Homework 3 Solutions

1. Indicator Random Variables

① Indicator Random Variables (10 points)

10

Let X_i be the indicator random variable for the event "the ith customer gets their hat back."

Given a sample space S with $n!$ permutations of outcomes where n customers are randomly assigned one of n hats, it can be shown that the Pr(the ith customer gets their hat back) is equal to $\frac{1}{n}$.

And since $E[X_A] = Pr(A)$

We have, by Linearity of Expectations,

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

2. Indicator Random Variables

2) let X_{ij} be indicator random variable.

10

$X_{ij} = I\{A[i] > A[j]\}$ for $1 \le i < j \le n$

The probability of getting first number is bigger than second so $(X_{ij} = 1) = 1/2$.

Using Lemma, $E[X_{ij}] = 1/2$ since $A[i] > A[j]$ or $A[j] > A[i]$.

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

$\rightarrow E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \rightarrow$ replace $E[X_{ij}]$ with $1/2$.

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1/2$$

$$= \sum_{i=1}^{n-1} (n-(j)+1)\, 1/2$$

$$= \sum_{i=1}^{n-1} (n-(i+1)+1)\, 1/2$$

$$= \sum_{i=1}^{n-1} \frac{(n-i)}{2}$$

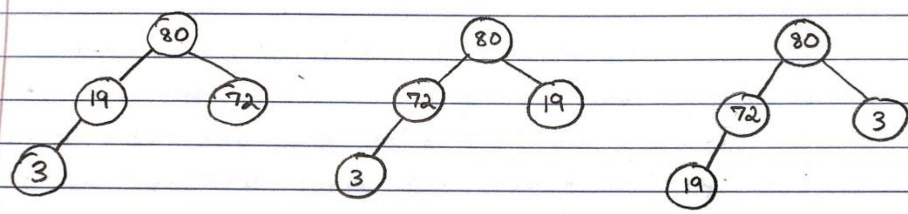
$$= \sum_{i=1}^{n-1} \frac{n}{2} - \sum_{i=1}^{n-1} \frac{i}{2}$$

$$= \frac{n(n-1)}{2} - \frac{n(n-1)}{4} \rightarrow E[X] = \frac{n(n-1)}{4}$$

3. Heaps

③ $\{3, 80, 19, 72\}$

10



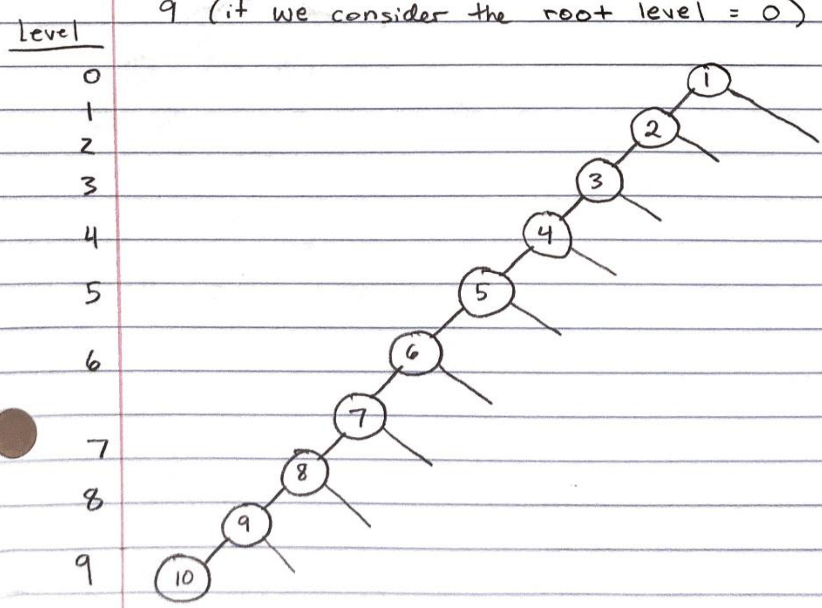
Only $\boxed{\text{3 max-heaps}}$ can be made from the set of integers. All other configurations do not satisfy the max-heap property.

4. Heap and Heap Property

15

④ Observe that on a min-heap, an element x at the i^{th} level has $i-1$ ancestors. By the property of min-heaps, these $i-1$ ancestors are guaranteed to be less than x. This implies x cannot be among the smallest $i-1$ elements of the heap. Using this property, we can conclude that the k^{th} smallest element cannot be deeper than the k^{th} level of the heap.

Thus in min-heap consisting of the integers $\{1 \ldots 2047\}$, the maximum depth that the integer 10 can appear is 9 (if we consider the root level = 0)



5. Heap and Heap Property
   The smallest element resides on one of the leaves. About n/2 of the nodes are the leaves as we discussed in the class. Intuitively, we'll need O(n/2) time to find the smallest element (by comparison). So the worst running time is O(n).

6. Heap Sort
   Follow the algorithm and figure in the textbook

7. Priority Queue
   runs in O(lgn) time

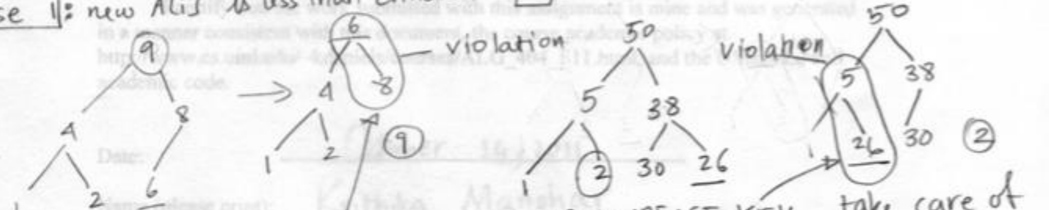
(6) Exercise 6.5-8, p. 166
Assume index i is not out-of-bounds.

Pseudocode

```
HEAP-DELETE (A, i)
1  exchange A[i] with A[A.heapsize]
2  deleted = A[A.heap-size]                    O(1)
3  A.heap-size = A.heap-size-1
4  if deleted > A[i]                         → O(lg n)
5      MAX-HEAPIFY (A, i)
6  else HEAP-INCREASE-KEY (A, i, A[i])     or → O(lg n)
```

Correctness: The last node in the heap is moved to i and the element to be deleted is moved to the end, then heap-size is decremented, essentially removing A[i]. However, there are two cases to consider when swapping these nodes in order to maintain a max-heap

Case 1: new A[i] is less than children   2: A[i] needs to travel up the heap.



The calls to MAX-HEAPIFY and HEAP-INCREASE-KEY take care of both violations of the max-heap property if they occur and maintain the max-heap property when finished executing.

Mechanical correctness: This algorithm never tries to access an element of heap A that is out-of-bounds, and the deleted element is pushed outside the heap (line 3).

Because HEAP-DELETE calls either MAX-HEAPIFY (O(lg n)) or HEAP-INCREASE-KEY (O(lg n)) if deleting node i results in a violation of the max-heap-property, and all other steps are constant (O(1)) work, then HEAP-DELETE's runtime is upper bounded by lg n: $\boxed{T(n) = O(\lg n)}$