

Polynomial(2) test results:

The screenshot shows a terminal window titled "Run: polynomial_161044112". The output displays three test cases for a polynomial function. Each case shows the input parameters (n and x), the polynomial coefficients, and the resulting value.

```
Run: polynomial_161044112
D:\active_courses\algorithms\hw\hw3\code\venv\Scripts\python.exe
n: 4  x: 3 polynomial: [2, -6, 2, -1]
result: 5
-----
n: 2  x: 3 polynomial: [5, -4]
result: 11
-----
n: 2  x: 5 polynomial: [2, -1]
result: 9
```

Count_str(3) test results:

The screenshot shows a terminal window displaying test results for a string counting function. It processes three different strings, identifying substrings between specified start and end letters.

```
String: TXZXXJZWX
Start_letter: X      End_letter: Z
Substring count: 4
-----
String: abab
Start_letter: a      End_letter: b
Substring count: 3
-----
String: tttccc
Start_letter: t      End_letter: c
Substring count: 9
```

Cluster(5-a) test results:

```
D:\active_courses\algorithms\hw\hw3\code\ver
Table:
['A', 'B', 'C', 'D', 'E', 'F', 'G']
[3, -5, 2, 11, -8, 9, -5]
The most profitable cluster:
['C', 'D', 'E', 'F']
[2, 11, -8, 9]
-----
Table:
['A', 'B', 'C', 'D', 'E', 'F', 'G']
[333, -5, 2, 11, -8, 9, 500]
The most profitable cluster:
['A', 'B', 'C', 'D', 'E', 'F', 'G']
[333, -5, 2, 11, -8, 9, 500]
-----
Table:
['A', 'B', 'C', 'D', 'E', 'F', 'G']
[344, -5, 2, 11, -8, 9, -5]
The most profitable cluster:
['A', 'B', 'C', 'D', 'E', 'F']
[344, -5, 2, 11, -8, 9]
```

1a

$T(n) \leftarrow \text{Algorithm alg 1 } (L[0-n-1])$

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

else

$T(n-1) \leftarrow [\text{tmp} = \text{alg1 } (L[0-n-2])]$

2 \leftarrow if ($\text{tmp} \leq L[n-1]$) return tmp

2 \leftarrow else return $L[n-1]$

$$T(n) = \begin{cases} 2 & n=1 \\ T(n-1)+4 & n>1 \end{cases}$$

Substitute :

$$T(n) = T(n-1)+4$$

$$= [T(n-2)+4]+4$$

$$= [[T(n-3)+4]+4]+4$$

$$\hookrightarrow T(n-3)+3 \cdot 4$$

$$\vdots \quad \quad \quad = T(n-(n-1))+(n-1) \cdot 4$$

$$= T(1)+(n-1) \cdot 4$$

$$= 4n-1+2 = \Theta(n)$$

* ~~Don't do this~~

①

①b) Algorithm alg2 ($x[l \dots r]$)

```

 $1 \leftarrow \text{if } (l=r) \text{ return } x[l]$ 
    else
         $1 \leftarrow \text{flr} = \text{floor}((l+r)/2)$ 
         $T(\frac{n}{2}) \leftarrow \text{tmp1} = \text{alg2}(x[l \dots \text{flr}])$ 
         $T(\frac{n}{2}) \leftarrow \text{tmp2} = \text{alg2}(x[\text{flr}+1 \dots r])$ 
         $2 \leftarrow \begin{cases} \text{if } (\text{tmp1} < \text{tmp2}) \text{ return tmp1} \\ \text{else return tmp2} \end{cases}$ 

```

$$T(n) = \begin{cases} 1 & l = r \\ 2(T_{\frac{n}{2}}) + 3 & l < r \end{cases}$$

\downarrow

$$T(n) = 2\left(\frac{T}{2}\right) + 3$$

According to case III,

$$T(n) = \Theta(n^{\log_b a})$$

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

$$T(n) = \Theta(n)$$

$$\begin{aligned} a &= 2 \geq 1 & \checkmark \\ b &= 2 \geq 2 & \checkmark \\ c &= 1 > 0 & \checkmark \end{aligned}$$

$$\begin{aligned} f(n) &= 3 = 3n^d \\ \Rightarrow d &= 0 \end{aligned}$$

If $f(n) \in \Theta(n^d)$ where $d \geq 0$ then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^{\log_b a}) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

case I
case II
case III

case III $a > b^d$

$$2 > 2^0$$

$$2 > 1 \quad \checkmark$$

* Complexities of the algorithms are the same. Algorithm b allows parallel processing. But Algorithm a have smaller number of function calls. Algorithm b may be choosable.

(2)

② calc-polynomial (poly, x)

```

I ← res ← 0
I ← n ← len(poly)
n ← for i ← 0 to n-1 :
    I ← product = 1
    I ← for j ← 0 to i :
        product = * * product
    I ← res = res + poly[n-i-1] * product
I ← return res

```

$$3 + \sum_{i=0}^n \sum_{j=0}^i 1 \Rightarrow \sum_{i=0}^n i \Rightarrow \frac{n(n+1)}{2} = \underline{\underline{O(n^2)}}$$

If we used Horner's method our complexity will be $O(n)$.

Horner's Rule:

$$\begin{aligned} & a_0 + a_1x + a_2x^2 + \dots + a_nx^n \\ & = a_0 + (x(a_1 + x(a_2 + x(a_3 + \dots + x(a_{n-1} + x a_n))))). \end{aligned}$$

By using this rule polynomial will be solved with n multiplication and n addition which serve us $O(n)$ complexity.

Note: Python implementation is in polynomial-16/ohh112.py

③

③ count_substrings(string, start-letter, end-letter):

counter $\leftarrow 0$

for $i \leftarrow 0$ to $\text{len}(\text{string}) - 1$:

if $\text{string}[i] \neq \text{start-letter}$:
 continue
end if

for $j \leftarrow (i+1)$ to $\text{len}(\text{string})$:

if $\text{string}[j] == \text{end-letter}$:
 counter $\leftarrow \text{counter} + 1$
 end if
end for

end for

return counter

end procedure

$$\sum_{i=0}^n \sum_{j=i+1}^n 1 \Rightarrow \sum_{i=0}^{n-1} n-i \Rightarrow n-0 + n-1 + n-2 + \dots + n-(n-1) \\ \Rightarrow n + (n-1) + (n-2) + \dots + 1 \\ = \frac{n(n+1)}{2} = O(n^2)$$

④

④

Procedure Closest Pair :

$$\min = \infty$$

for i from (1) to ($n-1$) :

{
 for j from ($i+1$) to (n) :
 $k\{ \text{dist} = \text{Euclidean Dist}(L[i], L[j])$
 if $\min > \text{dist}$:
 $\min = \text{dist}$
 end if
 end for
end for

end procedure

$$\sum_{i=0}^{n-1} \sum_{j=i+1}^n (k+1) \Rightarrow \sum_{i=0}^{n-1} (k+1)(n-i)$$
$$\Rightarrow (k+1)(1) + (k+1)(2) + (k+1)(3) \dots (k+1)(n)$$
$$\Rightarrow (k+1)(1+2+3\dots+n) \Rightarrow \frac{n(n-1)}{2} \cdot (k+1)$$
$$\Rightarrow \Theta(k \cdot n^2)$$

⑤

⑤ procedure find-most-prof (reg-prof-list)

⑥ max-start = (-1)

max-end = (-1)

g-max = -infinity

for i ← 0 to len (reg-prof-list)

for j ← (i+1) to len (reg-prof-list)

(j-i) sum-sublist = sublist (reg-prof-list [i:j+1])

if sum-sublist > g-max

g-max = sum-sublist

max-start = i

max-end = j+1

end for

end for

return reg-prof-list [max-start : max-end]

end procedure

$$\sum_{i=0}^n \sum_{j=i+1}^n j-i \Rightarrow \sum_{i=0}^n [(i+1)-i] + [(i+2)-i] + \dots + [n-i]$$

$$\Rightarrow \sum_{i=0}^n 1+2+3+\dots+(n-i)$$

$$\Rightarrow \sum_{i=0}^n \frac{(n-i)(n-i-1)}{2} \Rightarrow \sum_{i=0}^n n^2 \Rightarrow n \cdot n^2 \in \underline{\mathcal{O}(n^3)}$$

⑥