Test 2 Study Guide

- Master Theorem
  - $T(n) = 2T(\frac{n}{2}) + 1$
    - a = 2, b = 2, d = 0
    - $T(n) \in \Theta(n)$
  - $T(n) = 2T(\frac{n}{2}) + n$
    - a = 2, b = 2, d = 1
    - $T(n) \in \Theta(n \log n)$
  - $T(n) = 7T(\frac{n}{2}) + n^2$
    - a = 7, b = 2, d = 2
    - $T(n) \in \Theta(n^{\log_2 7})$
  - $T(n) = 2T(\frac{n}{3}) + n$
    - a = 2, b = 3, d = 1
    - $T(n) \in \Theta(n)$
  - $T(n) = 8T(\frac{n}{2}) + n^2$
    - a = 8, b = 2, d = 2
    - $T(n) \in \Theta(n^3)$
- Algorithms
  - Given an integer array of size n, find the maximum value of the sum of elements of a subarray where no two elements of the subarray are adjacent to each other. Design a Θ(n) algorithm using a decrease and conquer approach. Your algorithm just needs to find the maximum value of the sum. It does not need to identify the elements of the subarray.
    - Size 1 is just the element, size 2 is just the max between the two elements
    - If we know how to solve size k, how do we solve size k+1?
      - Three cases
        - If the new element is not part of the solution. the solution for k and k+1 will be the same
        - If it is included in the solution, the kth element cannot be. So the optimal solution will be the optimal solution for k-1 plus the element at k+1
        - If we have negative numbers, A[k+1] might be the max just by itself

        - The total solution then is max{$v_1, v_0 + A[k+1], A[k+1]$}
        - Alg MaxNonAdjacent (A[0...n-1])
            - $v_0 = A[0]$
            - $v_1 = \max(v_0, A[1])$
            - for i = 2 to n-1
                - v = max{$v_1, v_0 + A[i], A[i]$}
            - return v
- Design an algorithm to compute the integer part of the square root of a non-negative integer n. For example, for input n = 5, the output is 2. You can only use the four basic algebraic operations (+, −, ×, /) and the floor() operation, i.e., rounded down. Your algorithm should have a time complexity of O(log n).
    - alg sqrt(n)
        - l = 0
        - r = n
        - while true
            - x = floor($\frac{l+r}{2}$)
            - if x*x <= n
                - if (x+1)*(x+1) >= n
                    - return x
                - else
                    - left = x+1
            - else
                - r = x
- Given an array A[0 … n − 1] of n distinctive integers, reorder it such that A[0] < A[1] > A[2] < A[3] … … . • Design an O(n log n) time and Θ(1) space algorithm. Partial credits are given to solutions that produce a correct order but do not meet the time or space complexity requirements. • (3 Bonus Points) Design a Θ(n) time and Θ(1) space algorithm.
    - Presort then swap elements 1 and 2, 3 and 4, and so on, that will achieve nlogn
    - For linear
        - Alg WiggleSort(A)
            - up = true
            - for i = 0 to A.len - 2
                - if up

        - if A[i] > A[i+1]
            - swap(A[i], A[i+1])
        - up = false
    - else
        - if A[i] < A[i+1]
            - swap(A[i], A[i+1])
        - up = true
- OR just use quickselect to find median, then do l r l r l r...