Problem Set 2

1. Create a min-heap by inserting the following elements in order:

   $$11, 15, 3, 5, 8, 9, 10, 100, 20, 1, 1, 7, 13, 25, 31, 60, 2, 21$$

   Draw the resulting heap as a tree and as an array. Then draw the heap that you get after removing the minimum element.

2. (a) Given an array A of n integers, find the median of A in $O(n \log n)$ time.[1]

   (b) Give an $O(n \log n)$ algorithm that finds two distinct elements of A which are as close as possible – that is with $i \neq j$ and $|A[i] - A[j]|$ as small as possible. You just need to give the elements, not their indices.

   (c) Given n points $(x_1, y_1), \ldots (x_n, y_n)$ in the plane, find the maximum slope of a line that passes through two of the points in $O(n \log n)$ time. If there is a vertical line then ∞ should be your output.

3. A sequence of n alien invaders is lined up along a hallway of the Enterprise with each invader at an integer position between 0 and $n - 1$ inclusive. You have in your possession k explosive ordinances which you're allowed to place anywhere along the hallway and must detonate simultaneously. Each invader has some resilience $R[i]$, which means the alien at position i is killed if it's within the blast zone of at least $R[i]$ different explosives[2].

   Due to a recent initiative to make war more eco-friendly, you've replaced all your munitions with electric versions. Unfortunately your k explosives still need to be charged. They're currently all on the charger, and will all have a blast zone of length t after t minutes of

[1] $O(n)$ is possible but hard with a deterministic algorithm.
[2] The blast zone of each explosive is a closed interval.

charging. Give an algorithm to find the shortest time you can wait before launching an attack which kills all the invaders. You're guaranteed that $R[i] \leq k$ for all i (otherwise it's not possible to kill all the invaders). Your algorithm should run in $O(n \log n)$ time [3].

4. For this problem you'll use a heap to implement a version of a priority queue. Follow the instructions and answer the questions in the notebook. You should only have to add a small amount of code. If you find yourself adding a lot of code, you're probably doing more work than you need to.

1 Other problems that you don't have to turn in.

- The goal of this problem is to show that any (deterministic) comparison-based sorting algorithm requires at least $\Omega(n \log n)$ comparisons in the worst case. Suppose that we're given an array $A[0], \ldots, A[n-1]$ containing the numbers $1, 2, \ldots, N$ in some order. We're not allowed to see the entries of A directly. What we can do is pick two indices $i \neq j$ and make a comparison query between $A[i]$ and $A[j]$ which will result in a response of either '$<$' if $A[i] < A[j]$ or '$>$' if $A[i] > A[j]$. Our goal is to output a permutation $i_1, \ldots, i_n$ of $(0, 1, 2, \ldots, n-1)$ such that

$$A[i_1] < A[i_2] < \cdots < A[i_n].$$

  1. Suppose that we have an algorithm making T comparison queries total[4]. How many possible sequences of responses are there?
  2. Show that for any correct algorithm there must be at least $n!$ possible sequences of responses.
  3. Conclude that $T \geq \Omega(\log(n!))$.
  4. Show that $\log(n!)$ is $\Omega(n \log n)$, thereby showing that T is at least $\Omega(n \log n)$.

- Let $F : \mathbb{Z} \to \mathbb{R}$ be a real-valued, n-periodic function on the integers meaning that $F(x) = F(x+n)$ for all integers x (you can think of this as a function defined on $\{0, \ldots n-1\}$ but with wrap-around). We say that x is a local maximum of F if $F(x) \geq F(x-1)$ and $F(x) \geq F(x+1)$.

[3] I'll eventually post some hints on Campuswire, but try it on your own first.

[4] We're free to assume exactly T comparison queries. Any algorithm which makes fewer queries could perform additional "dummy" queries to get up to T.

Give an algorithm to find a local maximum of F using only $O(\log n)$ function evaluations.