

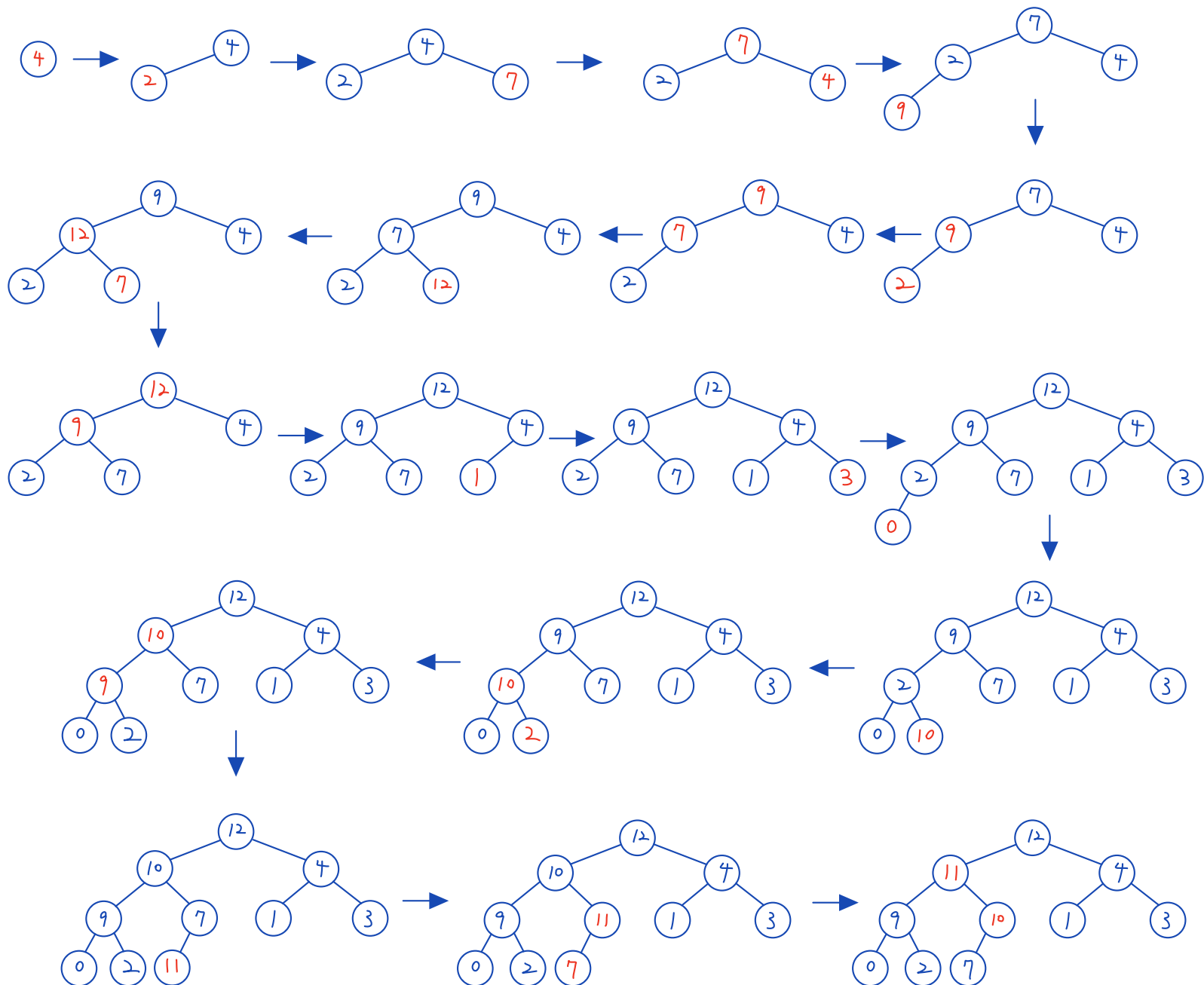
1. My Answer: $O(n \log(n))$

My reason:

In the "build_heap" algorithm, we iterate through all element in the array and implement the "heap_insert" algorithm each time, every "heap_insert" operation will have complexity of $O(\log(n))$, so we can get upper bound overall complexity of $O(n) * O(\log(n)) = O(n \log(n))$.

2. My Answer: [12, 11, 4, 9, 10, 1, 3, 0, 2, 7]

My steps:

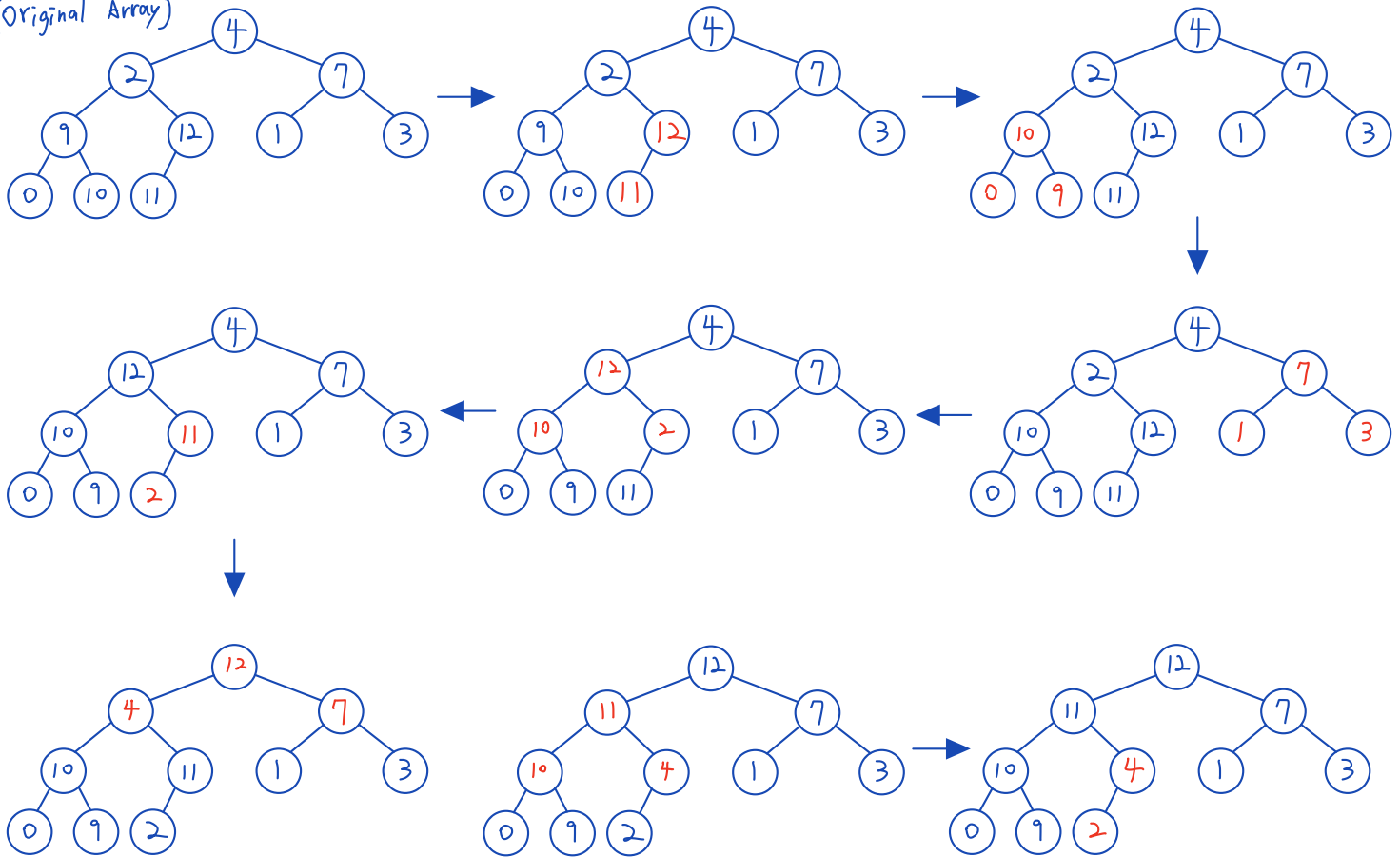


∴ After above steps, we get [12, 11, 4, 9, 10, 1, 3, 0, 2, 7]

3. My Answer: [12, 11, 7, 10, 4, 1, 3, 0, 9, 2]

My steps:

(Original Array)



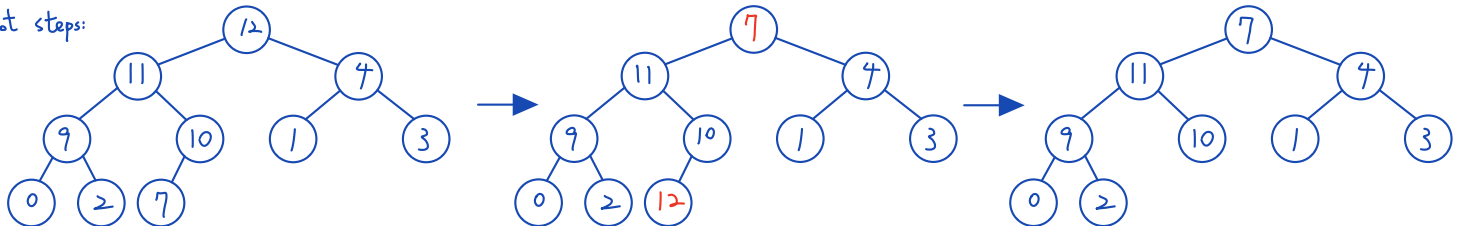
∴ After above steps, we get [12, 11, 7, 10, 4, 1, 3, 0, 9, 2]. #

4. My Answer: [11, 10, 4, 9, 7, 1, 3, 0, 2]

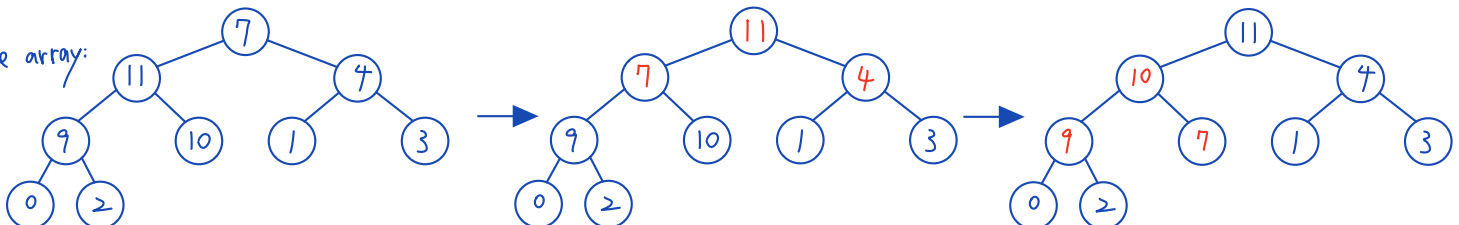
My steps:

from problem 2 we got [12, 11, 4, 9, 10, 1, 3, 0, 2, 7]

Remove the root steps:



Heapify the array:



∴ After above steps, we get [11, 10, 4, 9, 7, 1, 3, 0, 2]. #

5. My pseudocode:

```
Problem5
Heapify

1 # Hsuan-You Lin_Module 7 Problem Set - Problem5
2 def Heapify(arr, start, end):
3     for i in range(end//2 - 1, start - 1, -1):
4         Left = 2 * i + 1 # left = 2*i + 1
5         Right = 2 * i + 2 # right = 2*i + 2
6         root = i # Initialize largest as root
7
8         if arr[root] < arr[Left]:
9             root = Left
10        if Right < end and arr[root] < arr[Right]:
11            root = Right
12
13        arr[i], arr[root] = arr[root], arr[i]
14        if root != i:
15            Heapify(arr, root, end)
16
17 def HeapSort(arr):
18     for i in range(len(arr) - 1, -1, -1):
19         Heapify(arr, 0, i+1)
20         arr[i], arr[0] = arr[0], arr[i]
21
22 #/-----main function-----
23
24 if __name__ == '__main__':
25     arr = [4, 2, 7, 9, 12, 1, 3, 0, 10, 11]
26     HeapSort(arr)
27     print("Sorted array is:", arr)

Module7 — -bash — 80x24
[(base) pisces:Module7 pisces$ python Problem5.py
Sorted array is: [0, 1, 2, 3, 4, 7, 9, 10, 11, 12]
(base) pisces:Module7 pisces$]
```

6. My Answer: $O(n \log(n))$

My reason:

In my problem 5 pseudocode used a loop "for i in range(len(arr), -1, -1)", which means it has the worst case when the heapify operation requires sinking through the tree's length, let length = n, so we got complexity of $O(\log(n))$.

Thus, we can calculate the worst case complexity of my method is $O(n \log(n))$.