1 Median minimizes L^1 norm

This is how I'd do this problem, but this level of detail isn't required for full credit on the problem.

The minimum of

$$f(c) = \sum_{i=1}^{n} |x_i - c|$$

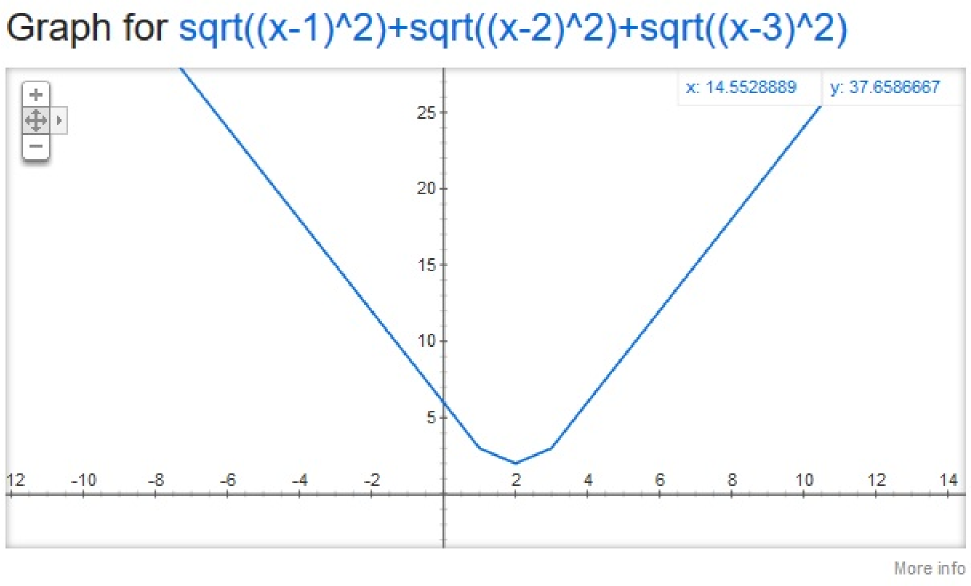
happens at $c = \text{median}\{x_1, \dots, x_n\}$. When n is odd, the value for c is uniquely determined. If n is even, we can use either median, or any point in between them.

The median can be computed in $O(n)$ expected time using the algorithm similar to quicksort: see pages 10-12 of this .pdf: `https://people.eecs.berkeley.edu/~vazirani/algorithms/chap2.pdf`

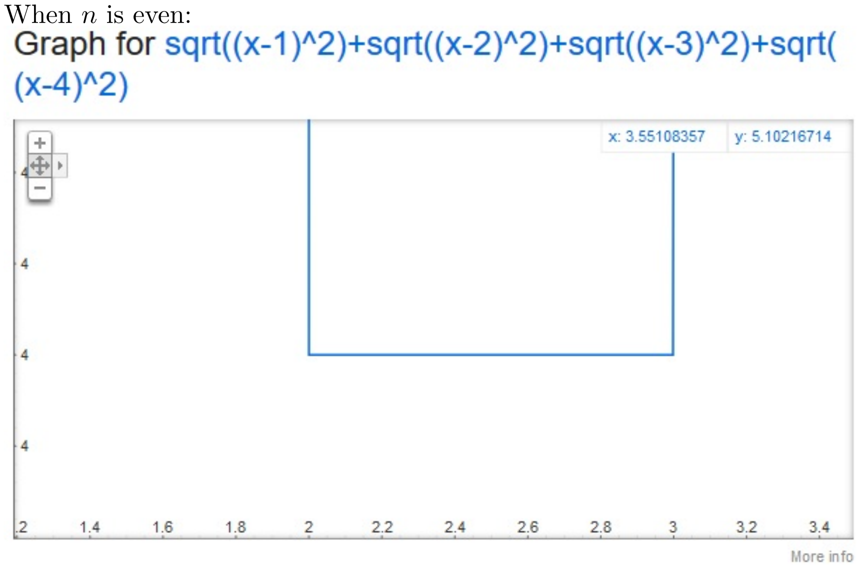
In fact, this algorithm can be massaged to work in guaranteed $O(n)$ time. See here: `http://web.mit.edu/~neboat/www/6.046-fa09/rec3.pdf`

Some motivating graphs.

When n is odd:



When n is even:



Proof of correctness.

Write $|x_i - c|$ as $\sqrt{(x_i - c)^2} = \sqrt{(c - x_i)^2}$. Compute the derivative (with respect to c) as $\sum_{i=1}^{n} \frac{c-x_i}{|c-x_i|}$. We want to find critical points: $f'(c)$ is never zero, so the only critical points are when the derivative is undefined. This happens when the argument inside the absolute value is 0 (recall $|x - a|$ is always continuous, but not differentiable at a — just note that $\lim_{x \to a^+} \frac{x-a}{|x-a|} = 1$, but $\lim_{x \to a^-} \frac{x-a}{|x-a|} = -1$).

First suppose n is odd, so the median is well-defined and unique. Consider one term of the derivative $\frac{c-x_i}{|c-x_i|}$. This is undefined at x_i, but is ± 1 depending on which side you're on. Now at any point on the open subintervals where f is differentiable to the right of the median, $c - x_i > 0$ for more of the terms than < 0, so the derivative is positive and f is thus increasing. For any point to the left of the median (where f is differentiable), $c - x_i < 0$ for more of the terms than > 0, so the derivative is negative and thus f is decreasing. So f is always increasing on (μ, ∞) and always decreasing on $(-\infty, \mu)$, where μ is the median, so the median minimizes the L^1 norm.

Now suppose n is even: same argument applies. Let the two middle elements be $\mu_1, \mu_2 (\mu_1 < \mu_2)$. To the right of μ_2, more terms satisfy $c - x_i > 0$ than < 0, so increasing. To the left of μ_1, more terms satisfy $c - x_i < 0$ than > 0, so decreasing.

Between μ_1 and μ_2, the slope is 0 (i.e., the function is constant between them). To see this, since we have an even number of terms excluding the two medians, their contributions to the derivative cancel out. So it suffices to show that for $a < b$, $|x - a| + |x - b| = b - a$ (and thus the derivative is 0 between the two medians). This means we can choose any point in the closed interval $[\mu_1, \mu_2]$. Simply interpret this as $|x - a|$ being the distance from a to x, and $|x - b| = |b - x|$ as the distance from x to b. Since $a < b$, and we're on a horizontal line, equality holds.

2 k-selection problem from midterm

How can we find the k smallest elements of an array efficiently? **Note**: the k elements we find don't need to be in sorted order!

Solution 1 (bad): Sort the array in $O(n \log n)$ time and grab the first k elements.

Solution 2 (decent): Use the heap method gone over in a previous recitation (see `pollard_rho_092917.pdf`) for $O(n \log k)$ time. Then this (max) heap (which contains k elements) is precisely the set of the k smallest elements of the original array.

Solution 3 (good): Use the quicksort-like method to find the kth smallest element. Note that because of the partitions used, every element to the left of the kth smallest will be smaller and everything to the right will be larger at this point. So we can just grab the k smallest elements of this array. This takes *expected* $O(n)$ time. If this algorithm is not clear to you, create a data set and run through the calculations like we did in discussion.

Note that (see link above in the median problem) there is a version of this algorithm that runs in *worst-case* $O(n)$ time if we care about worst-case time for some reason. (In practice, the expected-time algorithm is much better: the constants hidden in the deterministic $O(n)$ time are horrendous.) It's not important to know the details of the worst-case $O(n)$ algorithm, just be aware that it exists.

(Use **Solution 2** if you're actually asked to find the median, kth smallest, etc. in an interview! Coding the partition stuff, and getting the recursion right, on the fly will be a disaster, but calling the PriorityQueue Java class (or whatever language) will give you a nice 10-line solution.)

3 Inserting into a binary search tree

`http://www.algolist.net/Data_structures/Binary_search_tree/Insertion`

4 Deleting from a binary search tree

`http://www.algolist.net/Data_structures/Binary_search_tree/Removal`