Nice — I’ll trace your **heap sort** on the exact input vector<int> arr = {9,4,3,8,10,2,5} and **track** the important values (indices, children, selected largest, swaps) step-by-step. I’ll show both a human-readable explanation and a compact “console-style” log you can copy.

**Input**

[9, 4, 3, 8, 10, 2, 5] (indices 0..6, n = 7)

**PHASE 1 — Build max-heap**

We call heapify for i = n/2 - 1 down to 0 ⇒ i = 2, 1, 0.

**heapify(i = 2), n = 7**

* l = 2\*i+1 = 5, r = 2\*i+2 = 6
* values: arr[2]=3, arr[5]=2, arr[6]=5
* compare: left(2) > root(3)? NO. right(5) > root(3)? YES → largest = 6
* swap arr[2] ↔ arr[6] → swap 3 and 5
* array after swap: [9, 4, 5, 8, 10, 2, 3]
* recursive heapify(6) → l=13, r=14 out of range → stop.

**State:** [9, 4, 5, 8, 10, 2, 3]

**heapify(i = 1), n = 7**

* l = 3, r = 4
* values: arr[1]=4, arr[3]=8, arr[4]=10
* left(8) > 4 → largest = 3; right(10) > 8 → largest = 4
* swap arr[1] ↔ arr[4] → swap 4 and 10
* array: [9, 10, 5, 8, 4, 2, 3]
* recursive heapify(4) → children out of range → stop.

**State:** [9, 10, 5, 8, 4, 2, 3]

**heapify(i = 0), n = 7**

* l = 1, r = 2
* values: arr[0]=9, arr[1]=10, arr[2]=5
* left(10) > 9 → largest = 1; right(5) > 10? NO
* swap arr[0] ↔ arr[1] → swap 9 and 10
* array: [10, 9, 5, 8, 4, 2, 3]
* recursive heapify(1): l=3, r=4, values arr[1]=9, arr[3]=8, arr[4]=4 → no child > 9 → stop.

**Heap built:** [10, 9, 5, 8, 4, 2, 3] (max-heap)

**PHASE 2 — Sorting (extract max, reduce heap size, re-heapify)**

We will swap the root arr[0] with arr[i] and heapify on reduced heap size i. I’ll show each outer-loop step (i = 6 → 1).

**i = 6**

* swap arr[0] ↔ arr[6]: swap 10 and 3
* array after swap (before heapify): [3, 9, 5, 8, 4, 2, 10] (10 is now in final place)
* call heapify(arr, n=6, i=0):
  + l=1,r=2, vals 3,9,5 → left 9 > 3 → largest=1
  + swap arr[0]↔arr[1] → [9,3,5,8,4,2,10]
  + heapify(1): l=3,r=4, vals 3,8,4 → left 8 > 3 → largest=3
  + swap arr[1]↔arr[3] → [9,8,5,3,4,2,10]
  + heapify(3): children out of range → stop.

**State:** [9, 8, 5, 3, 4, 2, 10]

**i = 5**

* swap arr[0] ↔ arr[5]: swap 9 and 2
* before heapify: [2, 8, 5, 3, 4, 9, 10] (9 final)
* heapify(arr, n=5, i=0):
  + l=1,r=2, vals 2,8,5 → left 8 > 2 → largest=1
  + swap → [8,2,5,3,4,9,10]
  + heapify(1): l=3,r=4, vals 2,3,4 → left 3 > 2 → largest=3; right 4 > 3 → largest=4
  + swap arr[1]↔arr[4] → [8,4,5,3,2,9,10]
  + heapify(4): children out of range → stop.

**State:** [8, 4, 5, 3, 2, 9, 10]

**i = 4**

* swap arr[0] ↔ arr[4]: swap 8 and 2
* before heapify: [2,4,5,3,8,9,10] (8 final)
* heapify(arr, n=4, i=0):
  + l=1,r=2, vals 2,4,5 → left 4 > 2 → largest=1; right 5 > 4 → largest=2
  + swap arr[0]↔arr[2] → [5,4,2,3,8,9,10]
  + heapify(2): children out of range → stop.

**State:** [5, 4, 2, 3, 8, 9, 10]

**i = 3**

* swap arr[0] ↔ arr[3]: swap 5 and 3
* before heapify: [3,4,2,5,8,9,10] (5 final)
* heapify(arr, n=3, i=0):
  + l=1,r=2, vals 3,4,2 → left 4 > 3 → largest=1
  + swap arr[0]↔arr[1] → [4,3,2,5,8,9,10]
  + heapify(1): children out of range → stop.

**State:** [4, 3, 2, 5, 8, 9, 10]

**i = 2**

* swap arr[0] ↔ arr[2]: swap 4 and 2
* before heapify: [2,3,4,5,8,9,10] (4 final)
* heapify(arr, n=2, i=0):
  + l=1,r=2, vals 2,3 → left 3 > 2 → largest=1
  + swap arr[0]↔arr[1] → [3,2,4,5,8,9,10]
  + heapify(1): children out of range → stop.

**State:** [3, 2, 4, 5, 8, 9, 10]

**i = 1**

* swap arr[0] ↔ arr[1]: swap 3 and 2
* array becomes [2,3,4,5,8,9,10]
* heapify(arr, n=1, i=0) → no children → stop.

**Final sorted (ascending):** [2, 3, 4, 5, 8, 9, 10]

Initial: [9, 4, 3, 8, 10, 2, 5]

-- Build heap --

heapify(i=2): l=5(r=6), vals=(3,2,5) -> largest=6 -> swap idx2(3) with idx6(5)

-> [9, 4, 5, 8, 10, 2, 3]

heapify(i=1): l=3(r=4), vals=(4,8,10) -> largest=4 -> swap idx1(4) with idx4(10)

-> [9, 10, 5, 8, 4, 2, 3]

heapify(i=0): l=1(r=2), vals=(9,10,5) -> largest=1 -> swap idx0(9) with idx1(10)

-> [10, 9, 5, 8, 4, 2, 3]

-- After build: [10, 9, 5, 8, 4, 2, 3]

-- Sorting --

i=6: swap idx0(10) with idx6(3) -> [3,9,5,8,4,2,10]

heapify(0,n=6): -> swap idx0(3) with idx1(9) -> [9,3,5,8,4,2,10]

heapify(1): -> swap idx1(3) with idx3(8) -> [9,8,5,3,4,2,10]

i=5: swap idx0(9) with idx5(2) -> [2,8,5,3,4,9,10]

heapify(0,n=5): -> swap idx0(2) with idx1(8) -> [8,2,5,3,4,9,10]

heapify(1): -> swap idx1(2) with idx4(4) -> [8,4,5,3,2,9,10]

i=4: swap idx0(8) with idx4(2) -> [2,4,5,3,8,9,10]

heapify(0,n=4): -> swap idx0(2) with idx2(5) -> [5,4,2,3,8,9,10]

i=3: swap idx0(5) with idx3(3) -> [3,4,2,5,8,9,10]

heapify(0,n=3): -> swap idx0(3) with idx1(4) -> [4,3,2,5,8,9,10]

i=2: swap idx0(4) with idx2(2) -> [2,3,4,5,8,9,10]

heapify(0,n=2): -> swap idx0(2) with idx1(3) -> [3,2,4,5,8,9,10]

i=1: swap idx0(3) with idx1(2) -> [2,3,4,5,8,9,10]

done.

Final: [2, 3, 4, 5, 8, 9, 10]

Perfect 👍 — let’s visualize **Heap Sort** step-by-step using a **binary tree layout** instead of flat arrays.

Your input:

[9, 4, 3, 8, 10, 2, 5]

9

/ \

4 3

/ \ / \

8 10 2 5

## Build Heap Phase

### heapify(i=2)

Check node 3 with children 2 and 5. Largest is 5. Swap.

9

/ \

4 5

/ \ / \

8 10 2 3

### heapify(i=1)

Check node 4 with children 8 and 10. Largest is 10. Swap.

9

/ \

10 5

/ \ / \

8 4 2 3

### heapify(i=0)

Check node 9 with children 10 and 5. Largest is 10. Swap.

10

/ \

9 5

/ \ / \

8 4 2 3

✅ **Max Heap built**

## orting Phase

### Step i = 6 (swap root with last element)

Swap 10 ↔ 3

3

/ \

9 5

/ \ / \

8 4 2 [10]

Heapify root: 3 vs 9,5 → largest 9. Swap. Then fix child.

9

/ \

8 5

/ \ / \

3 4 2 [10]

### Step i = 5 (swap root with index 5)

Swap 9 ↔ 2

2

/ \

8 5

/ \ / \

3 4 [9] [10]

Heapify: 2 vs 8,5 → swap with 8. Then 2 vs 3,4 → swap with 4.

8

/ \

4 5

/ \ / \

3 2 [9] [10]

### Step i = 4 (swap root with index 4)

Swap 8 ↔ 2

2

/ \

4 5

/ \ / \

3 [8] [9] [10]

Heapify: 2 vs 4,5 → swap with 5.

5

/ \

4 2

/ \ / \

3 [8] [9] [10]

### Step i = 3 (swap root with index 3)

Swap 5 ↔ 3

3

/ \

4 2

/ \ / \

[5] [8] [9] [10]

Heapify: 3 vs 4,2 → swap with 4.

4

/ \

3 2

/ \ / \

[5] [8] [9] [10]

### Step i = 2 (swap root with index 2)

Swap 4 ↔ 2

2

/ \

3 [4]

/ \ / \

[5] [8] [9] [10]

Heapify: 2 vs 3 → swap.

3

/ \

2 [4]

/ \ / \

[5] [8] [9] [10]

### Step i = 1 (swap root with index 1)

Swap 3 ↔ 2

2

/ \

[3] [4]

/ \ / \

[5] [8] [9] [10]

Heapify size=1 → nothing.

2

/ \

3 4

/ \ / \

5 8 9 10