1. a) alg1 $\Rightarrow$ $T(n) = T(n-1) + 1 =$

$$T(n-2) + 1 + 1$$

$$T(n-3) + 1 + 1 + 1$$

$$\vdots$$

$$\underbrace{1 + 1 + 1 + \ldots + 1}_{n \text{ times}} = n \in \Theta(n)$$

CamScanner ile tarandı

b) alg 2 $\Rightarrow$ $T(n) = 2T(\frac{n}{2}) + 1$

$a = 2$

$b = 2$ $\qquad a > b^k \Rightarrow \boxed{2 > 2^0}$

$c = 1$ $\qquad T(n) \in \Theta(n^{\log_b a})$

$k = 0$ $\qquad T(n) \in \Theta(n)$

Both algorithms are in the $\Theta(n)$ complexity class, but the second one makes more function calls than the first one does. So, I would prefer the first algorithm.

2) Brute force algorithm complexity:

For every element of polinomial, power function is called.

Assume there is n number of polinomial element and average number of power of polinomial element is m

Brute force algorithm has a power function which calculates the power of number with $\Theta(m)$ complexity.

```
def power(num, pow):
    if(pow<=0):
        return 1
    return (num * power(num, pow-1))
```

$$T(n) = T(n-1)+1$$
$$T(n) = \sum_{i=0}^{n} 1 = n \in \Theta(n)$$

More efficient power function has $\Theta(\log m)$ complexity

```
def power2(num, pow):
    if (power ≤ 0):
        return 1
    val = 1
    if( pow % 2 == 1):
        val = num
        pow = pow - 1
    temp = power2(num, pow/2)
    return ( val * temp * temp)
```

$$T(n) = T\left(\frac{n}{2}\right)+1$$
if $n = 2^k \Rightarrow \boxed{k = \log n}$
$$T(2^k) = T(2^{k-1})+1$$
$$T(2^{k-1}) = T(2^{k-2})+1$$
$$\vdots \quad \} k \text{ steps}$$
$$T(2^k) = k.1 = k$$
$$\boxed{T(n) = \Theta(\log n)}$$

If the polynomial function implemented using this power function, its complexity becomes $\Theta(n \log m)$.

$$\sum_{i=0}^{n} \log m = n.\log m = \Theta(n \log m)$$

3) Algorithm counts the number of start symbols and when encountered with end symbol, it adds the number of start symbols found 'so far' to sum. It traverses the string only once so, its complexity is $\Theta(n)$

$$\boxed{\sum_{i=0}^{n} 1 = \Theta(n)}$$

4) Euclidian distance: $d(p,q) = \sqrt{(p_1-q_1)^2 + (p_2-q_2)^2 + \cdots (p_n-q_n)^2}$

```
func  eucDis(points, n, k):
    minDist = inf

    for i=0 to n-1:
        for j=i to n:
            foundDist = dist(points[i], points[j], k)
            if (foundDist < minDist):
                minDist = foundDist
            endif
        endfor
    endfor
    return sqrt(mindist)
end

func dist(p, q, k):
    sum = 0
    for i=0 to k:
        sum += ((p-q) * (p-q))
    endfor
    return sum
end
```

Complexity: $\sum_{i=0}^{n-1} \sum_{j=i}^{n} \sum_{0}^{k} 1 = \sum_{i=0}^{n-1} \sum_{j=i}^{n} k = \sum_{i=0}^{n-1} (n-i).k =$

$$= k \sum_{i=0}^{n-1} (n-i) = k(\underbrace{n + n-1 + \ldots + 1}_{\frac{n.(n+1)}{2}}) = k . \frac{n.(n+1)}{2} \in \Theta(n^2 k)$$

5) a) $$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\sum_{k=i}^{j+1} 1 = \sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j+1-i) = \sum_{i=0}^{n-1}\Big(\underbrace{(1)+(2)+\dots+(n-1)}_{\frac{n.(n-1)}{2}}\Big)$$

$$= \sum_{i=0}^{n-1}\frac{n.(n-1)}{2} = n.\left(\frac{n.(n-1)}{2}\right)$$

$$= \frac{n^3-n^2}{2} \in \Theta(n^3)$$