HW 3

David Kravets

1. we pick 3/n or 3 out of n numbers. The chance that one of these three (we choose the optimal middle one) is in half the array should be $1-(1/2)^3 = 7/8$ or greater, The expected number of picks would be 1.14.

2.1 we can take the middle number in the array (ie A[n/2) and check if i is greater or less than the middle number. We then do the same search on the half of the array that i falls under (the smaller half or larger half). Each time, we check the middle number, and if it isn't equal to i we take the half of the array we think i is in. do this until i is found, or the array is smaller than 3.

* This is binary search.

2.2 Since the array is sorted, we can pick a number. If that number is larger/smaller than the number we are looking for, we can ignore the left or right part of the array.

This algorithm will run until it finds i, or runs out of numbers that satisfy the search condition. Since the search condition only ignores numbers that are greater or less than i, in one given cycle, i will always be found if present.

2.3 $f(n) = f(\frac{n}{2}) + O(1)$ while $x \geq 1$

$= f(\frac{n}{2^k}) + 1$

$n = 2^k \Rightarrow f(n) = f(1) + k = 1 + \log n$

$T(n) = O \log n$