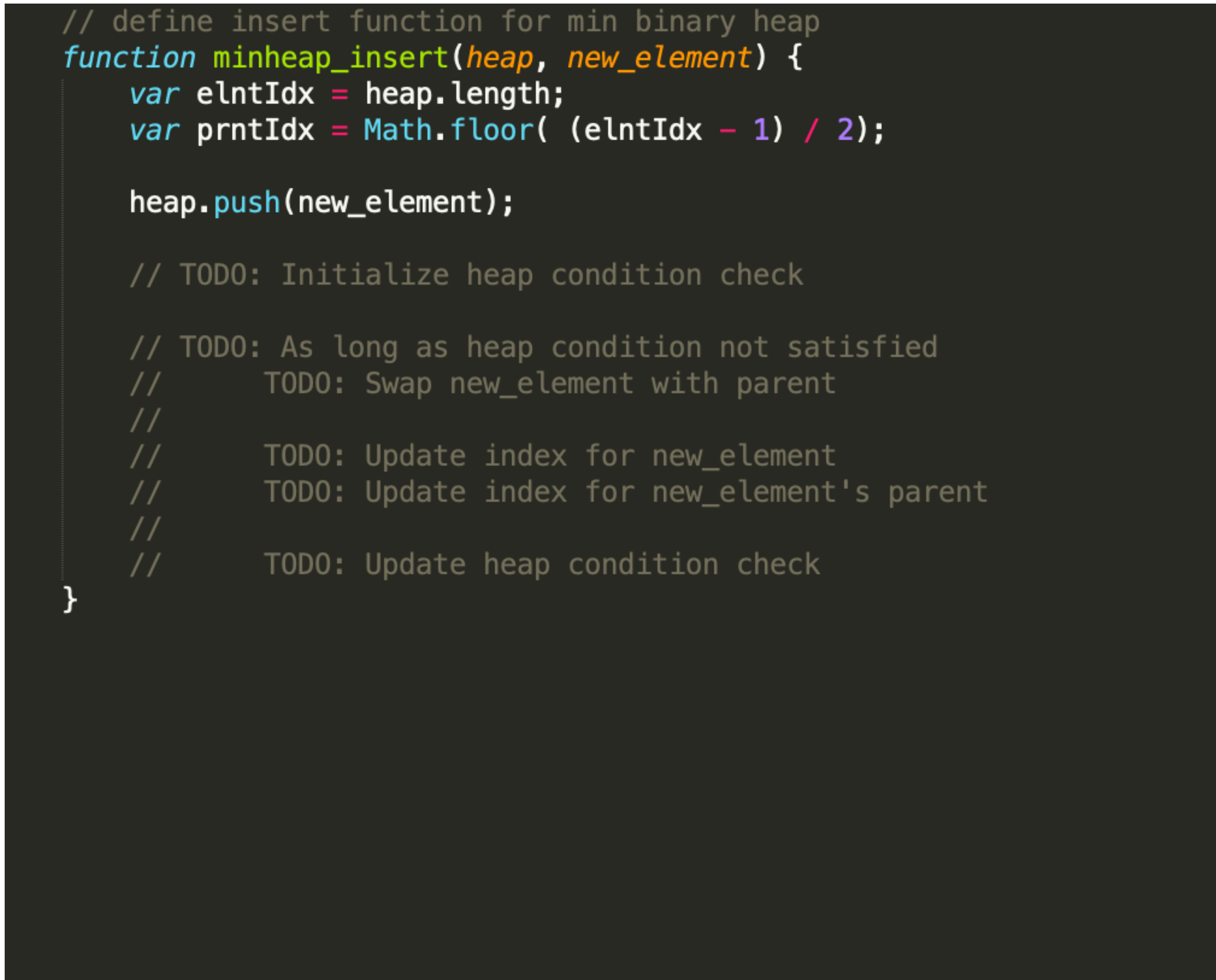




```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    // 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





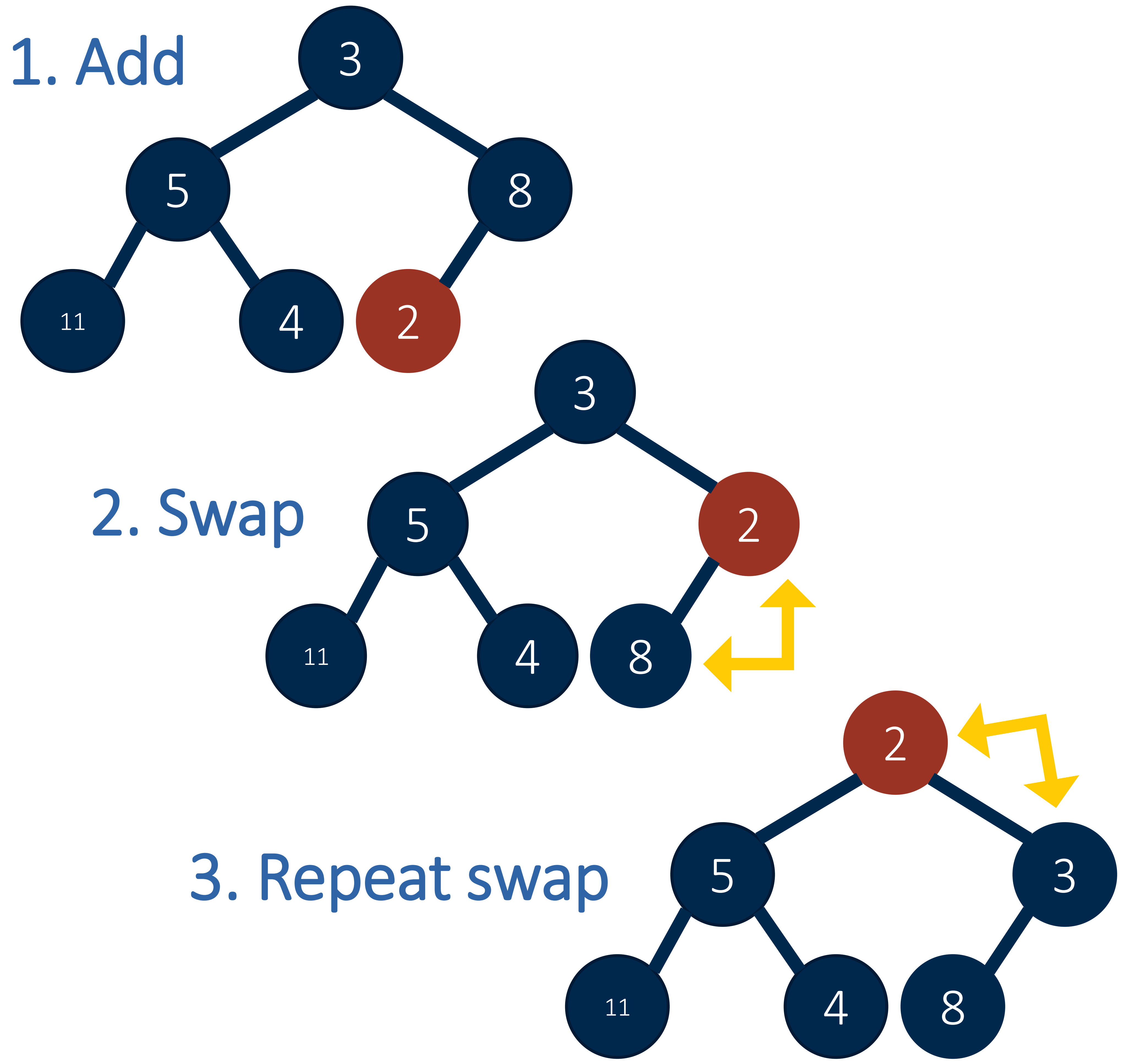
```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    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
}
```

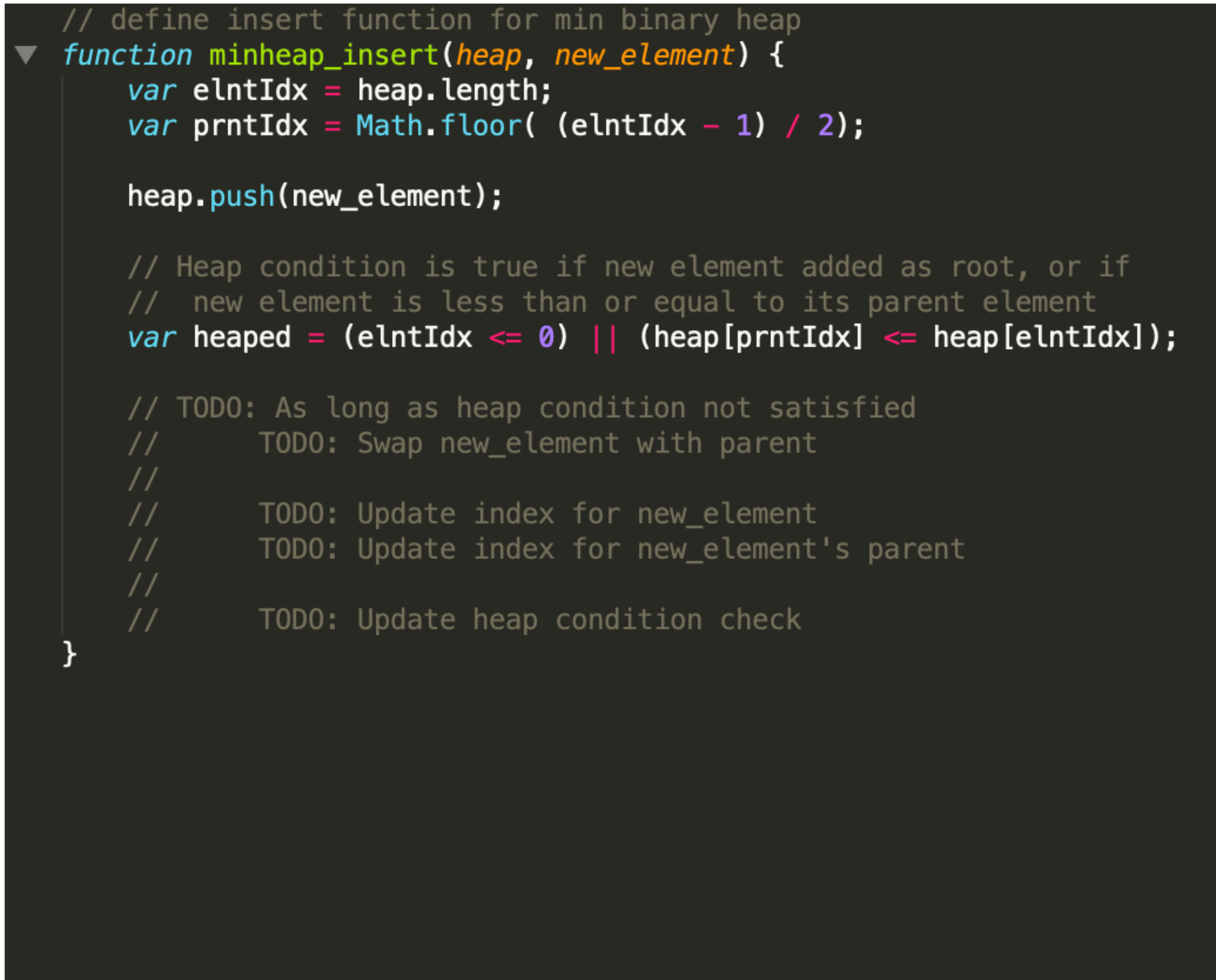
Heap Insert

1. Add

2. Swap

3. Repeat swap





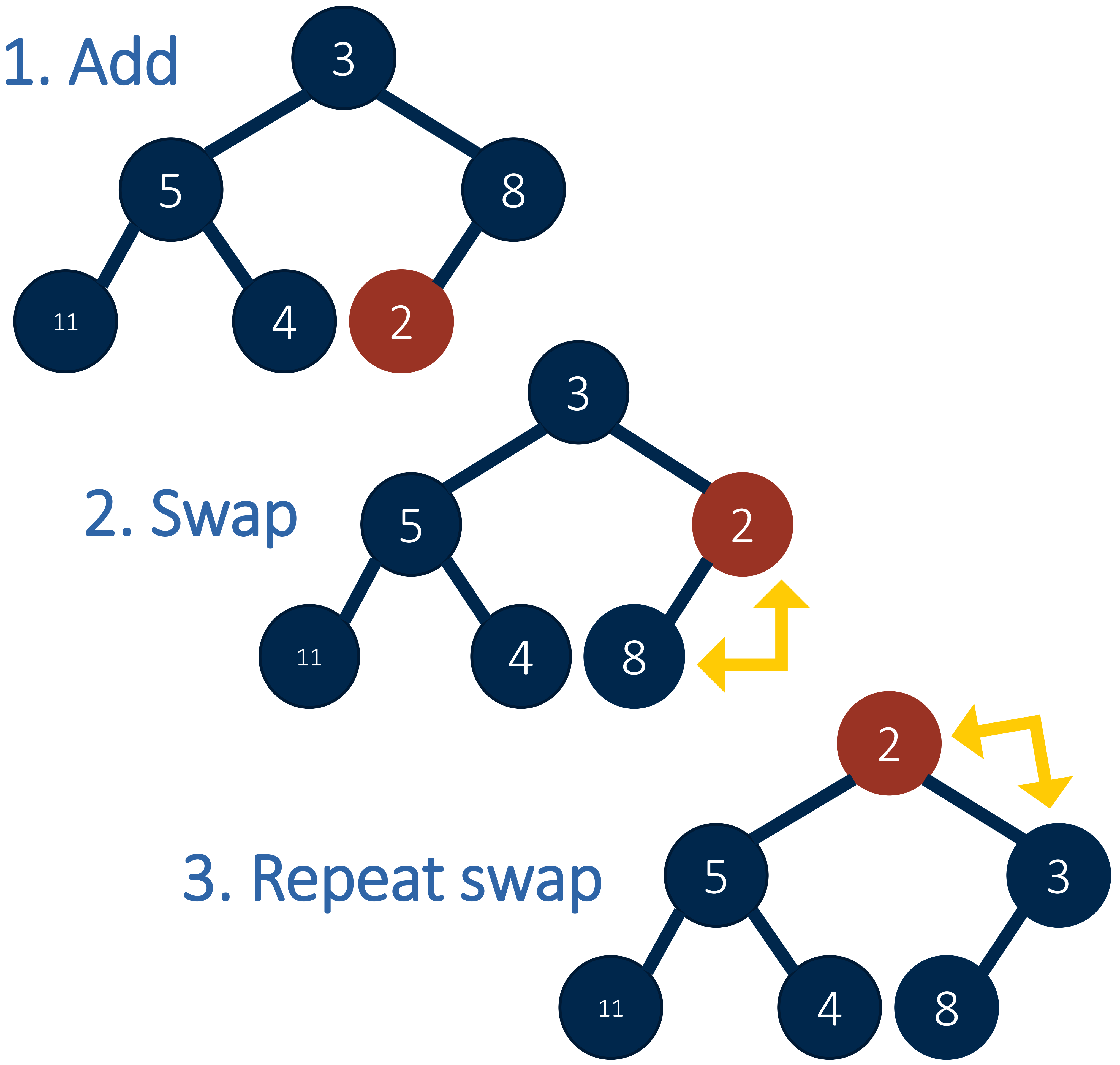
```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    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
}
```

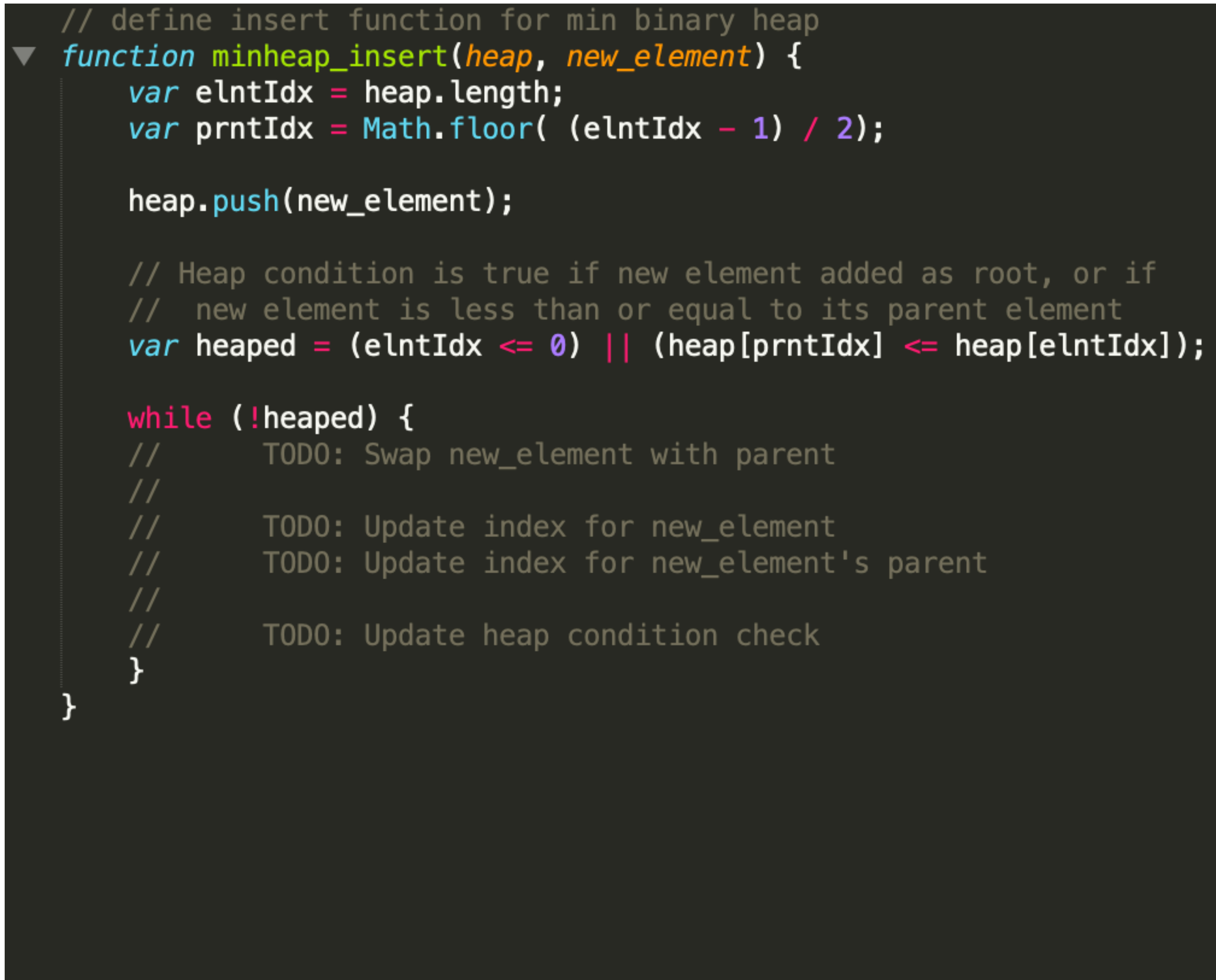
Heap Insert

1. Add

2. Swap

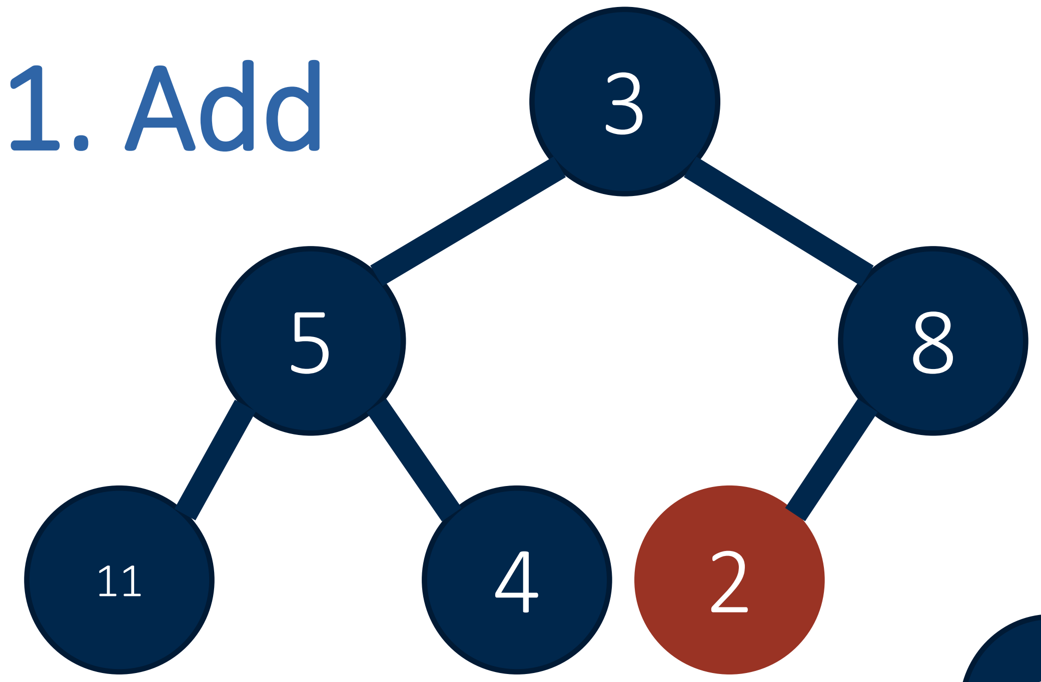
3. Repeat swap



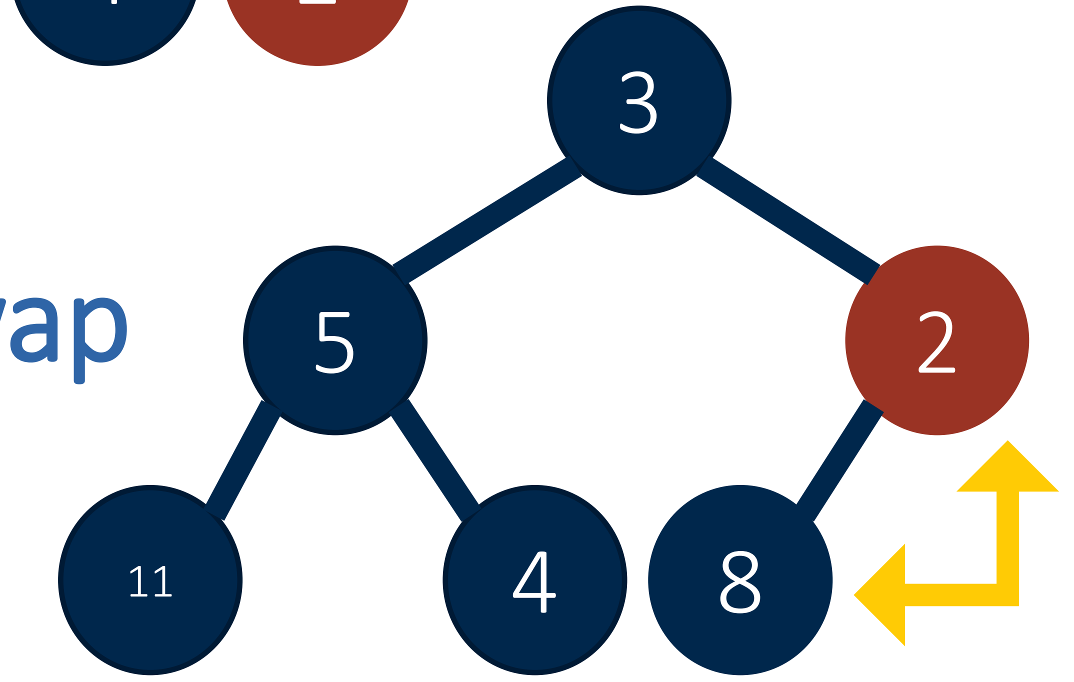

```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    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

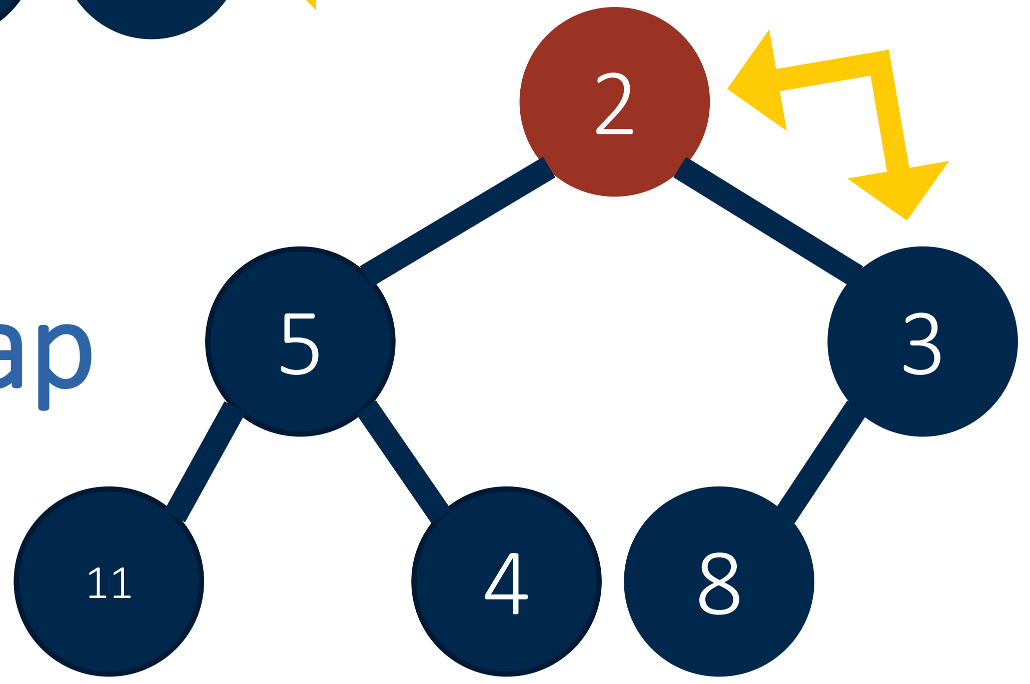
1. Add

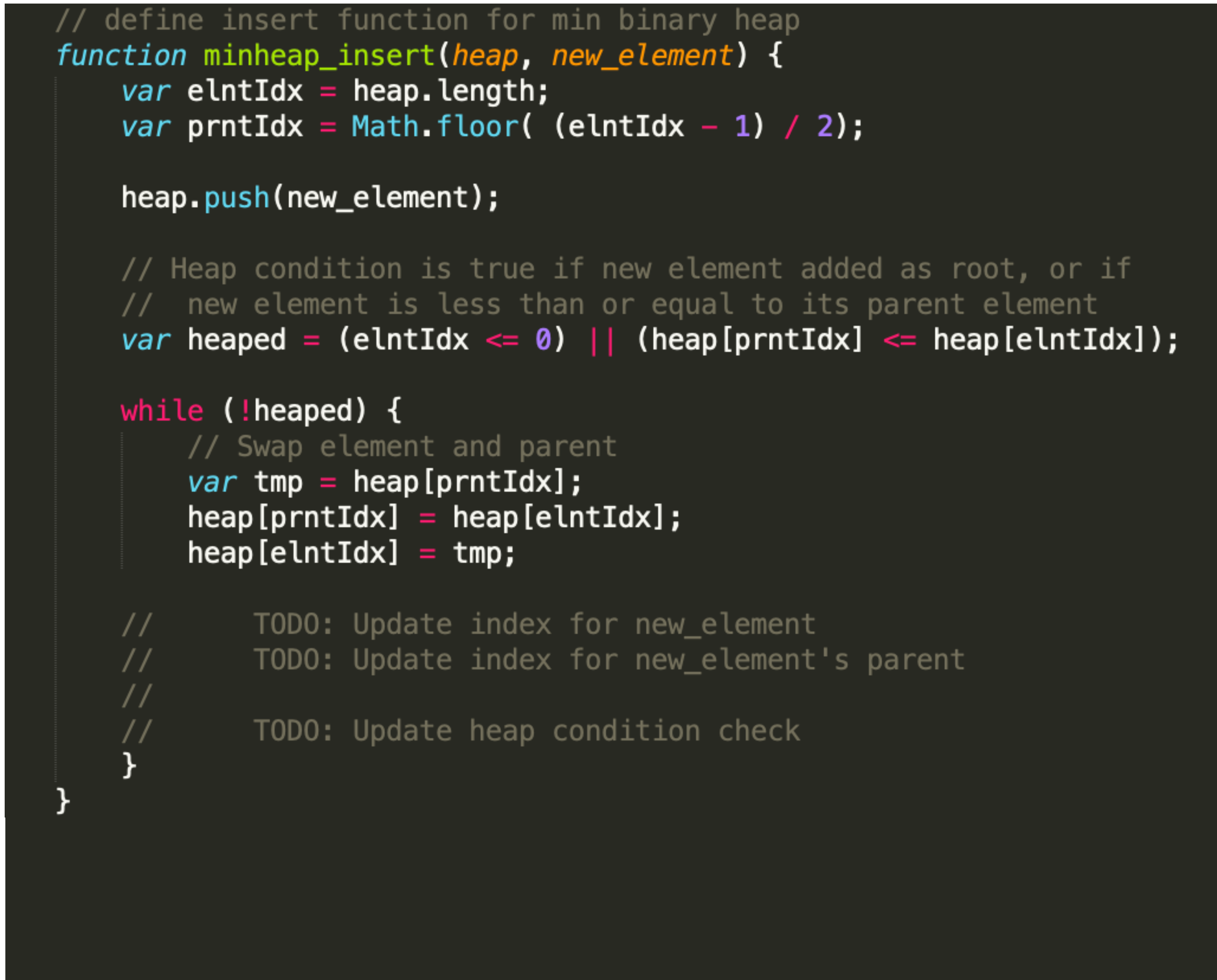


2. Swap



3. Repeat swap





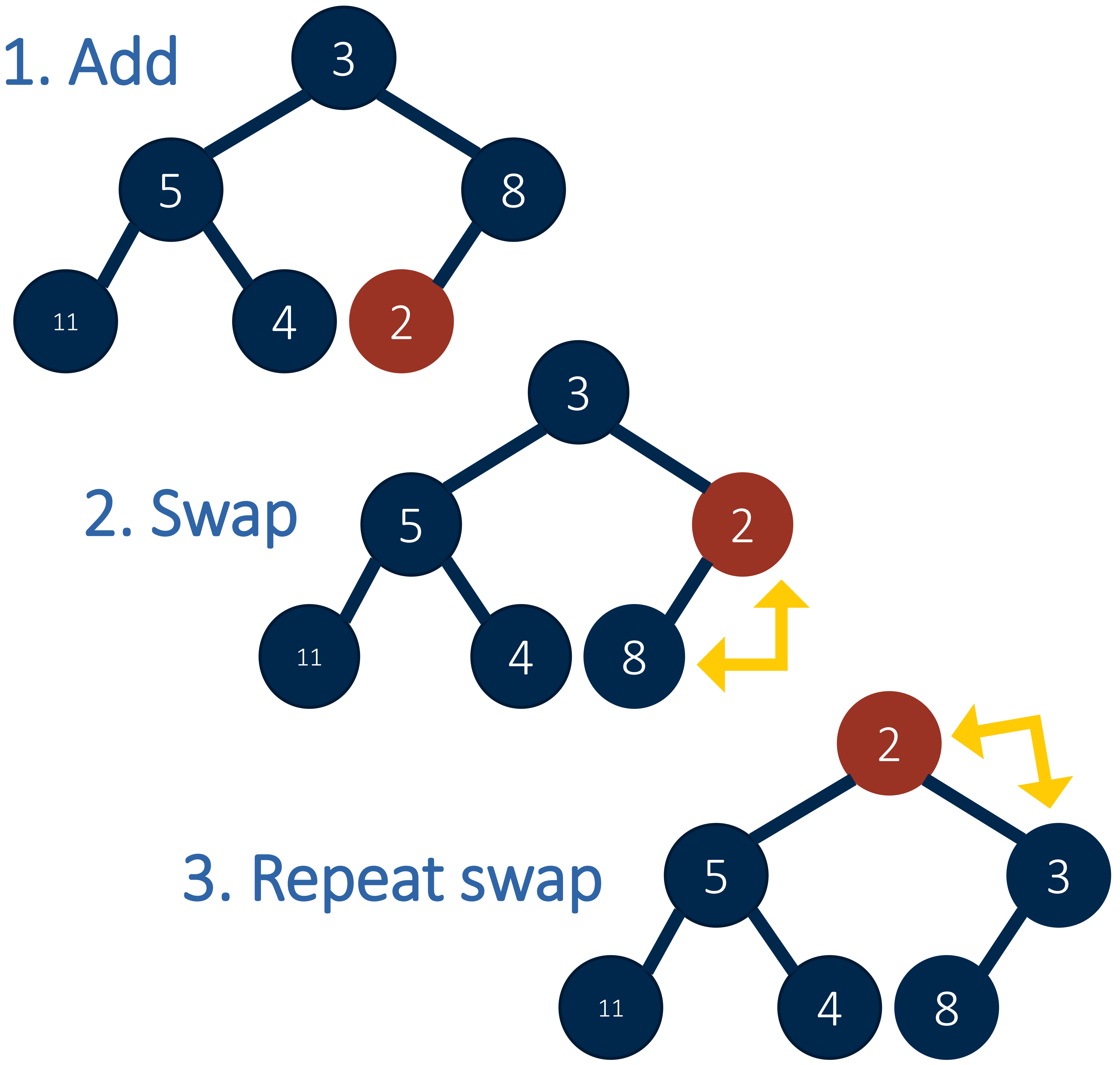
```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    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
    }
}
```

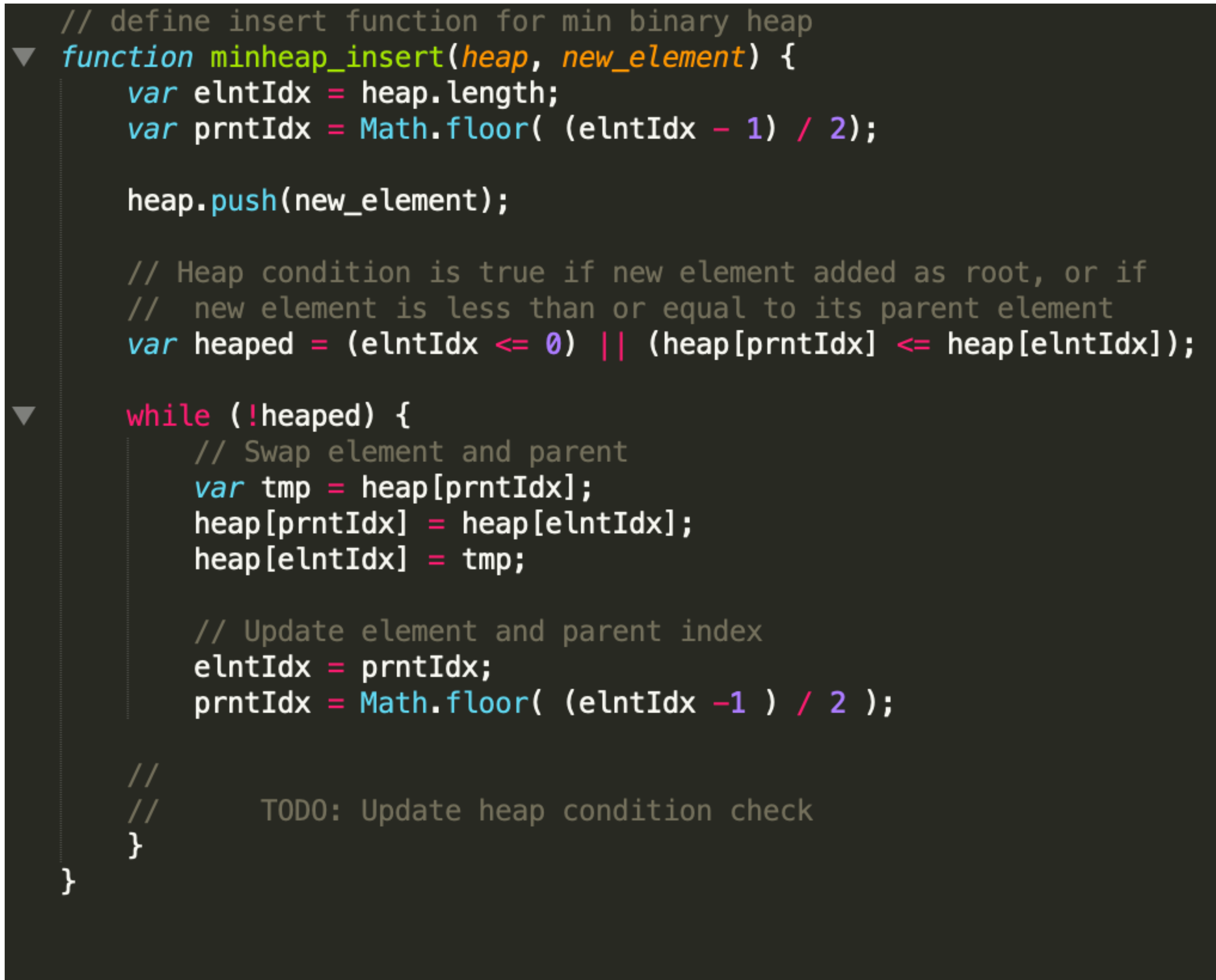
Heap Insert

1. Add

2. Swap

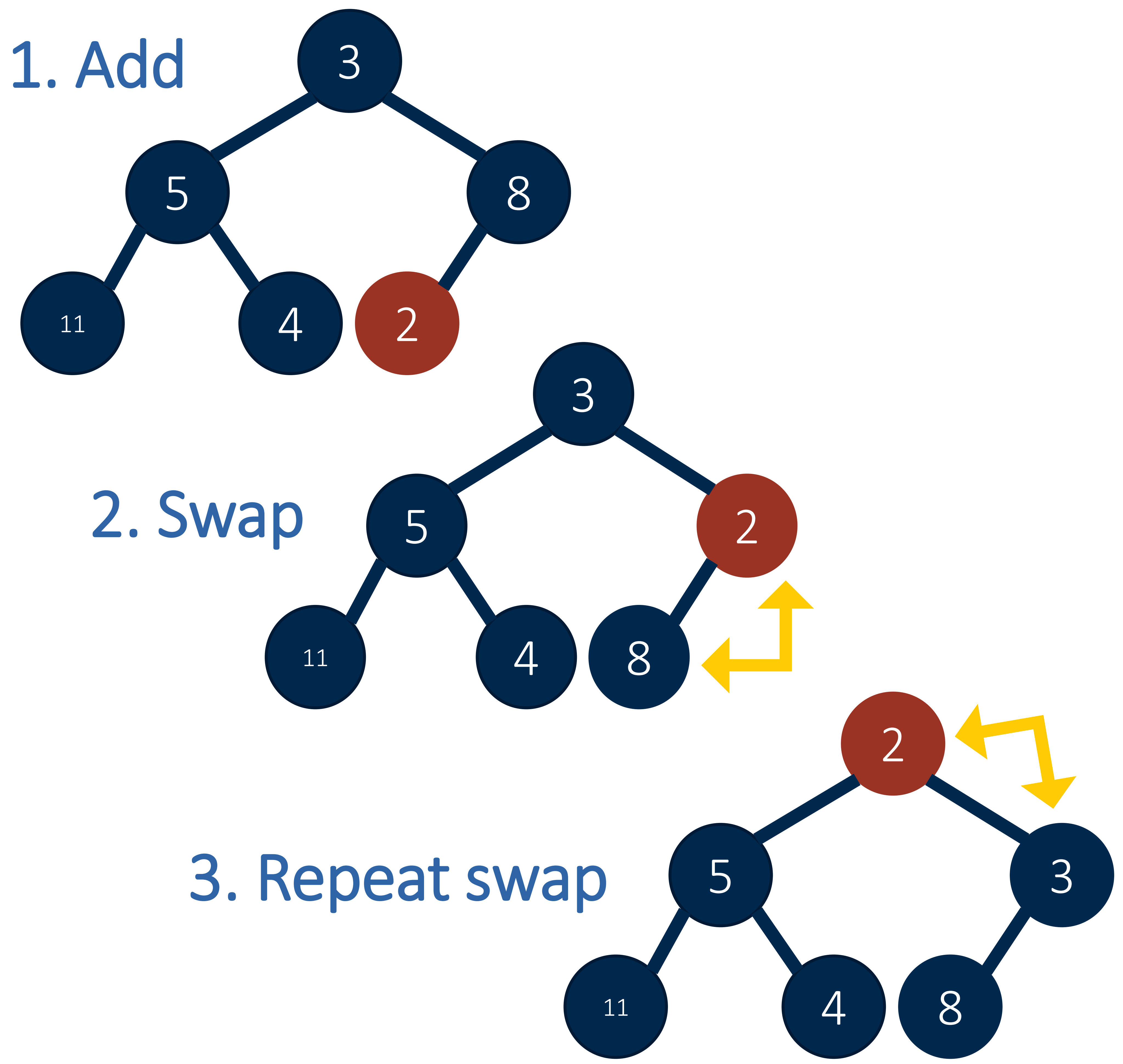
3. Repeat swap

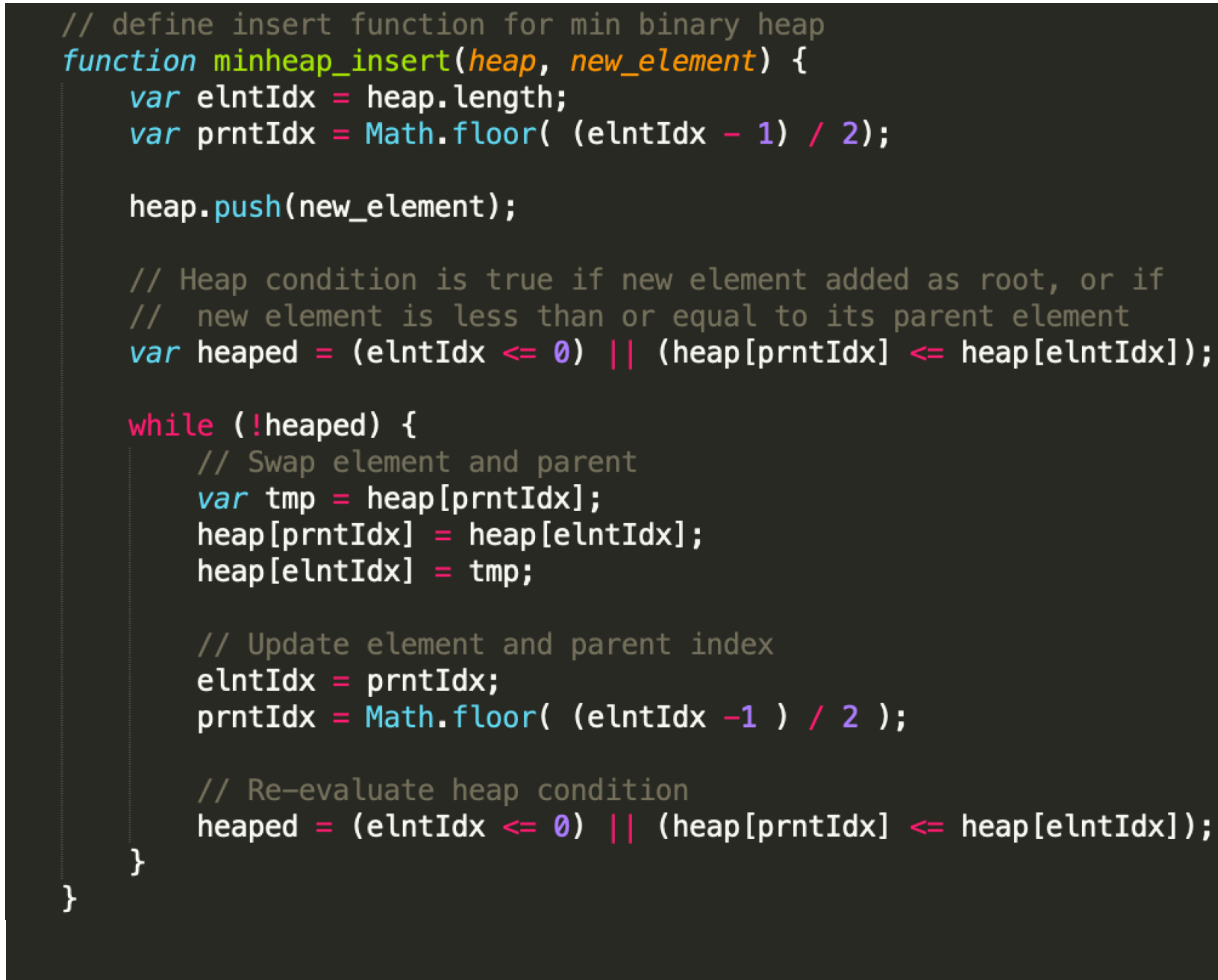




```
// define insert function for min binary heap
function minheap_insert(heap, new_element) {
    var elntIdx = heap.length;
    var prntIdx = Math.floor( (elntIdx - 1) / 2);

    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
    }
}
```

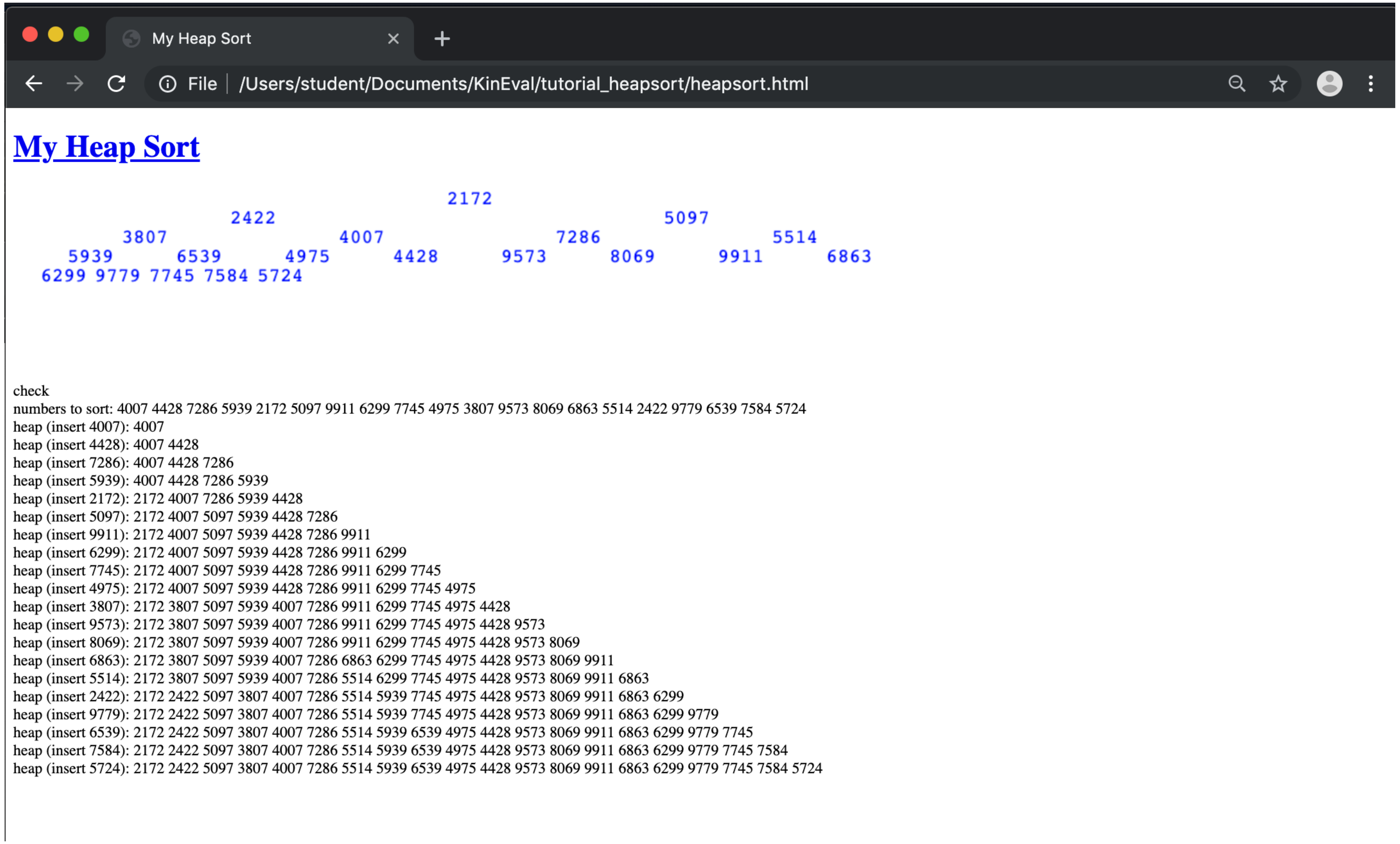
Heap Insert





```
// 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

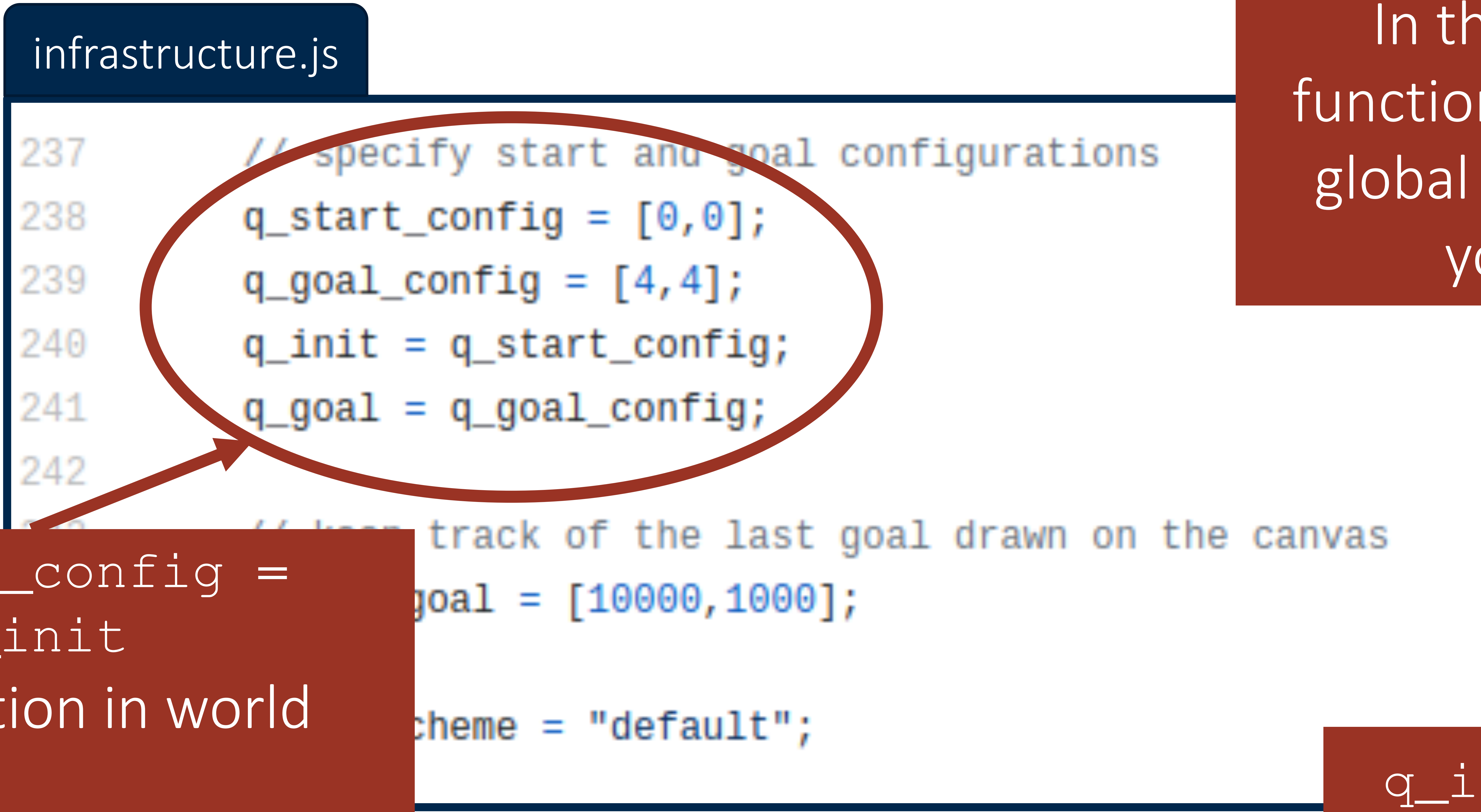
```
                                        2172
                    2422                                        5097
          3807                4007                7286                5514
     5939      6539      4975      4428      9573      8069      9911      6863
  6299 9779 7745 7584 5724
```

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;

        //       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

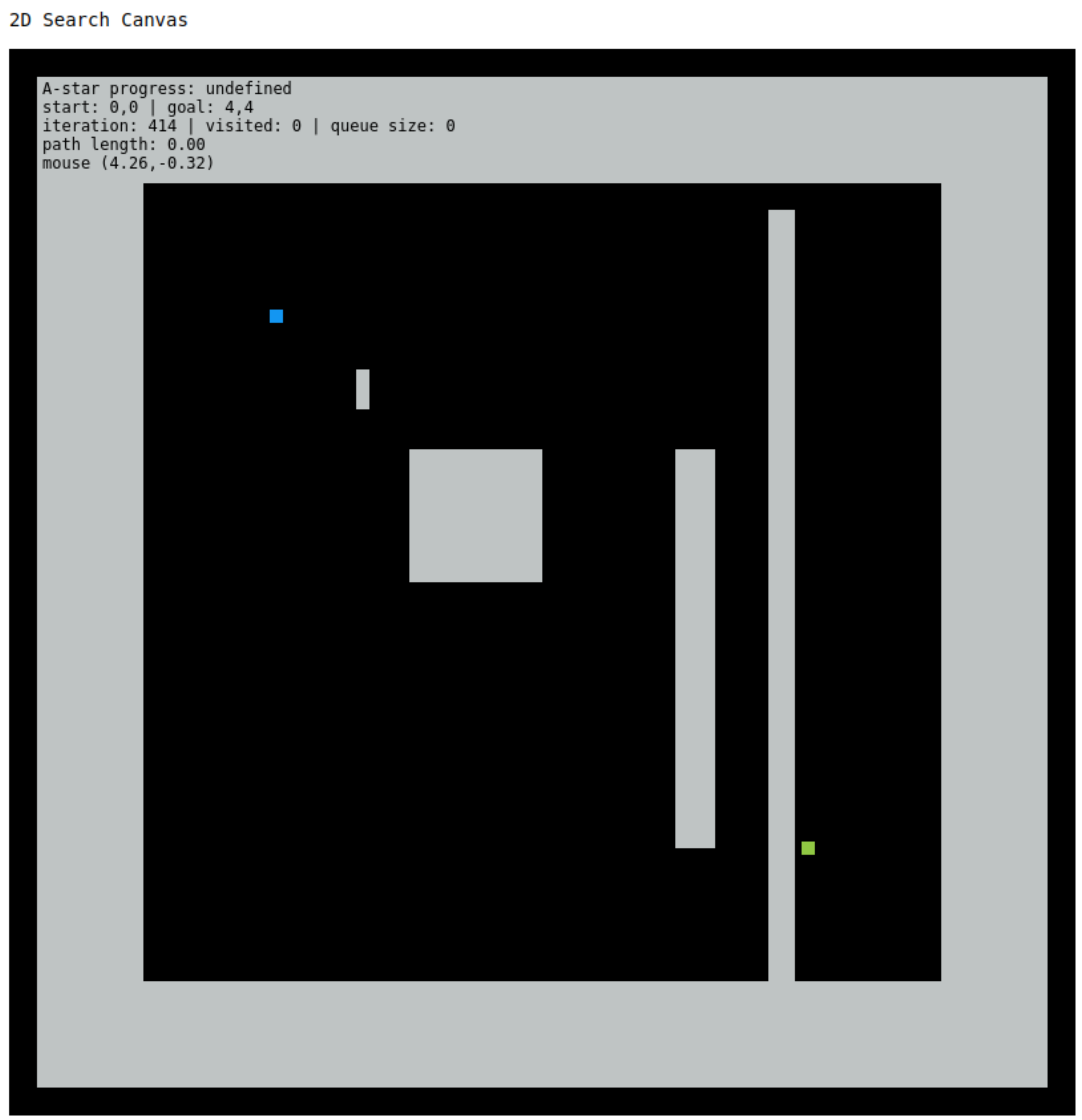
`q_init` and `q_goal` can be specified in URL

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)



2D Search Canvas

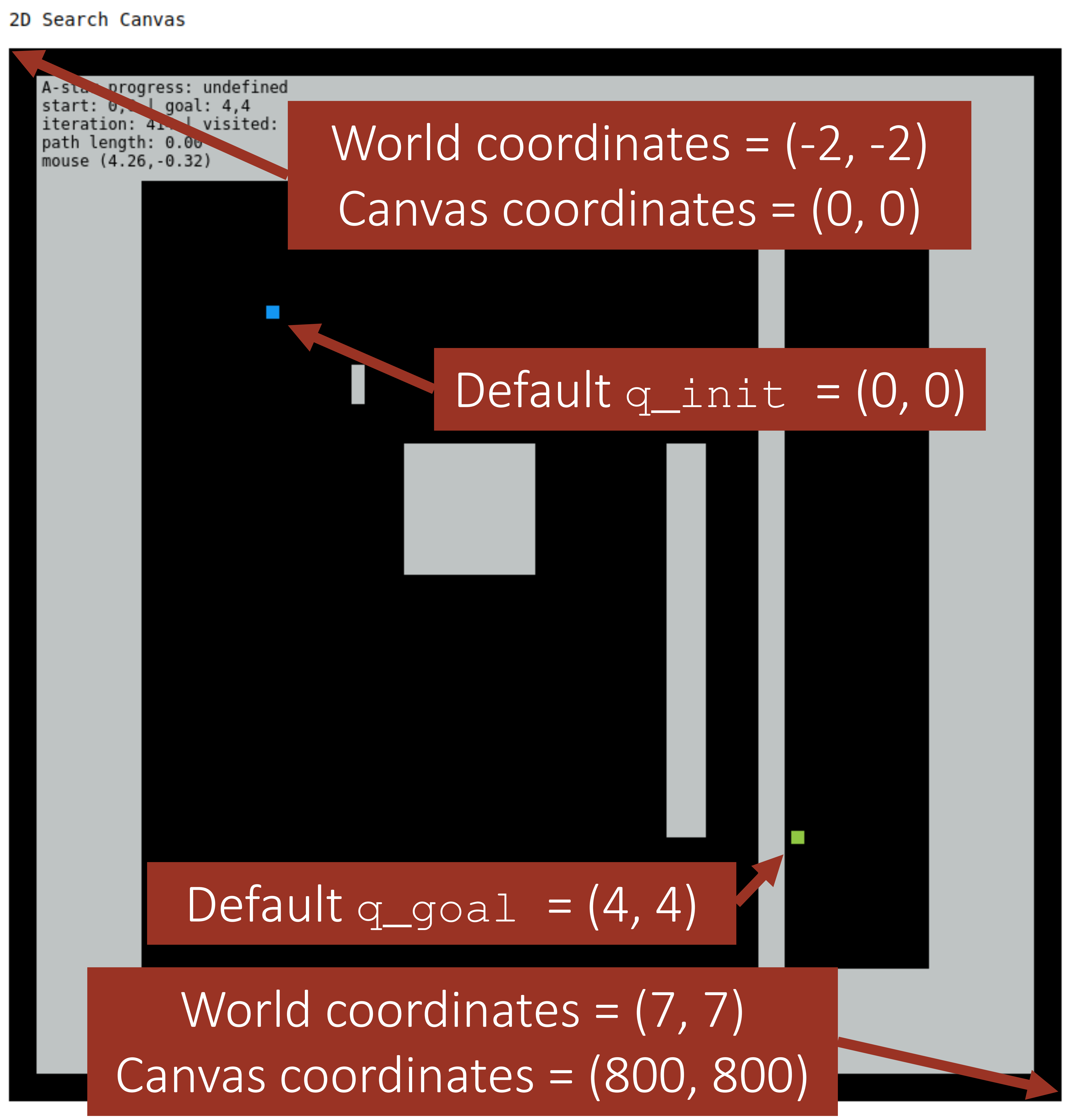
```
A-star progress: undefined
start: 0,0 | goal: 4,4
iteration: 414 | visited: 0 | queue size: 0
path length: 0.00
mouse (4.26,-0.32)
```

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)

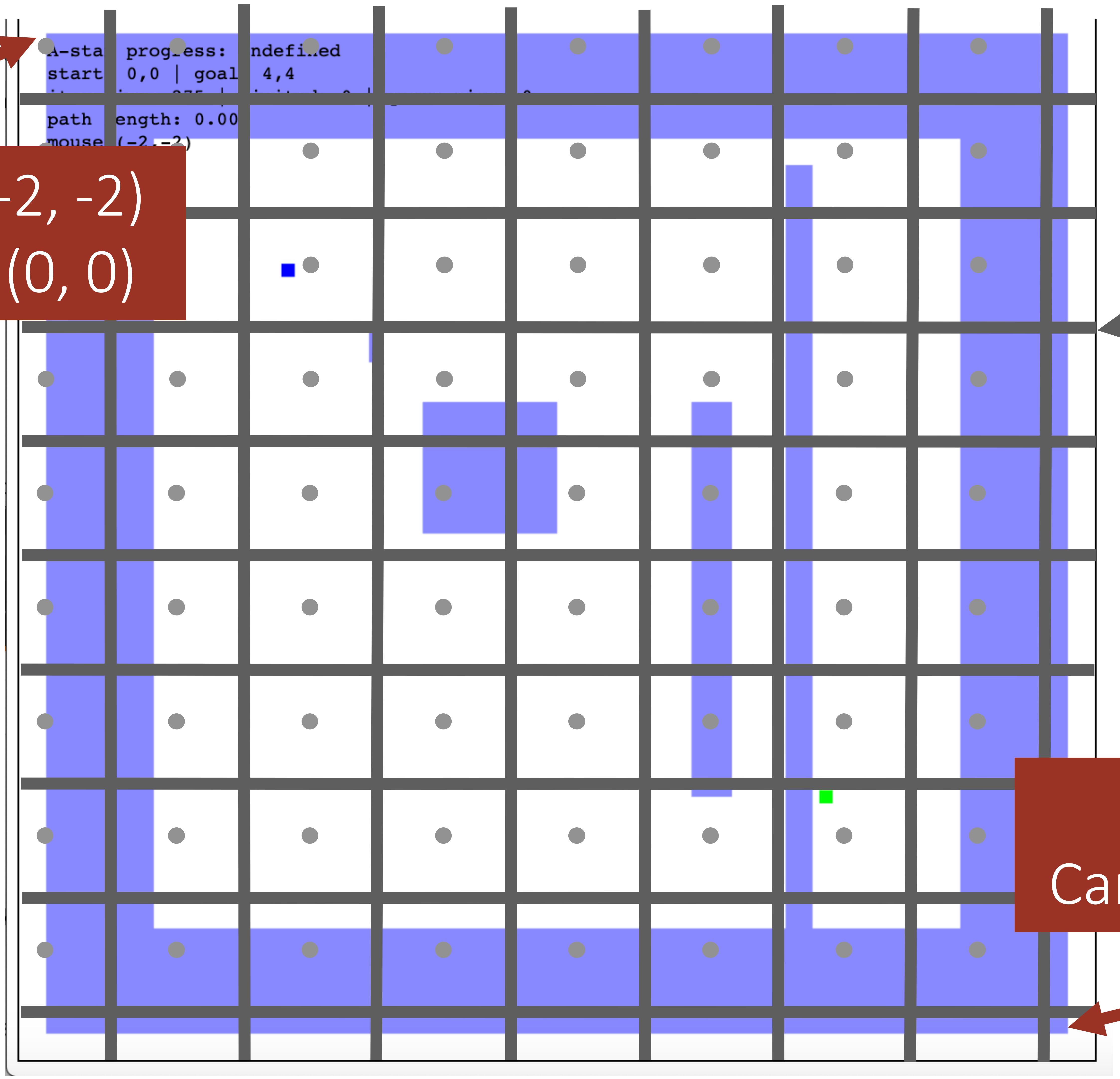


2D Search Canvas

```
A-st  progress: undefined
start: 0,  | goal: 4,4
iteration: 41  | visited:
path length: 0.00
mouse (4.26,-0.32)
```

World coordinates = (-2, -2)
Canvas coordinates = (0, 0)
Default q_init = (0, 0)
Default q_goal = (4, 4)
World coordinates = (7, 7)
Canvas coordinates = (800, 800)

Search Graph



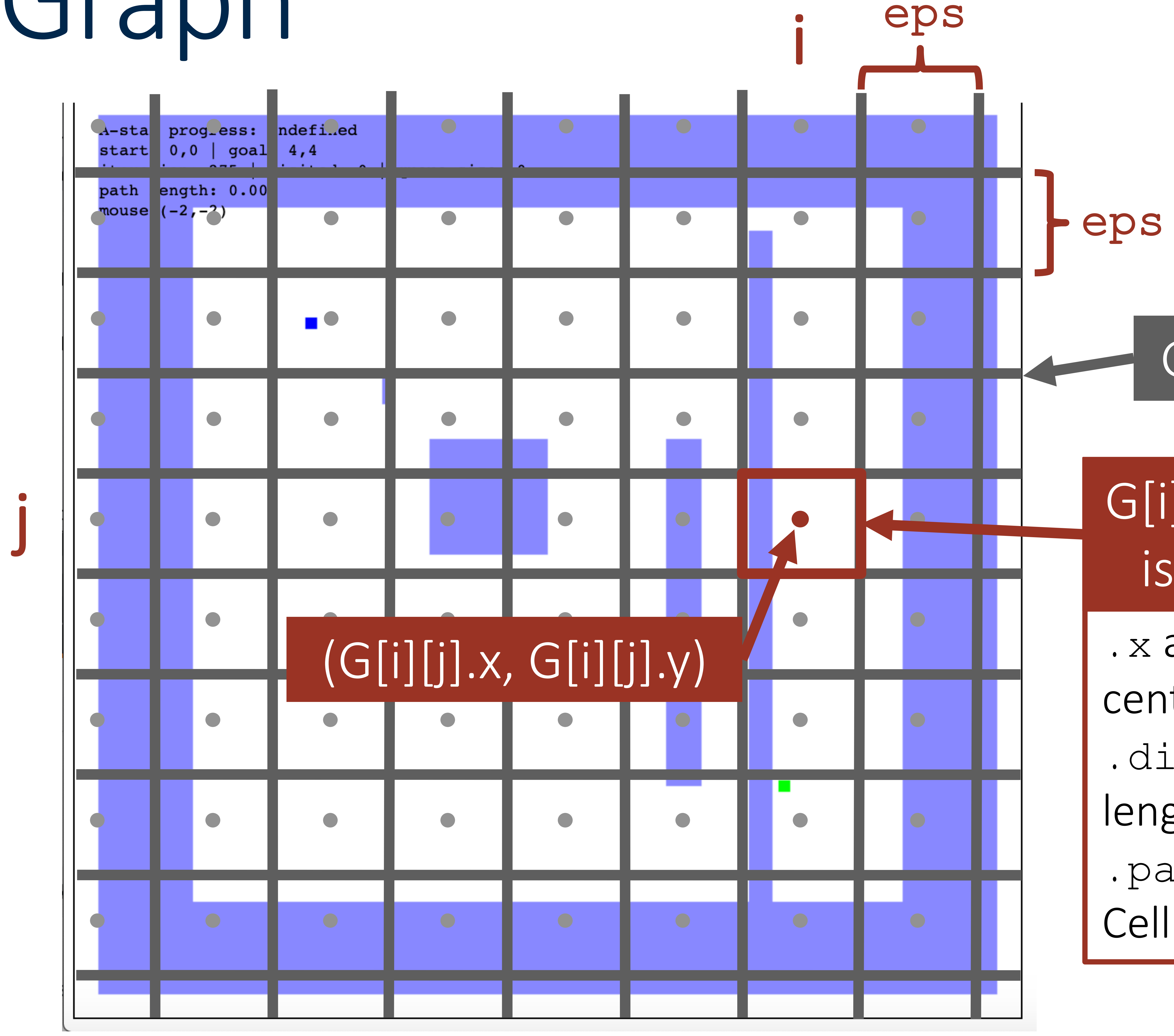
```
A-sta  progress:  ndefi ed
start  0,0 | goal  4,4
[illegible]
path  ength: 0.00
mouse (-2,-2)
```

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

G: search graph

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

Search Graph



```
A-star progress: undefined
start: 0,0 | goal: 4,4
[illegible]
path length: 0.00
mouse: (-2,-2)
```

G: search graph

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

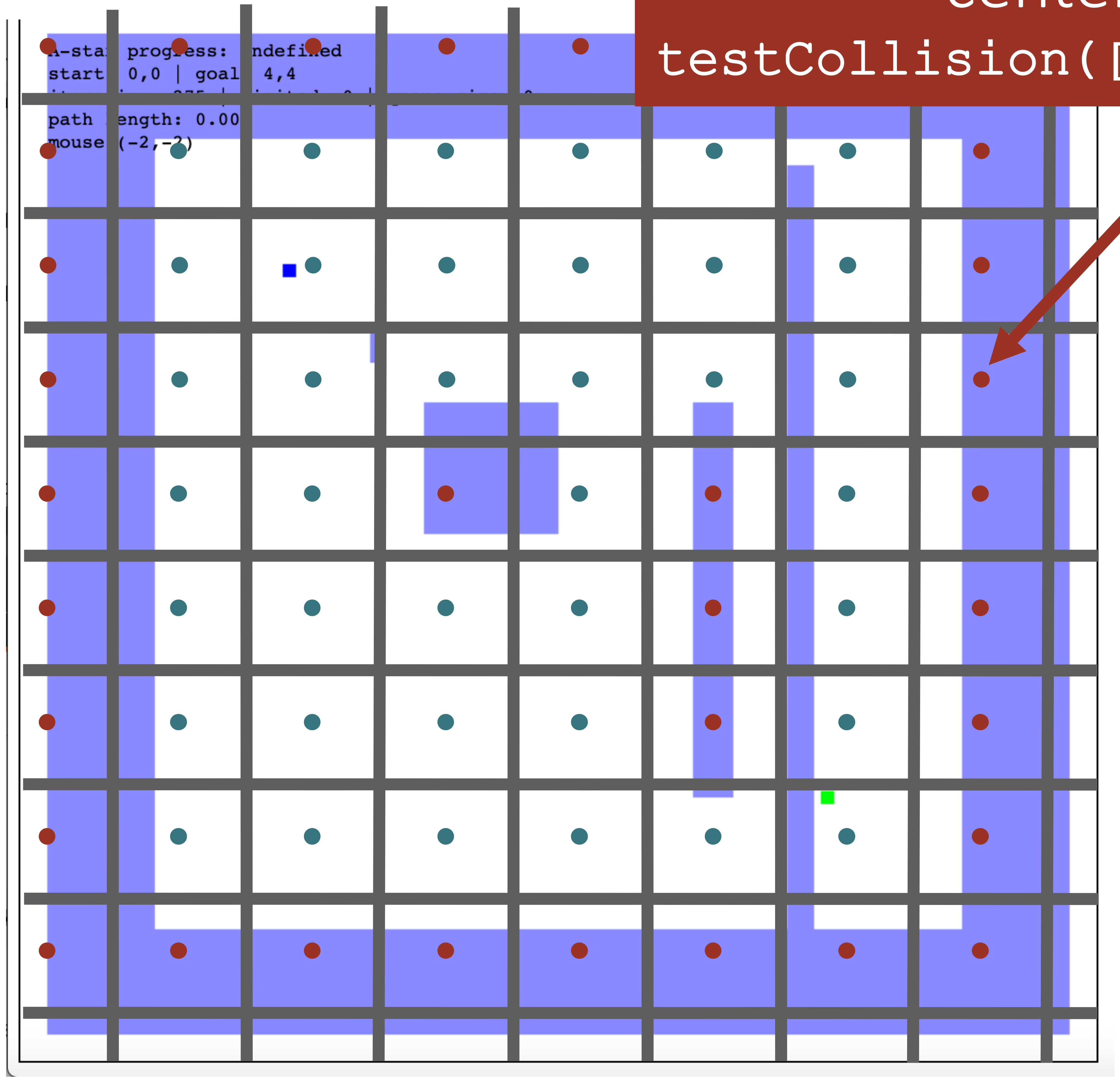
`.x` and `.y` specify cell center in world coordinates

`.distance` specifies path length to start through

`.parent` node

Cell height and width = `eps`

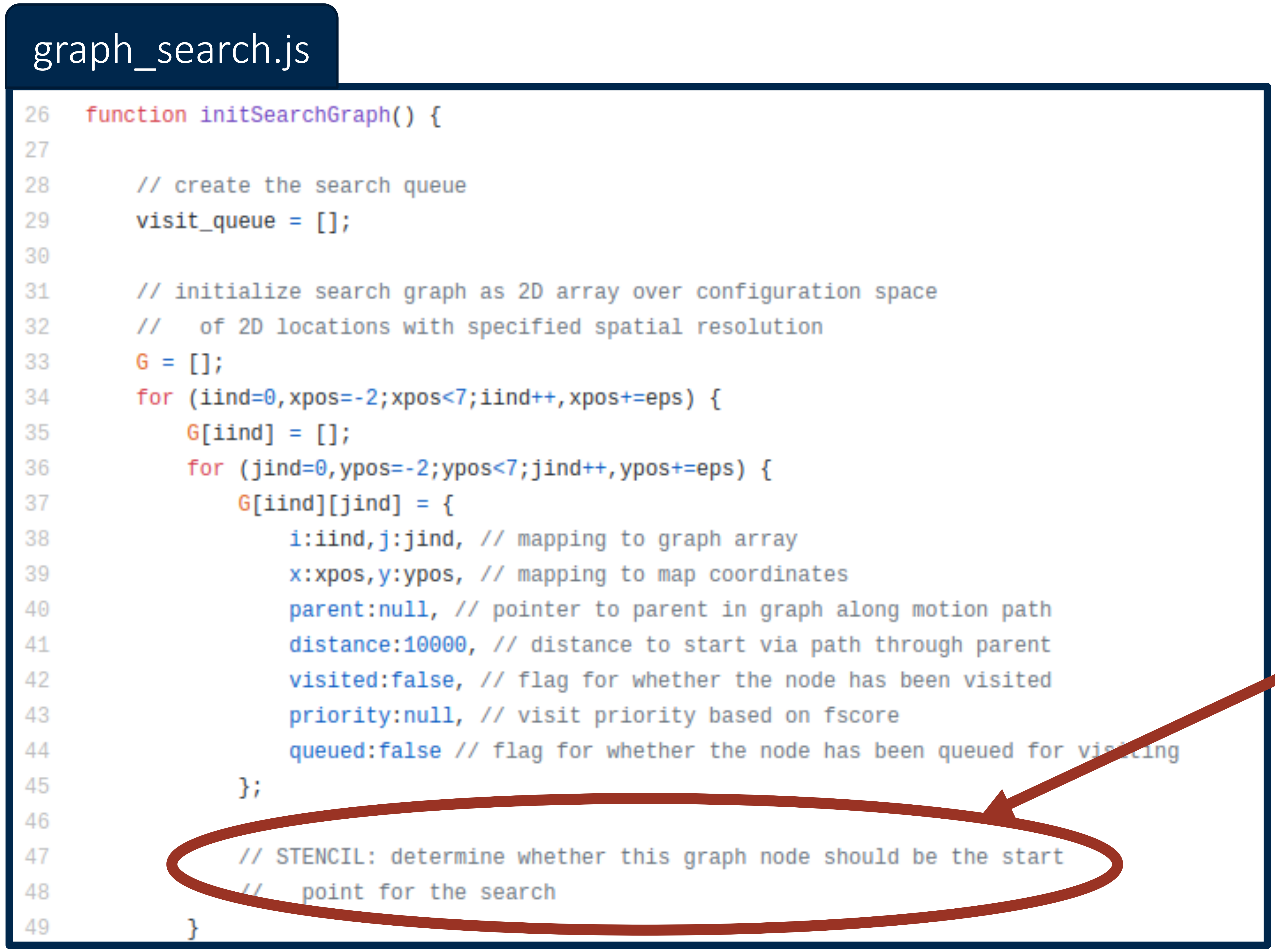
Collisions



```
A-star progress: undefined
start: 0,0 | goal: 4,4
[illegible]
path length: 0.00
mouse: (-2,-2)
```

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

Graph Search Initialization

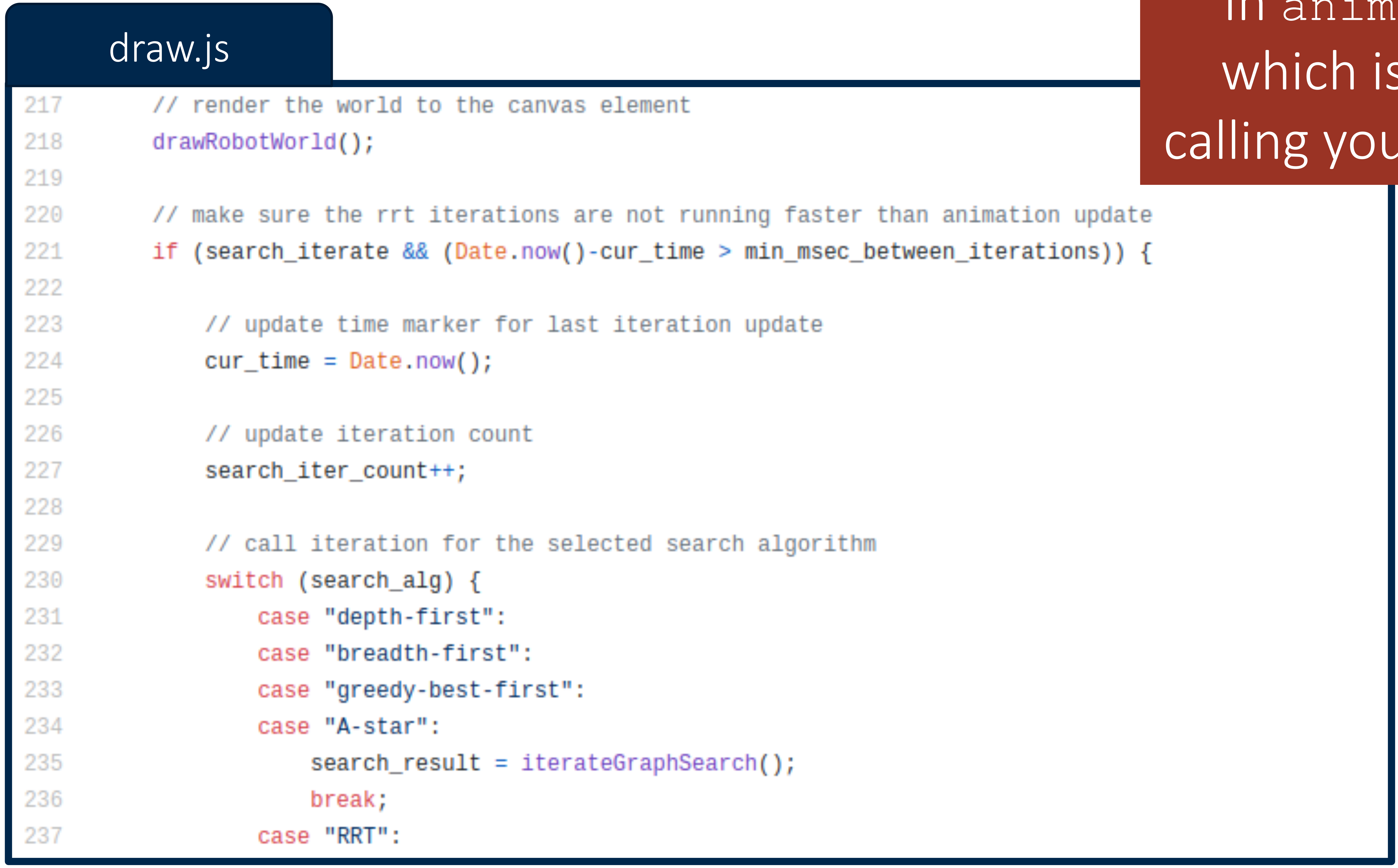


graph_search.js

```javascript
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

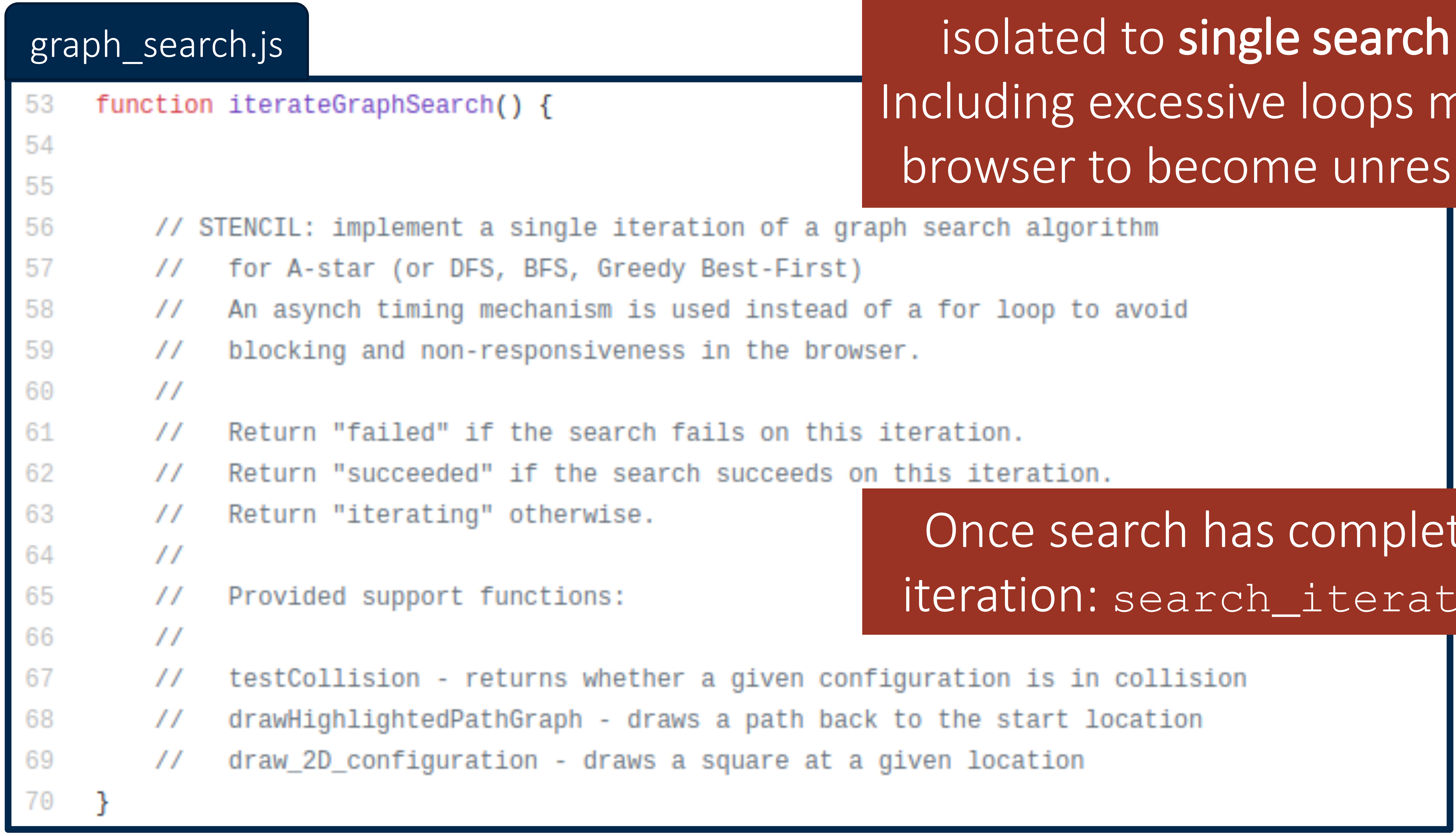
Graph Search Iteration


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":
```

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

Graph Search Iteration



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
}
```

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

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