Recall that we defined a priority queue $S$ together with the following operations (each of which runs in time $\log n$ except the second which runs in time 1).

- **Insert** $(S, x)$ which inserts $x$ into $S$.
- **Maximum** $(S)$ which returns the max element in $S$.
- **Extract-Max** $(S)$ which returns the max element and removes it from $S$.
- **Increase-Key** $(S, i, x)$ which increases element $i$'s key to $x$.

For the purpose of this problem we will call an algorithm “naive” if it only acts on $S$ through these function calls.

Now assume the priority queue is implemented as a max-heap and that you are also given access to the functions (the first four of which run in time 1 and the last in time $\log n$).

- **Parent** $(i)$ which returns the parent of the $i$-th element.
- **Left** $(i)$ which returns the left child of the $i$-th element.
- **Right** $(i)$ which returns the right child of the $i$-th element.
- **Remove** $(A)$ which removes the right most leaf of $A$.
- **Max-Heapify** $(A, i)$ which lets the $i$-th element “float” down the heap.

For the purpose of this problem we will call an algorithm “intelligent” if it additionally has access to these 4 functions.

(a) (5 Points) Suppose you would like to find the second max in a heap (i.e. the second largest element of $S$). One naive approach might be to run the following code:

```
1 FIND-2NDMax(S)
2   a = Extract-Max(S)
3   b = Maximum(S)
4   INSERT(S, a)
5 Return b
```

However this runs in time $1 + 2\log n$. Your job is to find an “intelligent” solution which takes time close to 1. Give pseudocode and formally analyze the correctness and runtime of your algorithm.

Solution: ****************** INSERT YOUR SOLUTION HERE ******************
(b) (5 Points) Now suppose you would like to extract the second max. Give a “naive” solution (similar to the example in part [a]) to this algorithm. Argue its correctness and analyze its runtime as precisely as possible.

Solution: **************** INSERT YOUR SOLUTION HERE ****************

(c) (5 Points) Now give an “intelligent” implementation of EXTRACT-2NDMAX(S) that runs in time close to \( \log n \). As usual argue correctness and analyze the runtime. How does this solution compare with the one from part (b)? (Hint: Consider using MAX-HEAPIFY.)

Solution: **************** INSERT YOUR SOLUTION HERE ****************

**** INSERT YOUR NAME HERE ****, Homework 4, Problem 1, Page 2
Consider the problem of merging $k$ sorted arrays $A_1, \ldots, A_k$ of size $n/k$ each, where $k \geq 2$.

(a) (8 points) Using a min-heap in a clever way, give $O(n \log k)$-time algorithm to solve this problem. Write the pseudocode of your algorithm using procedures Build-Heap, Extract-Min and Insert.

Solution: ***************** INSERT YOUR SOLUTION HERE ************

(b) (8 points) Let the number of arrays $k = 2$. Assume all $n$ numbers are distinct. Using the decision tree method and the fact (which you can assume without proof) that $\binom{n}{n/2} \approx \frac{2^n}{\sqrt{n!}}$, show that the number of comparisons for any comparison-based 2-way merging is at least $n - O(\log n)$.

(Hint: Start with proving that the number of possible leaves of the tree is equal to the number of ways to partition an $n$ element array into 2 sorted lists of size $n/2$, and then compute the latter number.)

Solution: ***************** INSERT YOUR SOLUTION HERE ************

(c*) Extra Credit: Show that any correct comparison-based 2-way merging algorithm must compare any two consecutive elements $a_1$ and $a_2$ in merged array $B$, where $a_1 \in A_1$ and $a_2 \in A_2$. Use this fact to construct an instance of 2-way merging which requires at least $n - 1$ comparisons, improving your bound of part (b).

Solution: ***************** INSERT YOUR SOLUTION HERE ************

(d**) Extra Credit: Show that for general $k$, any comparison-based $k$-way merging much take $\Omega(n \log k)$ comparisons, showing that your solution to part (a) is asymptotically optimal.

(Hint: You can either try to extend part (b) (easier) or part (c) from $k = 2$ to general $k$. Beware that calculations might get messy...)

Solution: ***************** INSERT YOUR SOLUTION HERE ************
You receive a sales call from a new start-up called MYPD (which stands for “Manage Your Priorities... Differently”). The MYPD agent tells you that they just developed a ground-breaking comparison-based priority queue. This queue implements $\text{Insert}$ in time $\log_2(\sqrt{n})$ and $\text{Extract\_max}$ in time $\sqrt{\log_2 n}$. Explain to the agent that the company can soon be sued by its competitors because either (1) the queue is not comparison-based; or (2) the queue implementation is not correct; or (3) the running time they claim cannot be so good. To put differently, no such comparison-based priority queue can exist.

(Hint: You can use the following Sterling’s approximation: $n! \approx \left(\frac{n}{e}\right)^n$ (where $e$ is euler’s constant))

Solution: ******************* INSERT YOUR SOLUTION HERE ******************* □