Exercise Sheet 4

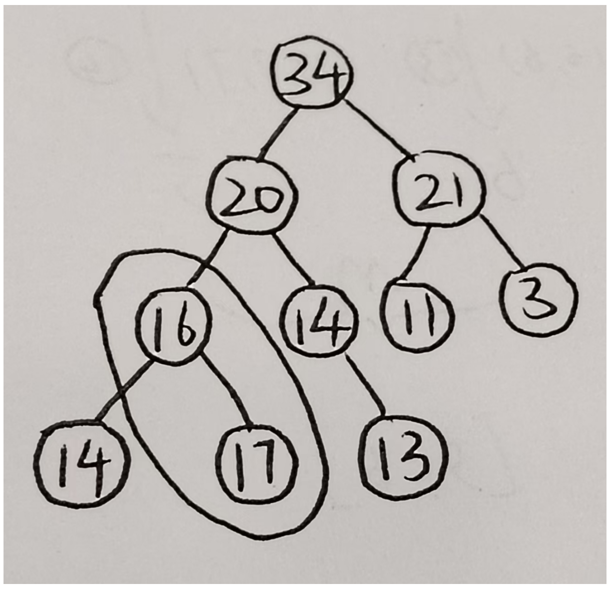
Handout: Sep 30 — Deadline: Oct 12, 4pm

1 Question 4.1

(0.1 marks) Say whether the following array is a Max-Heap (justify your answer):

34	20	21	16	14	11	3	14	17	13

A max-heap is a heap that every parent node is not smaller than its child node.Let draw the tree to see whether it satisfies this property.



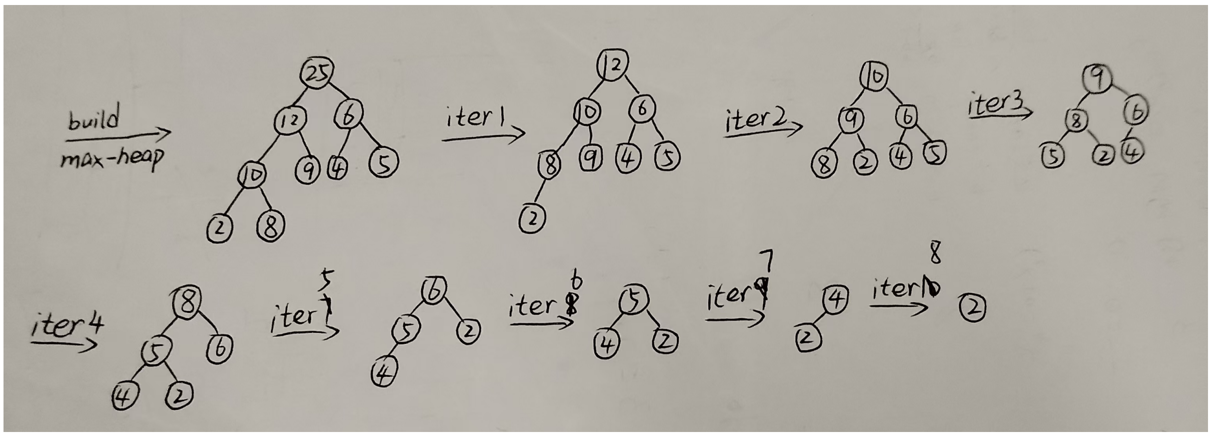
parent node 16 is smaller than its child node 17, so it's not a max-heap.

2 Question 4.2

(0.1 marks)
Consider the following input for Heapsort:

12	10	4	2	9	6	5	25	8

Create a heap from the given array and sort it by executing Heapsort. Draw the heap (the tree) after Build-Max-Heap and after every execution of Max-Heapify in line 5 of Heapsort. You don't need to draw elements extracted from the heap, but you can if you wish.



3 Question 4.3

(0.5 marks)

1. Provide the pseudo-code of a Max-Heapify(A, i) algorithm that uses a While loop instead of the recursion used by the algorithm shown at lecture.
2. Prove correctness of the algorithm by loop invariant.

1. **Pseudo-code of the iterative Max-Heapify(A, i):**

Algorithm 1 Max-Heapify(A, i) (iterative version)

```
while true do
    $l = 2i$
    $r = 2i + 1$
    $largest = i$
    if $l \leq$ heap_size[$A$] and $A[l] > A[largest]$ then
        $largest = l$
    end if
    if $r \leq$ heap_size[$A$] and $A[r] > A[largest]$ then
        $largest = r$
    end if
    if $largest \neq i$ then
        Exchange($A[i], A[largest]$)
        $i = largest$
    else
        break
    end if
end while
```

2. **Proof of correctness (by loop invariant):**

   Loop Invariant: At the start of each iteration of the **while** loop, the subtrees rooted at the left A[2i] and right A[2i+1] children i are max-heaps, and the tree whithout the subtree rooted at A[i] (cut off a subtree from original tree) is max-heaps.

   Initialization: Before the first iteration, we are given that the left and right subtrees of the original node i are already max-heaps, and the tree whithout the subtree rooted at A[i] is empty. Thus, the loop invariant holds initially.

   Maintenance: During an iteration, we compare $A[i]$, $A[l]$, and $A[r]$ to find the largest element. If one of the children is larger than $A[i]$, we swap $A[i]$ with the largest child and move i downward. After the swap, both subtrees of the new i remain max-heaps, since only the root of the subtree has changed. And every swap make the larger node become the parent of the smaller node, which satisfies the max-heap property. So the tree whithout the subtree rooted at A[i] is max-heap as well. Therefore, the loop invariant is maintained.

Termination: The loop terminates when $A[i] \geq A[l]$ and $A[i] \geq A[r]$ (or when these children do not exist). At this point, the node i satisfies the max-heap property, and since its subtrees the tree whithout the subtree rooted at A[i] are max-heaps by the invariant, the entire subtree rooted at the original i is a max-heap. MAX-HEAPIFY.

4 Question 4.4

(1.25 marks)

1. Show that each child of the root of an n-node heap is the root of a sub-tree of at most $(2/3)n$ nodes. (*HINT: consider that the maximum number of elements in a subtree happens when the left subtree has the last level full and the right tree has the last level empty. You might want to use the formula seen at lecture:* $\sum_{i=0}^{k-1} 2^i = 2^k - 1$).

2. As a consequence of (1) we can use the recurrence equation $T(n) \leq T(2n/3) + \Theta(1)$ to describe the runtime of Max-Heapify(A, n). Prove the runtime of Max-Heapify using the Master Theorem.

1. The maximum number of elements in a subtree happens when the left subtree has the last level full and the right tree has the last level empty. Assume the tree's height is h, left subtree' height is h-1 and right's is h-2, both are full in its own last level.Then left subtree has $\sum_{i=0}^{h-1} 2^i = 2^h - 1$ nodes and right tree has $\sum_{i=0}^{h-2} 2^i = 2^{h-1} - 1$, hence the whole tree has $1 + 2^h - 1 + 2^{h-1} - 1 = 3 \times 2^{h-1} - 1$ nodes. We can see that $2^h - 1 = \frac{2}{3} \times (3 \times 2^{h-1} - 1)$, so each child of the root of an n-node heap is the root of a sub-tree of at most $(2/3)n$ nodes.

2. In $T(n) \leq T(2n/3) + \Theta(1)$, $a = 1, b = \frac{3}{2}, n^{log_b a} = 1, f(n) = \Theta(1), f(n) = \Theta(n^{log_b a})$. So $T(n) = logn$.

5 Question 4.5

(1 mark)
Argue that the runtime of HEAPSORT on an already sorted array of distinct numbers is $\Omega(n \log n)$.

A already sorted array is a max-heap (or min-heap) naturally.In HEAPSORT algorithm, first is BUILD-MAX-HEAP, in which we call $\Theta(n)$ times MAX-HEAPIFY, and each call takes $\Theta(1)$ time since the array is already a max-heap, so the running time of BUILD-MAX-HEAP is $\Theta(n)$. Then We iter $\Theta(n)$ times, each time we exchange the root and the last node in the heap (which break the heap property!), extract the original "root", and shrink the heap size by one. After that we call MAX-HEAPIFY, and each call run $\Omega(logn)$ time to rebuild the heap since the smallest node need to be moved from root to leaf, so the loop's runtime is $\Omega(nlogn)$. As a result, the total runtime of HEAPSORT algorithm is $\Omega(nlogn)$.