Problem Set 1 SOLUTIONS: Algorithm Analysis

Big-Oh

1. We say that $n_0$ and $c$ are witnesses to the fact that $f(n)$ is $O(g(n))$ if for all $n \geq n_0$, it is true that $f(n) \leq c \cdot g(n)$.

   a) If $n_0 = 1$, what is the smallest value of $c$ such that $n_0$ and $c$ are witnesses to the fact that $(n + 2)^2$ is $O(n^2)$?

      Need to find $c$ such that $(n + 2)^2 \leq c \cdot n^2$ for $n \geq 1$.

      For $n = 1$, $8 \leq (c - 1) \cdot 1^2$ or $c \geq 9$.

      Can also expand $(n + 2)^2 = n^2 + 4n + 4$ to $n^2 + 4n^2 + 4n^2 = 9n^2$.

   b) If $c = 5$, what is the smallest non-negative integer $n_0$ such that $n_0$ and $c$ are witnesses to the fact that $(n + 2)^2$ is $O(n^2)$?

      Need to find $n_0$ such that $(n + 2)^2 \leq 5n^2$ for $n \geq n_0$.

      $n^2 + 4n + 4 \leq 5n^2$.

      True for $n \geq 2$.

   c) If $n_0 = 0$, for what values of $c$ are $n_0$ and $c$ witnesses to the fact that $(n + 2)^2$ is $O(n^2)$?

      There is no such $c > 0$.

2. Prove or disprove: $a^n$ is $O(b^n)$ if $1 < a \leq b$.

   We need to show that $R$ is true, where $R$ is: $\forall n \geq n_0$ $a^n \leq c \cdot b^n$, for some positive $c$.

   Set $b = a + \epsilon$, where $\epsilon$ is some non-negative value.

   Need to show that $a^n \leq c \cdot b^n$.

   Can write as $a^n \leq c(a + \epsilon)^n$.

   For $c \geq 1$, $R$ is true for any $\epsilon$.

Algorithm Analysis

3) Write the most efficient algorithm you can think of (in C, Java, pseudo-code) for the following: Given an array of $n$ integers, find the smallest. What is the running time in terms of big-oh, big-theta, big-omega? Explain your answer.

**Naïve algorithm:** The smallest integer can easily be computed in $O(n)$ time. We have an array $A[i]$ for integer in position $i$.

```c
min = A[1]
do for i from 2 to n:
   if A[i] < min set min = A[i]
return min
```

This algorithm is actually $\Theta(n)$, since we must examine every element of the array.
4) Write the most efficient algorithm you can think of for the following: Given a set of \( p \) points, find the pair closest to each other. What is the running time in terms of big-theta? Explain your answer.

**Naïve algorithm:** The closest pair of points can easily be computed in \( O(p^2) \) time. To do that, one could compute the distances between all the \( p(p - 1) / 2 \) pairs of points, then pick the pair with the smallest distance, as illustrated below.

```plaintext
minDist = ∞
for each p in P:
  for each q in P:
    if p ≠ q and dist(p, q) < minDist:
      minDist = dist(p, q)
      closestPair = (p, q)
return closestPair
```

The problem can be solved in \( O(p \log p) \) time using a recursive, divide-&-conquer approach:
1. Sort points along the x-coordinate
2. Split the set of points into two equal-sized subsets by a vertical line \( x = x_{mid} \)
3. Solve the problem recursively in the left and right subsets. This will give the left-side and right-side minimal distances \( d_{Lmin} \) and \( d_{Rmin} \) respectively.
4. Find the minimal distance \( d_{LRmin} \) among the pair of points in which one point lies on the left of the dividing vertical and the second point lies to the right.
5. The final answer is the minimum among \( d_{Lmin}, d_{Rmin}, \) and \( d_{LRmin} \).

Step 4 actually takes \( O(n) \) time. A naïve approach would require the calculation of distances for all left-right pairs, i.e., in quadratic time. The key observation is based on the following sparsity property of the point set. We already know that the closest pair of points is no further apart than \( dist = min(d_{Lmin}, d_{Rmin}) \).
Therefore, for each point \( p \) of the left of the dividing line we have to compare the distances to the points that lie in the rectangle of dimensions \( (dist, 2 \times dist) \) to the right of the dividing line, as shown in the figure. And what is more, this rectangle can contain at most 6 points with pairwise distances at least \( d_{Rmin} \). Therefore it is sufficient to compute at most 6 for left-right distances in step 4. The recurrence relation for the number of steps can be written as \( T(n) = 2T(n / 2) + O(n) \), which we can solve using the master theorem (Case 2) to get \( O(n \log n) \).

As the closest pair of points define an edge in the Delaunay triangulation, and correspond to two adjacent cells in the Voronoi diagram, the closest pair of points can be determined in linear time when we are given one of these two structures. Computing either the Delaunay triangulation or the Voronoi diagram takes \( O(n \log n) \) time.

5) Consider the sorting algorithm shown below. Find the number of instructions executed and the complexity of this algorithm.

```plaintext
for (i = 1; i < n; i++) {
  SmallPos = i;
  Smallest = Array[SmallPos];
  for (j = i+1; j <= n; j++)
    if (Array[j] < Smallest) {
      SmallPos = j;
      Smallest = Array[SmallPos]
    }
  Array[SmallPos] = Array[i];
  Array[i] = Smallest;
}
```
Here we have a loop (line 1) with $O(n)$ complexity, and a nested loop (lines 4-7) with $O(n)$ complexity, so the total complexity is $O(n^2) = O(n^2)$.

```plaintext
for (i = 0; i < n; i++) {
    SmallPos = i;
    Smallest = Array[SmallPos];
    for (j = i + 1; j < n; j++)
        if (Array[j] < Smallest) {
            SmallPos = j;
            Smallest = Array[SmallPos];
        }
    Array[SmallPos] = Array[i];
    Array[i] = Smallest;
}
```