1) a) Algoritm alg1 (L[0..n-1])

$Q(1) \leftarrow$ if (n==1) return L[0]

else

$T(n-1) \leftarrow$ temp = alg1 (L[0..n-2])

$Q(1) \leftarrow$ if (temp <= L[n-1]) return temp

$Q(1) \leftarrow$ else return L[n-1]

$T(n) = T(n-1) + Q(1)$

$\hookrightarrow T(n-2) + Q(1)$

$\sum_{1}^{n} Q(1) = Q(n)$

$\hookrightarrow$

$$\boxed{T(n) = Q(n)}$$

b) Algoritm alg2 (X[l ... r])

$Q(1) \leftarrow$ if (l==r) return X[l]

else

$Q(1) \leftarrow$ flr = floor ((l+r)/2)

$T(n/2) \leftarrow$ temp1 = alg2 (X[l ... flr])

$T(n/2) \leftarrow$ temp2 = alg2 (X[flr+1 ... r])

$Q(1) \leftarrow$ if (tmp1 <= tmp2) return tmp1

$Q(1)$ else return tmp2

$$T(n) = 2T(n/2) + \underbrace{Q(1)}_{n^0}$$

Master Theorem

$a = 2 \quad b = 2 \quad k = 0$

$a = 2 > \frac{2}{b^k} = 2^0 \Rightarrow T(n) = Q(n)$

I would prefer the first algoritm first algoritm uses less extra memory only tmp

2) def func(A[0,...n-1], x)

result = 0

Q(n) ← for i in range(0, n):

result += (A[n-i-1] * (x ** i))

return result

$T(n) = Q(n)$

it is not possible to design an algorithm to have better complexity.

Because we have to see all elements.

3) def count_str(str[0,...n-1], first, last)

count = 0

Q(n) ← for i in range(0, len(str))

if (str[i] == first):

Q(n-i) ← for j in range(i+1, len(str)):

Q(1) ← if (str[j] == last)

count += 1

return count

$T(n) = O(n^2)$

4) def min_distance(space[0,...n-1][0...k-1])

min = 999 # maximum number

sum = 0

Q(n) ← for i in range(0, len(space)):

Q(n-1) ← for j in range(i+1, len(space)):

Q(k) ← for k in range(0, len(space[0])):

sum += ((space[i][k] - space[j][k]) ** 2)

dist = sqrt(sum)

if dist < min: min = dist

sum = 0

return min

k dimensional space

$T(n) = O(kn^2)$

5)

a) def most_profitable(regions[0...n-1][0,1])

most = []

max = 0

O(n) for i in range(0, len(regions)):

sum = 0

O(n) for j in range(i, len(regions)):

sum += regions[j][1]

if sum > max:

max = sum

most = regions[i:j+1]

$O(n^2)$

$O(n)$

$O(n^2)$

return most

$O(n^3)$

b) def max_profit(arr[l...r])

if len(arr) == 1:

return arr[0][1]

index = int(len(arr)/2)

left = max_profit(arr[0:index]) → T(n/2)

right = max_profit(arr[index:None]) → T(n/2)

return left + right

$T(n) = 2T(n/2) + 1$

$\hookrightarrow 2T(n/4) + 1$

$\hookrightarrow$

$\log_2 n$

$2^{\log_2 n} = n \Rightarrow T(n) = O(n)$