

Solutions for Exercise Sheet 4

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

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
----	----	----	----	----	----	---	----	----	----

Solution: It's not a Max-Heap because node 17 violates the max-heap property as it's larger than its parent (i.e., 16).

Question 4.2 (0.1 marks)

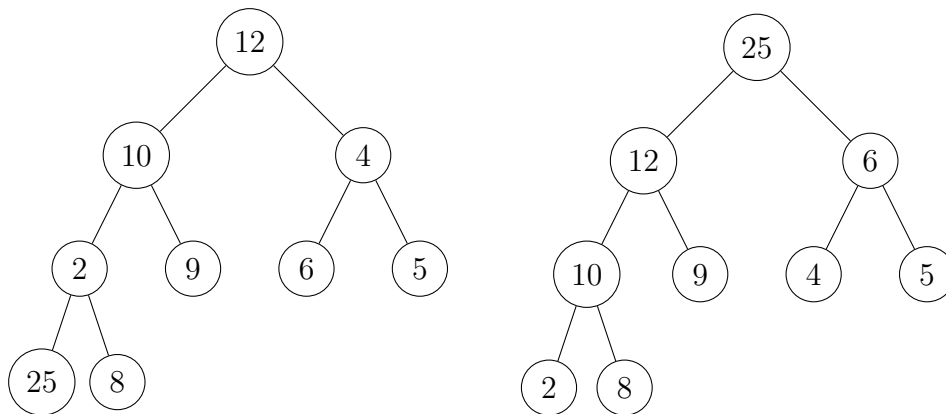
Consider the following input for HEAPSORT:

12	10	4	2	9	6	5	25	8
----	----	---	---	---	---	---	----	---

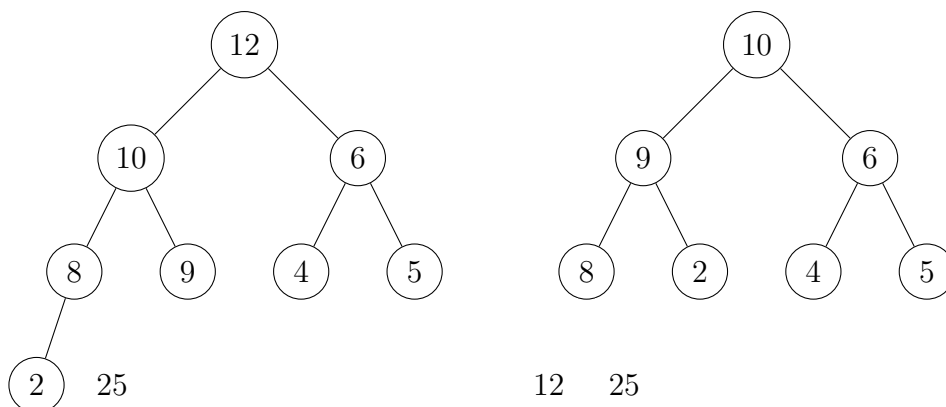
Create a heap from the given array and sort it by executing HEAPSORT. Draw the heap (the tree) after BUILD-MAX-HEAP and after every execution of MAX-HEAPIFY in line 5 of HEAPSORT. You don't need to draw elements extracted from the heap, but you can if you wish.

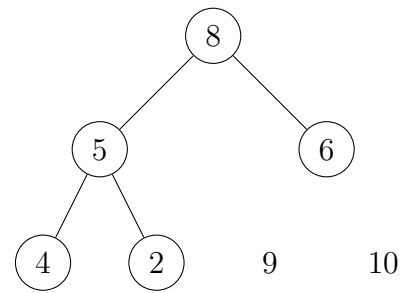
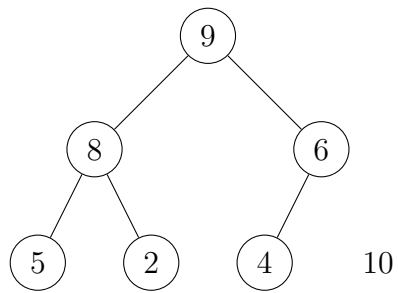
Solution:

Initial structure (for completeness, this wasn't asked for) and after BUILD-MAX-HEAP:

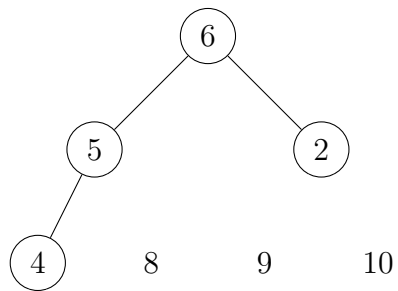


After each call of MAX-HEAPIFY:

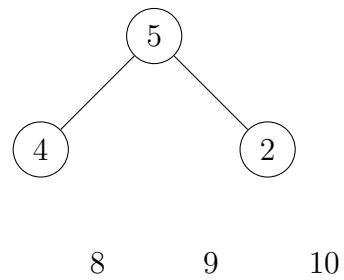




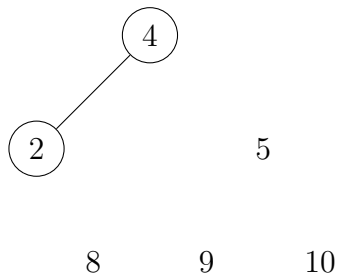
12 25



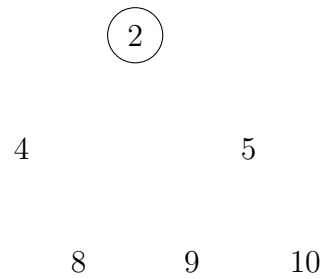
12 25



12 25



12 25



12 25

12 25

Question 4.3 (0.5 marks)

1. Provide the pseudo-code of a $\text{MAX-HEAPIFY}(A, i)$ 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.

Solution: 1)

MAX-HEAPIFY(A, i)

```
1: while true do
2:    $\ell = \text{Left}(i)$ 
3:    $r = \text{Right}(i)$ 
4:   if  $\ell \leq A.\text{heap-size}$  and  $A[\ell] > A[i]$  then
5:      $\text{largest} = \ell$ 
6:   else
7:      $\text{largest} = i$ 
8:   if  $r \leq A.\text{heap-size}$  and  $A[r] > A[i]$  then
9:      $\text{largest} = r$ 
10:  if  $\text{largest} == i$  then
11:    return
12:  Swap  $A[i]$  with  $A[\text{largest}]$ 
13:   $i = \text{largest}$ 
```

2) We set the following loop invariant: " Let ROOT be the node at which MAX-HEAPIFY(A, i) is called i.e., the initial i . At each iteration of the WHILE loop, the tree rooted in ROOT obtained by removing the two subtrees with roots Left(i) and Right(i) is a Max-Heap"

- **Initialisation:** Before the loop starts the considered tree consists only of the node ROOT which is trivially a Max-Heap.
- **Maintenance:** By loop invariant after i iterations the tree rooted in ROOT down to node i is a heap. In the current iteration we swap i with the largest between i , Left(i) and Right(i) ensuring that by the end of the iteration we have a heap of height $i + 1$.
- **Termination** the algorithm terminates when either: 1) i a leaf. Then the loop invariant says that the whole tree rooted in ROOT is a max-heap (because the two subtrees Left(i) and Right(i) are empty). 2) the heap-property is respected before a leaf is reached. Then the algorithm returns a Max-Heap because the loop invariant tells me that up to node i we have a Max-Heap and Left(i) and Right(i) are heaps too because the subtrees of ROOT were originally Max-Heaps and below i they are still untouched.

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.

Solution: 1) The size of the whole tree is 1 (the root) + |Left subtree| + |right subtree|. Let the right subtree have height h and the left subtree be full at height $h + 1$. Then:

$$|\text{Left subtree}| = 1 + 2 + 4 \cdots + 2^h + 2^{h+1} = \sum_{i=0}^{h+1} 2^i = 2^{h+2} - 1 \quad (1)$$

$$|\text{Right subtree}| = 1 + 2 + 4 \cdots + 2^h = \sum_{i=0}^h 2^i = 2^{h+1} - 1$$

Then,

$$n = 1 + (2^{h+2} - 1) + (2^{h+1} - 1) = 1 + (2(2^{h+1}) - 1) + (2^{h+1} - 1) = 3(2^{h+1}) - 1$$

So $2^{h+1} = \frac{n+1}{3}$, and substituting into Eq (1),

$$|\text{Left subtree}| = 2^{h+2} - 1 = 2(2^{h+1}) - 1 = 2 \frac{n+1}{3} - 1 = (2/3)n - 1/3 < (2/3)n$$

2) We apply the Master Theorem with $a = 1, b = 3/2, f(n) = \Theta(1)$, and watershed function $n^{\log_b a} = n^0 = 1$, because $\log_{1.5} 1 = 0$. Thus, Case 2 holds with $k = 0$ such that $n^{\log_b a} \log^k n = \Theta(1)$ and $T(n) = \Theta(n^0 \log^{k+1} n) = \Theta(\log n)$, and the runtime of Max-Heapify(A, n) is $O(\log n)$ since each subtree has *at most* $(2/3)n$ nodes.

Question 4.5 (1 mark)

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

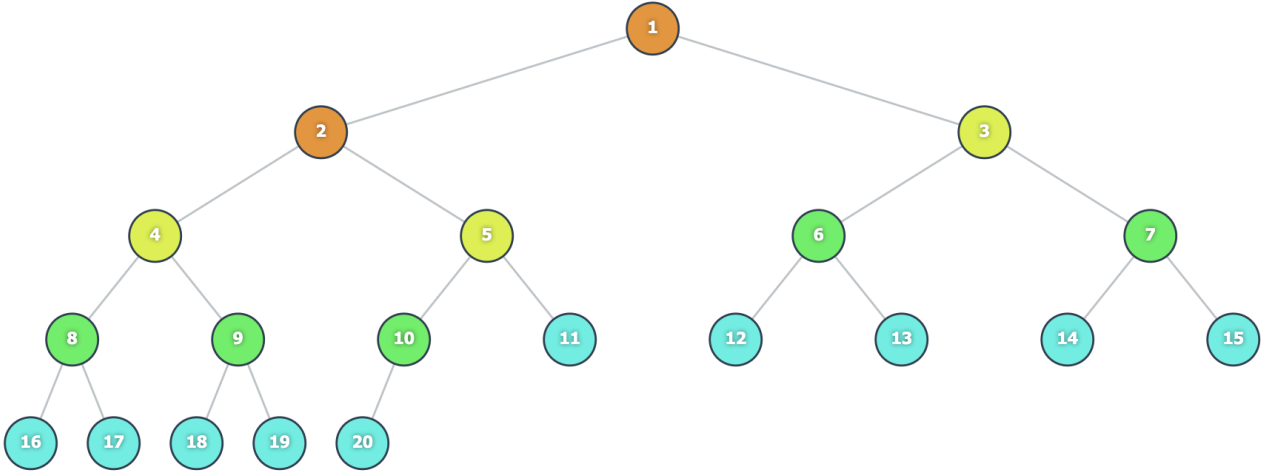


Figure 1: painted heap, blue: \mathcal{L}_0 , green: \mathcal{L}_1 , yellow: \mathcal{L}_2 , Orange: \mathcal{L}_3

Solution [By student Yang Yiming]: We will prove that BUILD-MAX-HEAP will move at least the $\lfloor n/4 \rfloor$ smallest elements to the end of the array (to be all leaves). Then each of these $\lfloor n/4 \rfloor$ HEAPIFY operations will have to traverse at least $\log n - 1$ levels after the root of the tree is swapped with the last element of the heap to reconstruct a Max-Heap. Hence the runtime is

$$T(n) \geq (n/4 - 1)(\log n - 1) = \Omega(n \log n)$$

Now we show that the $\lfloor n/4 \rfloor$ smallest elements will be leaves by the end of the BUILD-MAX-HEAP operation. Let "height" be the minimal distance between a node to the leaves, and let $h \circ x$ be node x 's height.

In order to describe nodes' positions easier, we divide the heap into 4 layers: \mathcal{L}_0 be the leaves, $\mathcal{L}_i = \{x : \max[h \circ \text{Left}(x), h \circ \text{Right}(x)] = i\}$ $i = 1, 2, 3$. Then we partition the nodes by value: let A be the set of the largest $\lceil n/2 \rceil$ nodes, B be the following largest $\lfloor n/4 \rfloor$ nodes, C be the following largest $\lceil n/8 \rceil$ nodes, and D be the set of the rest nodes.

Then as shown in Fig. 1, initially all A nodes are at \mathcal{L}_0 , all B nodes are in \mathcal{L}_1 , all C nodes in \mathcal{L}_2 , and D in \mathcal{L}_3 .

Now we prove: After BUILD-MAX-HEAP, the smallest $\lfloor n/4 \rfloor$ nodes would be in the leaves. (i.e. C, D nodes in leaves)

During the first $\lfloor n/4 \rfloor$ steps of BUILD-MAX-HEAP, each node in set B (at \mathcal{L}_1) would be swapped to its right leaf. Subsequently in the second $\lceil n/8 \rceil$ steps, nodes in C (at \mathcal{L}_2) would be firstly swapped to the \mathcal{L}_1 layer, then to the left child because the left child is in A while right child in B). After this step, all C nodes are leaves, and no two C nodes share same parent. The remaining $\lceil n/2 \rceil - \lceil n/8 \rceil \approx 3/8n$ leaves, as well as nodes at level $\mathcal{L}_1, \mathcal{L}_2$, are in A, B . Hence every \mathcal{L}_3 nodes (D nodes) could reach the bottom of the heap. Therefore the smallest $\lfloor n/4 \rfloor$ nodes would be in the leaves.

During the MAX-HEAPIFY steps, the total runtime contributed by the smallest $\lfloor n/4 \rfloor$ nodes is $\Theta(n \log n)$. Therefore, the overall running time of heap-sort in this case is $\Omega(n \log n)$.

Question 4.6 (0.45 marks)

Implement HEAPSORT(A, n) and the two problems "Heap" and "Heap Operations" on the Judge system.