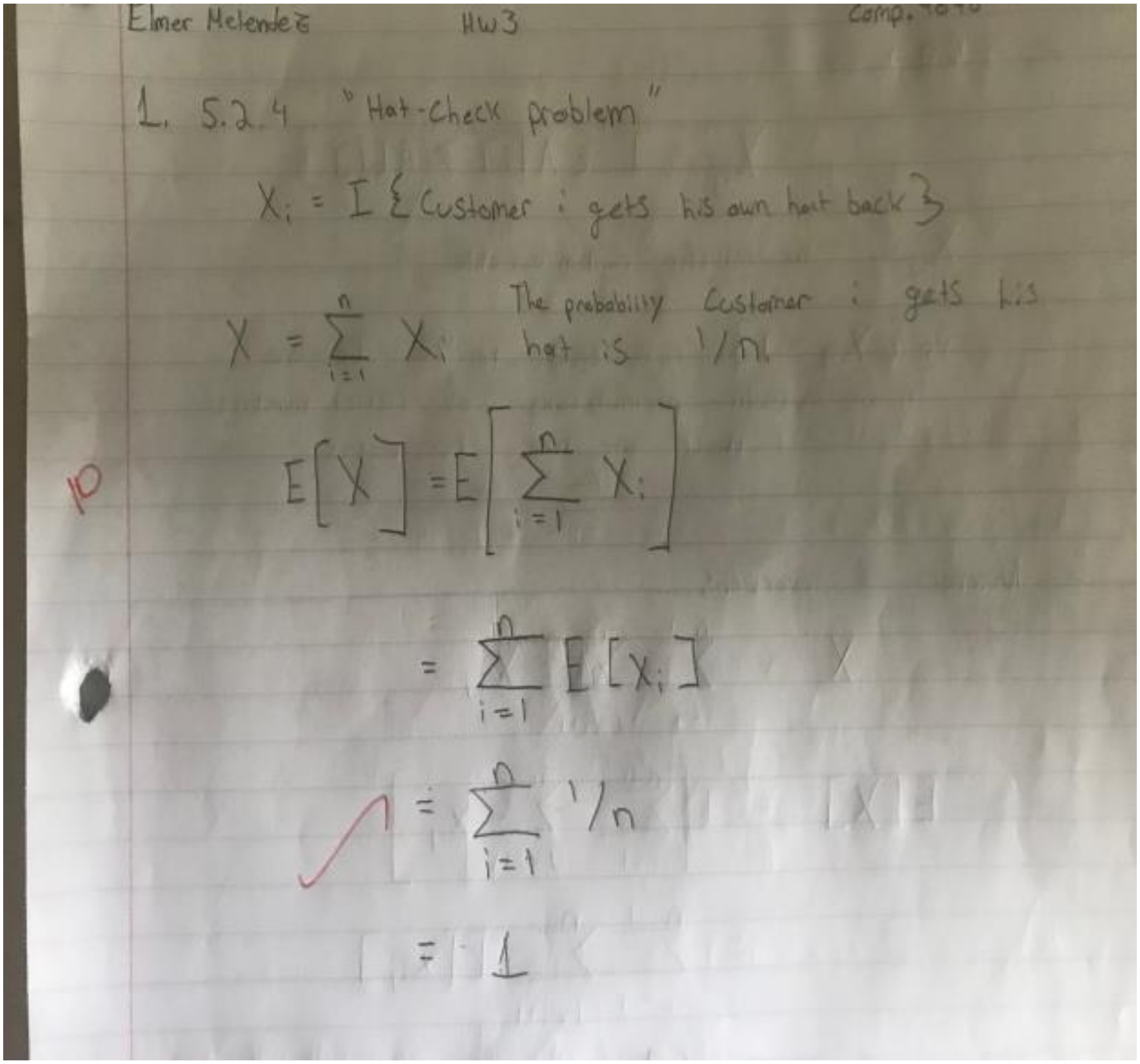
Prof. Benyuan Liu
Fall - 2017

ANALYSIS OF ALGORITHM - HW -3 SOLUTIONS

1. Credits: Elmer Melendz



Elmer Melendez Hw3 Comp.

1. 5.2.4 "Hat-Check problem"

$$X_i = I\{\text{Customer } i \text{ gets his own hat back}\}$$

$$X = \sum_{i=1}^{n} X_i$$ The probability Customer i gets his hat is $1/n$

10

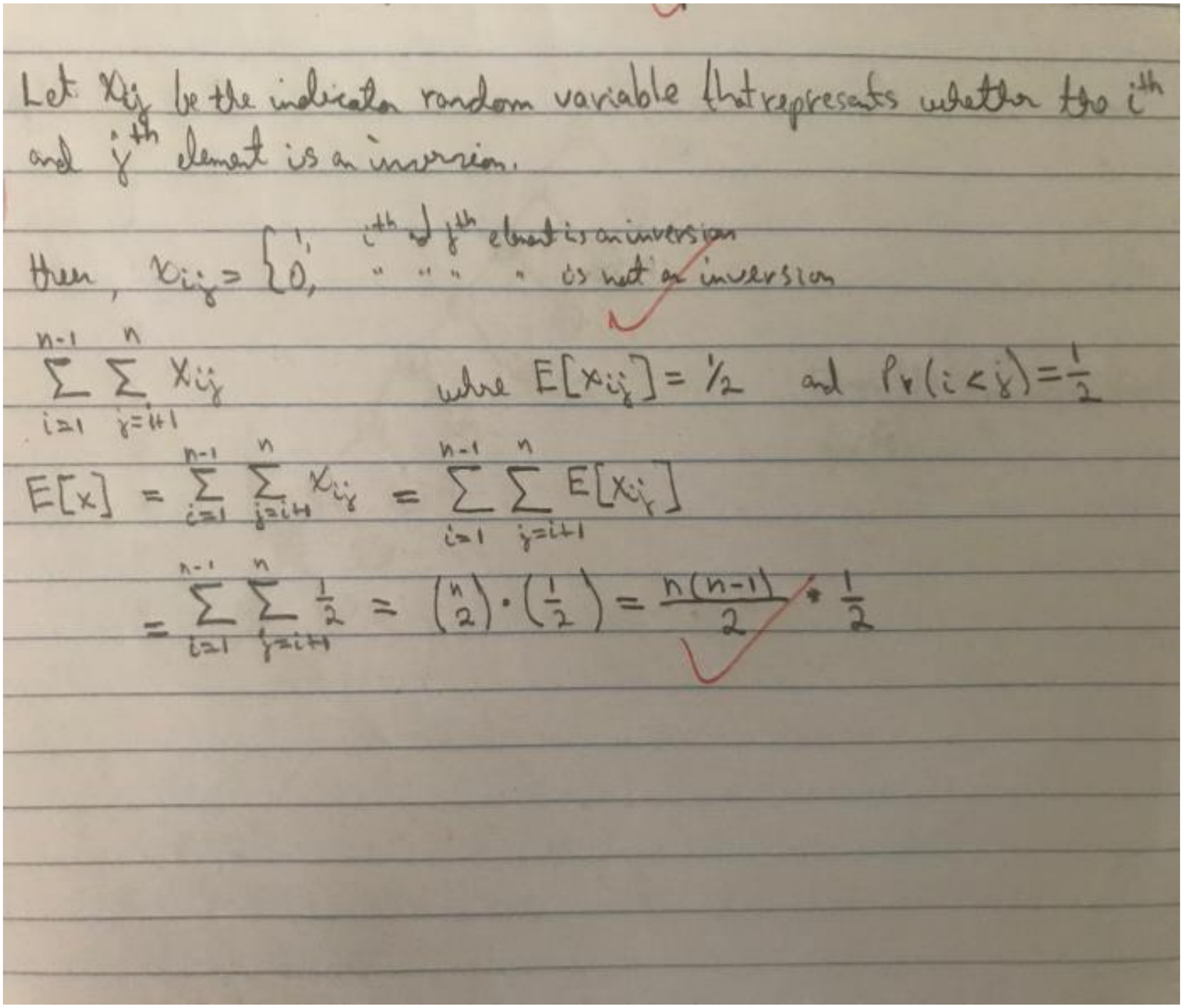
$$E[X] = E\left[\sum_{i=1}^{n} X_i\right]$$

$$= \sum_{i=1}^{n} E[X_i]$$

$$= \sum_{i=1}^{n} 1/n$$

$$= 1$$

2. Credits: Oscar



Let X_{ij} be the indicator random variable that represents whether the i^{th} and j^{th} element is an inversion.

then, $X_{ij} = \begin{cases} 1, & i^{th} \text{ and } j^{th} \text{ element is an inversion} \\ 0, & \text{" " " " is not an inversion} \end{cases}$

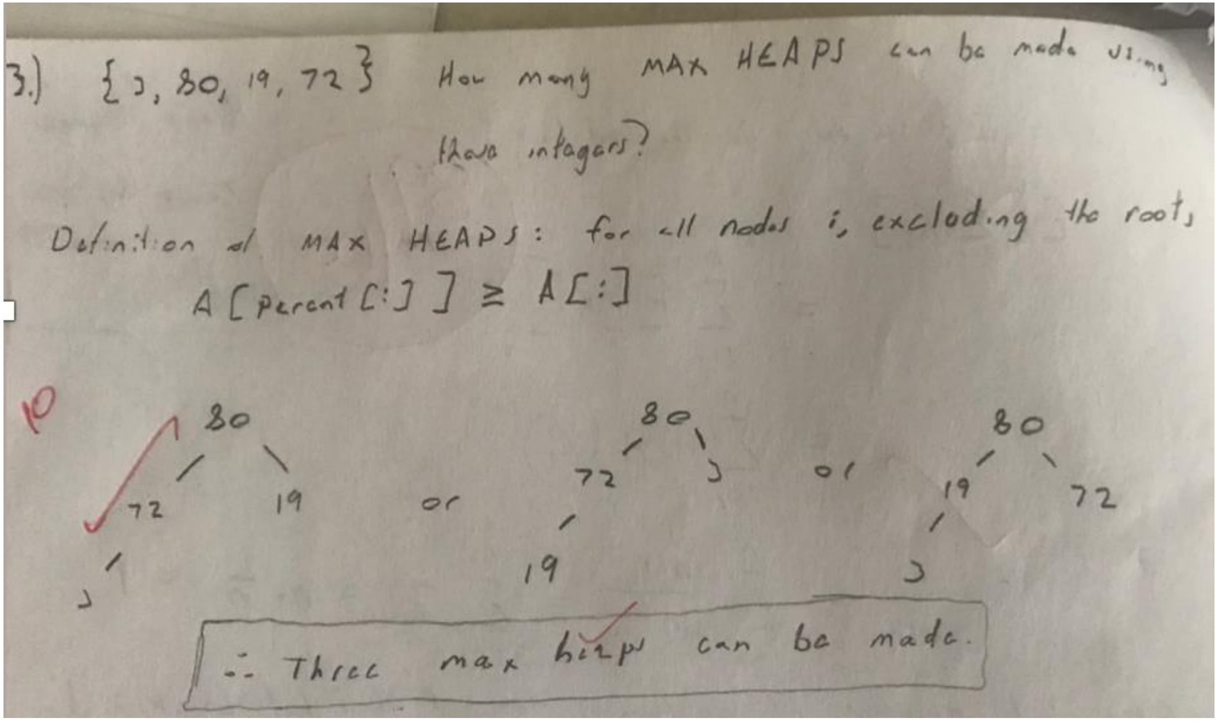
$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$ where $E[X_{ij}] = 1/2$ and $Pr(i<j) = \frac{1}{2}$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} = \binom{n}{2} \cdot \left(\frac{1}{2}\right) = \frac{n(n-1)}{2} * \frac{1}{2}$$

Prof. Benyuan Liu
Fall - 2017

3. Credits: Renan Campos



3.) {3, 80, 19, 72} How many MAX HEAPS can be made using these integers?

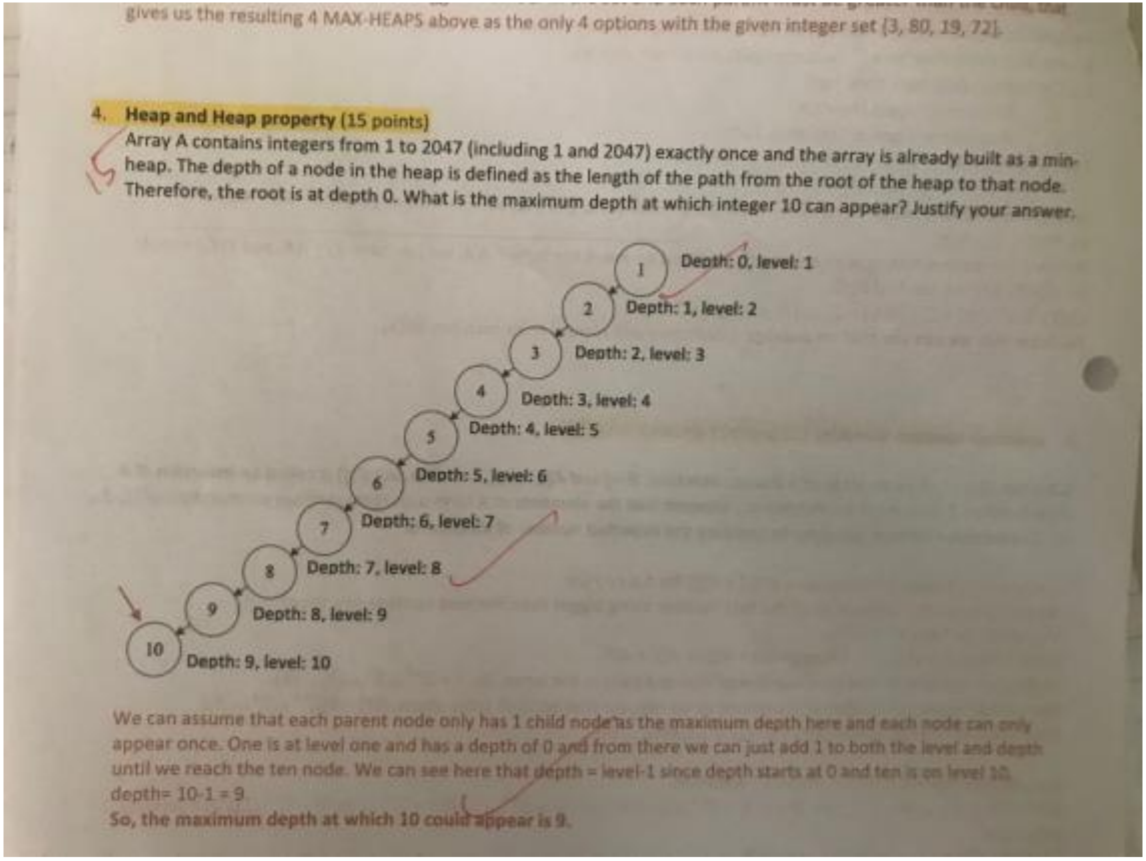
Definition of MAX HEAPS: for all nodes i, excluding the root,
A[Parent[i]] ≥ A[i]

10

80
72
19
3
or
80
72
3
19
or
80
19
72
3

∴ Three max heaps can be made.

4. Credits: Taylor M.Langlois



gives us the resulting 4 MAX-HEAPS above as the only 4 options with the given integer set {3, 80, 19, 72}.

4. **Heap and Heap property** (15 points)

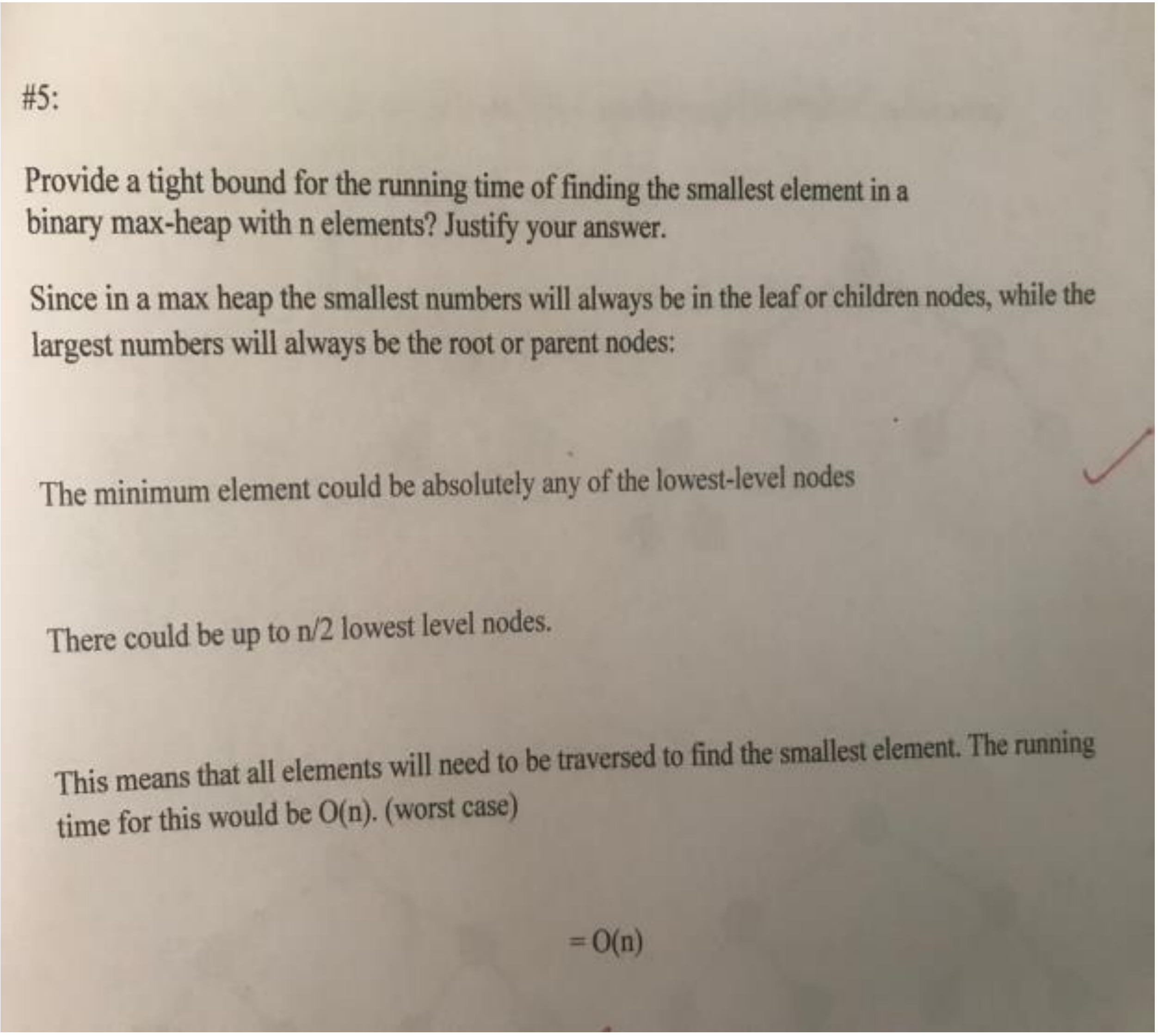
15

Array A contains integers from 1 to 2047 (including 1 and 2047) exactly once and the array is already built as a min-heap. The depth of a node in the heap is defined as the length of the path from the root of the heap to that node. Therefore, the root is at depth 0. What is the maximum depth at which integer 10 can appear? Justify your answer.

We can assume that each parent node only has 1 child node as the maximum depth here and each node can only appear once. One is at level one and has a depth of 0 and from there we can just add 1 to both the level and depth until we reach the ten node. We can see here that depth = level-1 since depth starts at 0 and ten is on level 10.
depth= 10-1 = 9

So, the maximum depth at which 10 could appear is 9.

5. Credits: Ryan Cauble



#5:

Provide a tight bound for the running time of finding the smallest element in a binary max-heap with n elements? Justify your answer.

Since in a max heap the smallest numbers will always be in the leaf or children nodes, while the largest numbers will always be the root or parent nodes:

The minimum element could be absolutely any of the lowest-level nodes ✓

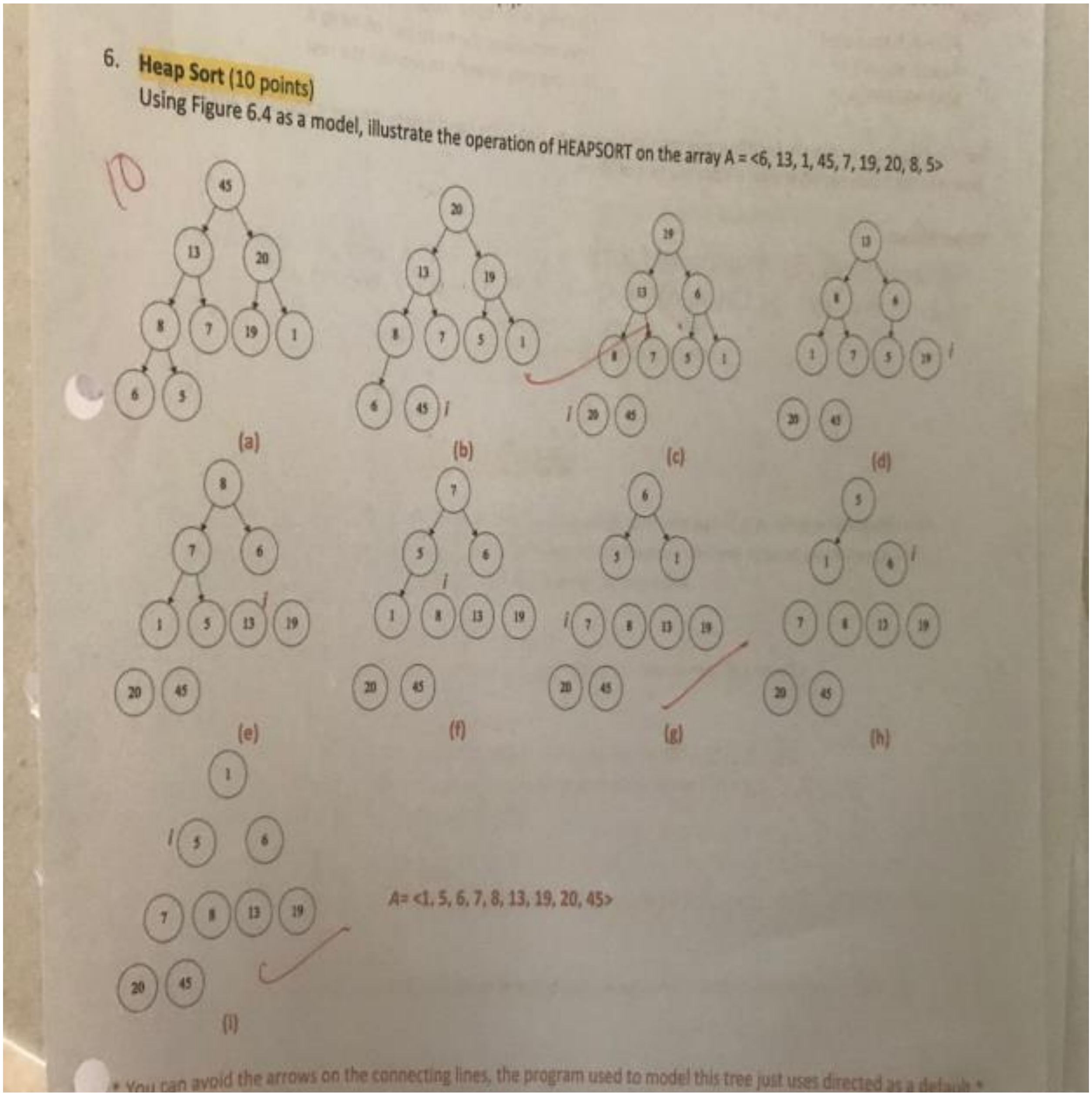
There could be up to n/2 lowest level nodes.

This means that all elements will need to be traversed to find the smallest element. The running time for this would be O(n). (worst case)

= O(n)

Prof. Benyuan Liu
Fall - 2017

6.Credits: Taylor M.Langlois



6. **Heap Sort (10 points)**
Using Figure 6.4 as a model, illustrate the operation of HEAPSORT on the array A = <6, 13, 1, 45, 7, 19, 20, 8, 5>

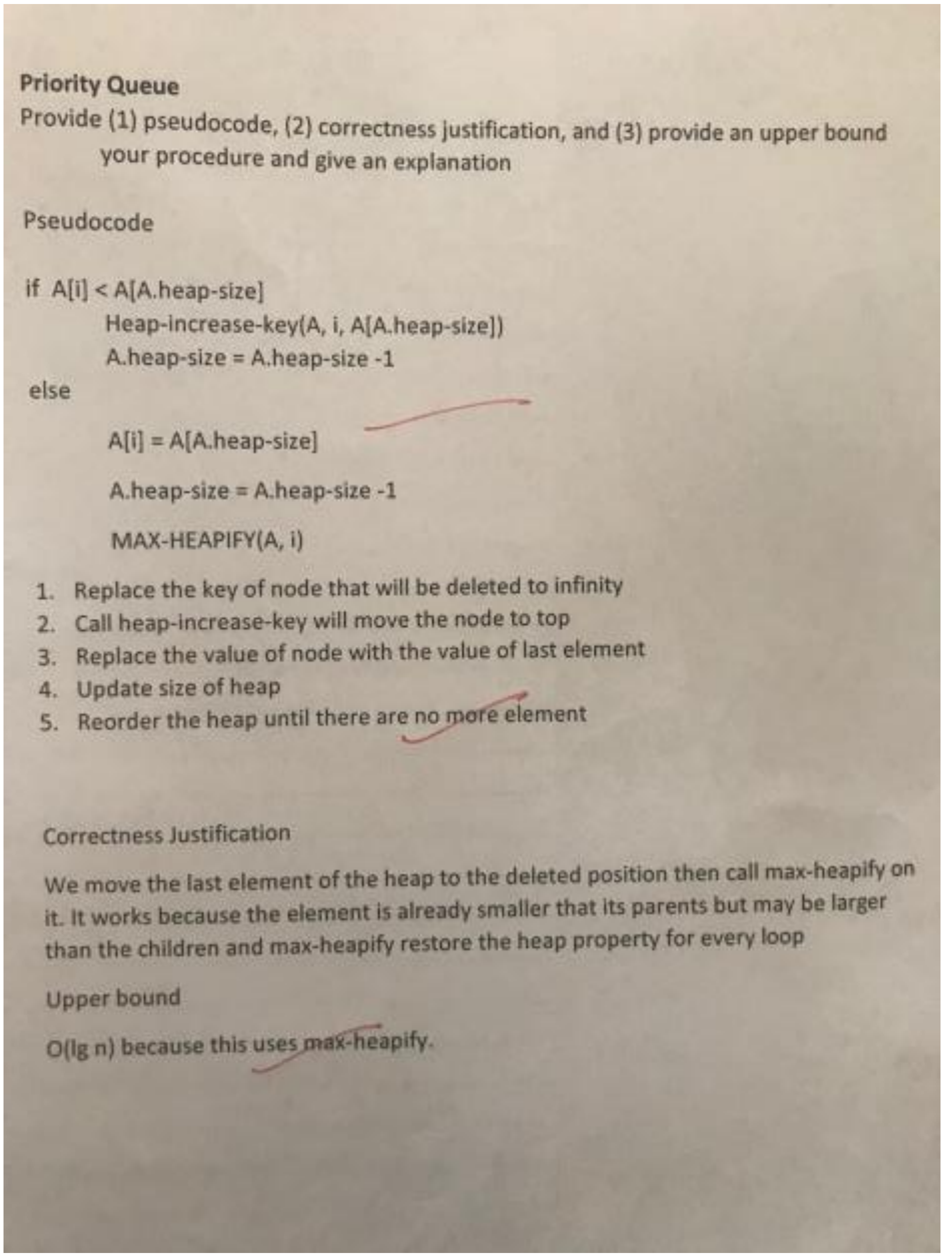
10

A= <1, 5, 6, 7, 8, 13, 19, 20, 45>

* You can avoid the arrows on the connecting lines, the program used to model this tree just uses directed as a default *

Prof. Benyuan Liu
Fall - 2017

7. Credits: Minh Nguyen



Priority Queue

Provide (1) pseudocode, (2) correctness justification, and (3) provide an upper bound your procedure and give an explanation

Pseudocode

```
if  A[i] < A[A.heap-size]
        Heap-increase-key(A, i, A[A.heap-size])
        A.heap-size = A.heap-size -1
else
        A[i] = A[A.heap-size]
        A.heap-size = A.heap-size -1
        MAX-HEAPIFY(A, i)
```

1. Replace the key of node that will be deleted to infinity
2. Call heap-increase-key will move the node to top
3. Replace the value of node with the value of last element
4. Update size of heap
5. Reorder the heap until there are no more element

Correctness Justification

We move the last element of the heap to the deleted position then call max-heapify on it. It works because the element is already smaller that its parents but may be larger than the children and max-heapify restore the heap property for every loop

Upper bound

O(lg n) because this uses max-heapify.