Answer the following questions and submit to D2L by deadline. This assignment is about Divide-and-Conquer Chapter 5 of your textbook.

1. Your textbook, ***Exercise 5.1, question number 2.a***.

   a. Write pseudocode for a divide-and-conquer algorithm for finding values of both the largest and smallest elements in an array of n numbers.

```
def minMax(arr, lft, rgt, min, max):
      if lft==rgt:
            if arr[lft] > max:
                  max = arr[lft]
            if arr[lft] < min:
                  min = arr[lft]
            return (min,max)
      else:
            (min,max) = minMax(arr, lft, math.floor((lft+rgt)/2), min, max)
            (min,max) = minMax(arr, math.floor((lft+rgt)/2)+1, rgt, min, max)
            return (min,max)

print(minMax([4,1,9,0,4,4], 0, 5, 10, -10))
#This function's second and third argument are the first and last index of the
array, and the functions forth and fifth arguments serve only to initialize min
and max and can take any values
```

2. Your textbook, ***Exercise 5.1, question 5***.

   5. Find the order of growth for solutions of the following recurrences.

   a. T (n) = 4T (n/2) + n, T (1) = 1

   b. T (n) = 4T (n/2) + n^2, T (1) = 1

   c. T (n) = 4T (n/2) + n^3, T (1) = 1

   a = 4, b = 2

   1) for every literal

   $n^{(\log_b a)} = 1^{(\log_2 4)} = 1^{(2)} = 1$

   2) for every literal

   $1^x = 1 \Rightarrow f(n) == n^{(\log_b a)} \Rightarrow O(\log n)$

3. **Reading Comprehension**: I gave you a handout about Quicksort. You need to study it on your own and understand the algorithm. Try to use some examples to make sense of the Quicksort algorithm described in Section 5.2 of your textbook. Once understood Quicksort, answer ***Exercises 5.2, question 7***.

   a. Estimate how many times faster quicksort will sort an array of one million random numbers than insertion sort.

   - In the worst case both insertion sort and quick sort bigO(n^2) so there is not a performance difference.

   - In the average case though, quick sort performs at $n\log_2 n$ while selection sort is still n^2 so:

   f(1000000) = $n\log_2 n$ = 19,931,568 = approx. 2x10^7

   = n^2 = 1x10^12

   insertionSort time / quickSort time = 50,000 faster

   b. True or false: For every n > 1, there are n-element arrays that are sorted faster by insertion sort than by quicksort? false

4. In class we discussed Closest-pair and Convex-Hull problems. Review the lecture and handout I gave, and answer ***Exercises 5.5, question 12***.

   Shortest path around - There is a fenced area in the two-dimensional Euclidean plane in the shape of a convex polygon with vertices at points p1(x1, y1), p2(x2, y2), . . . , pn(xn, yn) (not necessarily in this order). There are two more points, a(xa, ya) and b(xb, yb) such that xa < min{x1, x2, . . . , xn} and xb > max{x1, x2, . . . , xn}. Design a reasonably efficient algorithm for computing the length of the shortest path between a and b. [ORo98]

```
def distance(points):
    a =  min(points, key = lambda point: point.x)
    index = points.index(a)

    l = index
    result = []
    result.append(a)
    while (True):
        q = (l + 1) % len(points)
        for i in range(len(points)):
            if i == l:
                continue
            d = direction(points[l], points[i], points[q])
            if d > 0 or (d == 0 and distance_sq(points[i], points[l]) >
distance_sq(points[q], points[l])):
                q = i
        l = q
        if l == index:
            break
        result.append(points[q])
    return result
```

algorithm:

given 2 points are close enough to polygon (PO)

1) calc point A distance to all lines in PO return shortest, call it intersectA

2) calc point B distance to all lines in PO return shortest, call it intersectB

3) calc right and left length of intersectA to intersectB using points in PO return shortest call it POshortest

4) final path is (A, intersectA, POshortest, intersectB, B)