1. (a) The idea for finding the maximum value using the divide-conquer stradegy would be to use recursion to find the largest value in the left half, then find the largest value in the right half, and then return the larger of the two numbers.
Code:
int maxValue(int[] A, int low, int high) {
if(high-low $\leq$0){return A[high];} else {
int left = maxValue(A, low, high/2);
int right = maxValue(A, high/2, high);
if(left $\geq$ right){return left;}
return right;
}}

   (b) The recurrence: $T(n) = 2T(\frac{n}{2}) + c$
Following the picture we have drawn in class several times, we know $T(n) = c * \#$ of items in the tree which we have shown to be n
Therefore, we can conclude that $T(n) = \Theta(n)$

   (c) The two algorithms are asymptotically identical because they both have the same time complexity: $\Theta(n)$

   (d) In practice, the recursion-based implementation is slower because of the memory required to remember each previous step during the recursion's run.

2. (a) An idea would be to modify the binary search algorithm and replace the equality check with array[mid] == mid instead of array[mid] == key
Code:
int prodFinder(int[] A, int low, int high){
if(low > high){return -1;}
int mid = (low+high)/2;
if(A[mid] == mid){return mid;}
else if(a[mid] < mid){
return prodFinder(A, mid+1, high);}
else{return prodFinder(A, low, mid-1);}
}

   (b) In theory, this algorithm would have the exact same time complexity as binary search: $O(\log n)$.

3. (a) Radix sort

      i. Sort data without comparisons by using individual digits that share the same position and value.
      ii. The algorithm takes the most significant part of each number and groups elements with the same digit into one bucket. Then, each bucket is sorted recursively, starting with the next digit to

the right. Finally, all of the buckets are concatenated together in order.

iii. $O(n)$

iv. This algorithm will spend $O(n)$ in ANY case.

v. https://en.wikipedia.org/wiki/Radix_sort#Recursion

(b) Karatsuba algorithm

i. The multiplication of two n-digit numbers.

ii. If the two numbers are single digits each, then they can be immediately multiplied. Otherwise, If $n > 1$ then the product of 2 n-digit numbers can be expressed in terms of 3 products of 2 $(n/2)$-digit numbers using a specific placement of each number in an equation discovered by Karatsuba.

iii. $\Theta(n \hat{} \log 3)$

iv. Again, this algorithm will spend $\Theta(n \hat{} \log 3)$ in ANY case.

v. https://en.wikipedia.org/wiki/Karatsuba_algorithm#Recursive_application

(c) Closest Pair algorithm

i. Given a coordinate plane, find the smallest distance between two points.

ii. This algorithm works in almost the exact same way as the maximum subarray algorithm in the sense that it divides the plane into two halves and recursively checks to see if the closest pair of points is on the left half, the right half, or going across the divide. Once all of the smallest point distances have been found, they are compared to each other recursively and in the end the smallest distance is returned.

iii. $O(n \log n)$

iv. Again, this algorithm will spend $O(n \log n)$ in ANY case.

4. $T(n) = 2T(\frac{n}{3}) + n^2 = O(n^2)$
$c = 500$
$n_0 = 1$
$T(n) \leq (500)n^2$ for all $n \geq (1)$
$2T(\frac{n}{3}) + n^2 = \Sigma(\log_2 n)(i = 0)(\frac{2}{9})^i n^2 \leq \Sigma(\infty)(i = 0)(\frac{2}{9})^i n^2 = \frac{1}{1-2/9} n^2 =$
$O(n^2)$
$O(n^2) \leq (500)O(n^2)$

5. The recurrence: $T(n) = 2T(\frac{n}{2}) + n^2$
Following the picture we have drawn in class several times, we know
$T(n) = n^2 * \#$ of rows in the tree which we have shown to be $\log_2 n$
Therefore, we can conclude that $T(n) = n^2 \log_2 n = \Theta(n^2)$