Problem 1

(a) $A = [2, 6, 7, 1, 3, 5, 4]$

(b) j will never reach 6, as the loop is exclusive. It iterates through 0 to 5 for j.

Value of j	**Value of i**	**Array**
0	0	[2,6,7,1,3,5,4]
1	0	[2,6,7,1,3,5,4]
2	0	[2,6,7,1,3,5,4]
3	1	[2,1,7,6,3,5,4]
4	2	[2,1,3,6,7,5,4]
5	2	[2,1,3,6,7,5,4]

(c) $A = [2, 1, 3, 4, 7, 5, 6]$

Problem 2

(a) Taking either the smallest or the largest element of A as the pivot would lead to the worst partition. In this case, at each iteration, we have to compare all of the elements of A to our pivot.

(b) The resulting recurrence would be:

$$T(n) = T(n-1) + \Theta(n)$$

At each iteration, we have to do $(n-1)$ comparisons. After that we would have the same problem to solve but with one less element.

(c) We hypothesize that $T(n) = \Theta(n^2)$:

$$\text{Base case: } T(1) = \Theta(1) \text{ which holds.}$$

$$\text{Inductive hypothesis: } T(k) = \Theta(k^2) \rightarrow T(k) \leq ck^2$$

$$\text{Inductive step: Does } T(k+1) = \Theta((k+1)^2) \text{ hold?}$$

$$T(k+1) \overset{?}{\leq} c(k+1)^2$$

$$\rightarrow T(k) + ck \overset{?}{\leq} ck^2 + 2ck + c$$

$$\rightarrow T(k) \overset{\checkmark}{\leq} ck^2 + ck + c \text{ (By inductive hypothesis)}$$

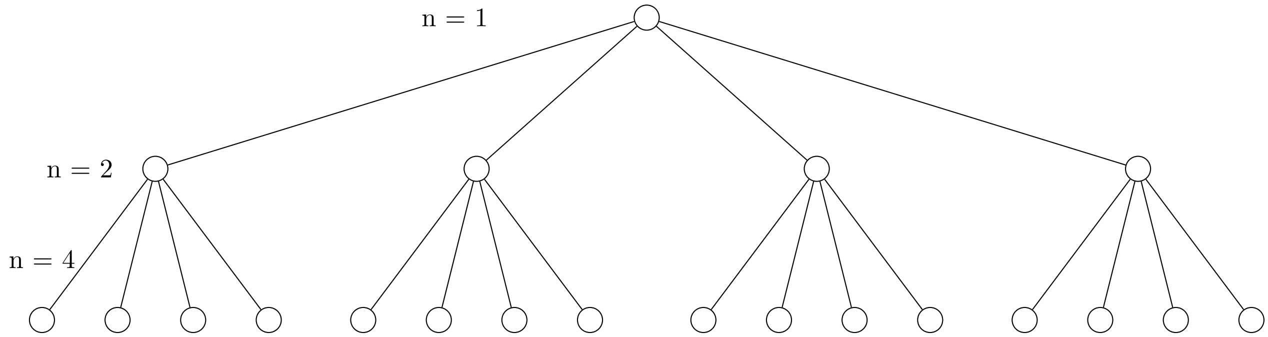
$$\rightarrow T(n) = \Theta(n^2)$$

Alternatively, we can solve the recurrence equation directly as follows:

$$T(n) = T(n-1) + n = T(n-2) + (n-1) + n = ... = 1 + 2 + ... + (n-1) + n = \frac{n(n+1)}{2} = \Theta(n^2)$$

Problem 3

(a)



The recursion tree carries on with a branching factor of 4, meaning each node has four children, as the recursive call is called four times.

At depth d, the amount of work is $n/2^d$.

The tree has $\log n + 1$ levels.

The tree has n^2 leaves (referring only to the bottom layer).

(b) The recurrence relation is $4T(n/2) + cn$, with base cn, as seen by the root of the recursion tree.

To solve for $T(n)$, we will find a pattern for the recurrence relation. Since the function is recurring by a power of 2, we will use $log_2(n)$ for the summation. With the following summation, note how many iterations are being done, as summations are inclusive.

$$\sum_{i=0}^{log_2(n)}$$

n = 1	$log_2(1) = 0$
n = 2	$log_2(2) = 1$
n = 4	$log_2(4) = 2$
n = 8	$log_2(8) = 3$

By establishing how our summation will iterate, we can find the pattern established by the values we determined.

This gives us a summation of:

$$cn \sum_{i=0}^{log_2(n)} 2^i + \Theta(n^2)$$

$cn \sum_{i=0}^{log_2(n)} 2^i$ represents the work of the internal nodes. $\Theta(n^2)$ represents the extra work of the leaves.

Now, we must find the value of the summation. Note that this summation is a geometric series. Therefore, we can follow the formula:

$$\sum_{i=0}^{k} r^i = \frac{r^{k+1} - 1}{r - 1}$$

The following summation has our values substituted into the sum of a geometric series formula:

$$cn \sum_{i=0}^{log_2(n)} 2^i + \Theta(n^2) = cn \cdot \frac{2^{log_2(n)+1} - 1}{2 - 1} + \Theta(n^2)$$

Using logarithm rules:
$2^{\log_2(n)+1} \rightarrow 2 * 2^{\log_2(n)} \rightarrow 2n$

So our final answer is:

$$cn \cdot (2n-1) + \Theta(n^2) = 2cn^2 - cn + \Theta(n^2)$$

This gives us $\Theta(n^2)$.

(c) We first show that $T(n)$ is $O(n^2)$, then we show $T(n)$ is $\Omega(n^2)$. We'll use the substitution method for this.

We choose n^2 for the substitution, so we substitute T(n) as dn^2 in our equation, where d is some new constant.

$$T(n) \leq 4(d(\frac{n}{2})^2) + cn$$

$$T(n) \leq 4d \cdot \frac{n^2}{4} + cn = dn^2 + cn$$

For our cases, we will use $T(n) \leq d \cdot n^2 - cn$ as our guess.

For our base case, we'll take n to be 1.

$$T(1) \leq d(1)^2 - c(1)$$

$$T(1) \leq d - c$$

There exist constants that make this case true, so the base case holds.

Now, assume $T(n) \leq d \cdot n^2 - cn$ holds for all values of n less than n_1, where n_1 is the upper limit of which our guess applies.

As our inductive hypothesis, we assume our guess holds for all values of $n \leq n_1$, where n_1 is the upper value of n in which our guess holds.

By assuming this, we prove that $T(n) \in O(n^2)$.

Similarly, we can assume our guess holds for all values of $n \geq n_2$, where n_2 is the lower value of n in which our guess holds.

Therefore, $T(n) \in \Omega(n^2)$.

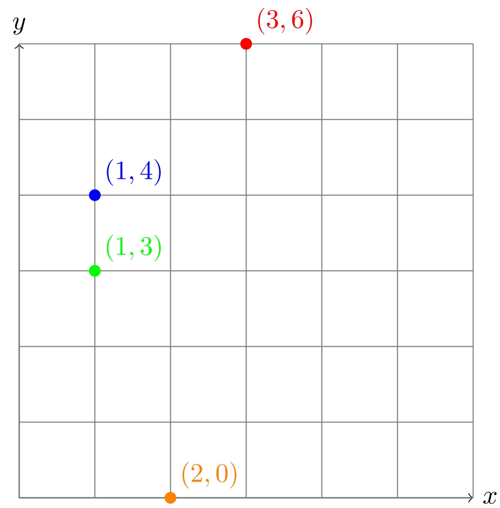
Since we have proved both $T(n) \in \Omega(n^2)$ and $T(n) \in O(n^2)$, we know $T(n) \in \Theta(n^2)$

Problem 4

(a) We can use a brute force algorithm to compute the distance between each two points and keep track of the least distance value. Assuming we have n points to begin with, for each point, we need to do $(n-1)$ comparisons. This leads to $\frac{n(n-1)}{2} = \Theta(n^2)$ comparisons.

(b) This approach fails to consider pairs that might lie across the dividing line, which could have a smaller distance than the minimum found within each half. Consider the example below:

Problem 4 continued on next page...



In this example, m_{top} is the distance between points $(1, 4)$ and $(3, 6)$, while m_{bottom} is the distance between points $(1, 3)$ and $(2, 0)$. Using the approach described leads to the incorrect answer $[(1, 4), (3, 6)]$, as the distance between these two points is less than the distance between the points in the m_{bottom}. The correct answer is $[(1, 4), (1, 3)]$.

(c) The incorrect algorithm performs recursive calls on each half, with each call handling $\frac{n}{2}$ points.The recurrence relation of this algorithm would be:

$$T(n) = 2T(\tfrac{n}{2}) + \Theta(n)$$

Solving this recurrence relation gives a time complexity of $\Theta(nlogn)$.

(d) The number of points on each side is $\frac{n}{2}$, therefore the number of cross-comparisons would be $\frac{n}{2} \times \frac{n}{2} = \frac{n^2}{4}$. The new recurrence relation is:

$$T(n) = 2T(\tfrac{n}{2}) + \Theta(n^2)$$
$$\rightarrow T(n) = 2T(\tfrac{n}{2}) + n^2$$
$$= 2(2T(\tfrac{n}{4}) + \tfrac{n^2}{4}) + n^2 = 2(2T(\tfrac{n}{4})) + \tfrac{n^2}{2} + n^2$$
$$\vdots$$
$$= n^2 \times (1 + \tfrac{1}{2} + \tfrac{1}{4} + ... + \tfrac{1}{2^{\log(n-1)}}) = n^2 \times (2 - \tfrac{2}{n}) = 2n^2 - 2n$$

So the complexity is equal to $\Theta(n^2)$.

(e) We can use the following pseudocode:

Input: List of points P in a 2D plane
Output: Closest pair of points from P and the minimum distance between them
$P_x \leftarrow$ Sort P by x-coordinates
$P_y \leftarrow$ Sort P by y-coordinates
Function ClosestPair(P_x, P_y)
if size of $P_y \leq 3$ **then**
    Find the closest pair by brute force
    return Minimum distance and closest pair

end if
$T \leftarrow$ Top half of P_y
$B \leftarrow$ Bottom half of P_y
$T_x \leftarrow$ Sort T by x-coordinates
$T_y \leftarrow$ Sort T by y-coordinates
$B_x \leftarrow$ Sort B by x-coordinates
$B_y \leftarrow$ Sort B by y-coordinates
$d_T \leftarrow$ ClosestPair(T_x, T_y)
$d_B \leftarrow$ ClosestPair(B_x, B_y)
$d \leftarrow \min(d_T, d_B)$
$S_x \leftarrow$ Points in P_x within distance d of dividing line
foreach point p in S_x **do**
  Check only the next 7 points in S_x for closer pairs
  Update d if a closer pair is found
end foreach
return Minimum distance d and the closest pair of points

This pseudocode efficiently finds the pair of points with the smallest distance. It recursively splits the points into halves, finds the closest pair in each half, and then checks only 7 nearby points across the dividing line to combine the results. You can refer to Section 5.4 of Kleinberg and Tardos for the geometric proof of the 7 neighbors.

(f) As the work done at each level is equal to $\Theta(n)$, the recurrence relation can be written as below:

$$T(n) = 2T(\tfrac{n}{2}) + \Theta(n)$$

Solving this would give the result $T(n) = \Theta(nlogn)$.