Mark Shinozaki
Homework #7
Cpts 350

1. Compute the Average-case complexity of badClosetpair.

→ Randomly Split the Set 'A' of points into two subsets 'A1' and 'A2'

→ badclosetpair on both subsets with find the closet pair each which yeilding distances '$\delta 1$' and '$\delta 2$'

→ For every point in A1, Compute the distance to every point in A2 and find the Smallest such distance 'δ'

→ the result is the minimum of '$\delta 1$', '$\delta 2$, and 'δ'

To find Avg case complexity,

→ The Split is linear $O(n)$

→ The recursion of badclosest pair on A1 and A2 each have a complexity that is defined as $T(n/2)$ because it is applied to the Subset of $n/2$

→ For each $n/2$ points in A1, $n/2$ points in A2 which would have a time complexity of $O((n/2)^2)$

The recurrence relation for this algorithm would be:

$$T(n) = 2T(n/2) + O((n/2)^2)^q$$

↓

However $O(n^2/4)$ will dominate as n grows large. Therefore, the Avg Case time complexity of bad closet pair is

$$O(n^2)$$

2. Compute the worst case complexity of betterkatatsuba

initialize $\beta = \alpha_1$

For $i = 2$ to n, do $\beta = \beta \cdot \alpha_i$

The katatsuba algo. has a worst case time complexity of $O(n^{1.59})$

↓

$|\beta_i| \rightarrow |\beta_1| = n \rightarrow |\beta_2| = |\beta_1| + n = 2n$

$|\beta_3| = |\beta_2| + n = 3n$ then $|\beta_i| = in$

↓

Since the time complexity is $O(n^{1.59})$, the complexity of $i\beta$ is $O((in)^{1.59}) = O(i^{1.59} * n^{1.59})$

↓

The time complexity of the loop is the sum of the time complexities of

$$T(n) = \sum_{i=2}^{\{n\}} O(i^{1.59} * n^{1.59})$$

↓

Using properties of Big O, the sum simplifies to

$$T(n) = O(n^{1.59}) * \sum_{i=2}^{\{n\}} i^{1.59}$$

$\sum_{i=2}^{\{n\}} i^{1.59}$ is a Riemann sum that equals to $(n^{2.59} - 2^{2.59})/2.59$

So the sum is $O(n^{2.59})$

Substitute it back into the equation for $T(n)$:

$$T(n) = O(n^{1.59}) * O(n^{2.59})$$

$$= O(n^{4.18})$$

Therefore the worst case time complexity of naivekaratsuba is $O(n^{4.18})$.

3. Compute worst-case complexity of betterkaratsuba.

The karatsuba algorithm has a worst case time complexity of $O(n^{1.59})$ for multiplying two bit strings of length n.

1. Divide the bit strings into 3 is constant time $O(1)$
2. When better karatsuba is run on each group, it divides the bit string into equal parts which would take $Z * T(n/2)$ time
3. Then when you apply better karatsuba algorithm this part takes $O(n^{1.59})$ time

When you combine these steps the recurrence Relation is $T(n) = Z * T(n/2) + O(n^{1.59})$

↓

$Z * T(n/2)$ is of divide and conquer with a time complexity of $O(n \log n)$

The second term $O(n^{1.59})$ is larger than $O(n \log n)$ for n so it will dominate the complexity of the Better karatsuba algorithm which would $O(n^{1.59})$ I believe if my logic is right

4. Design an algorithm that runs in time $O(n^3)$ and finds the closest pair of airplanes between all the n^3 airplanes.

a high level algorithm would be:

1. Divide the cube into smaller cubes of side 1 in length. This creates n^3 cubes. Each smaller cube holds at least one airplane since the minimum distance between any two planes is 1. $O(1)$
2. Create a hash table where each entry corresponds to smaller cube. $O(n^3)$
3. Check the neighboring cubes for each airplane. $O(n^3)$ in total

The algorithm runs in $O(n^3)$ time

* the true complexity is dominated by copying strings and comparing each pair in $O(nm)$

```
Cal_difference(alpha, beta)
    count_alpha_a = alpha.count('a')
    count_alpha_b = alpha.count('b')
    count_beta_a = beta.count('a')
    count_beta_b = beta.count('b')
    difference = abs(count_alpha_a - count_beta_a)**2
               + abs(count_alpha_b - count_beta_b)**2
    return difference
```

5. in python an algorithm that achieves this would be, $O(nm)$ time

```
def Smallest_difference(A)
    min_difference = float(inf)
    for i in range(len(A)):
        for j in range(i+1, len(A)):
            if i != j:
                difference = Cal_difference(A[i], A[j])
                if difference < min_difference:
                    min_difference = difference
    return min_difference
```