How to write an algorithm?

① algorithm Swap (a, b)
{
temp = a;
a = b;
b = temp;
}
(data types not defined in algo)

② algorithm Swap (a,b)
begin
temp = a;
a = b;
b = temp;
end.

③ algorithm Swap (a,b)
begin
temp := a;
a := b;
b := temp;
end

④ algorithm Swap (a,b)
begin
temp ← a;
a ← b;
b ← temp;
end

How to analyze an algorithm?

① time complexity :- algo should be time efficient.

after writing algo, we analyze the time, we get 'time function'.

② space :- algo → program → space consumed on the machine.

③ data transfer, network consumption (internet based, web based, cloud based)

④ Power consumption (using algo for hand held device)

⑤ cpu register consumption (algo for device drivers / system level programming)
(device drivers
system level programming)

Analysis :- (time complexity)

algorithm swap (a, b)
{
temp = a; → 1 unit of time
a = b; → 1
b = temp; → 1
}

(every simple statement takes 1 unit of time)

f(n) = 3 (simple statement)

time function. ↳ constant value. O(1)
↳ (represented as.)

Space analysis :- a, b, temp (variable used)

S(n) = 3 (constant) O(1) (usually represented as)
(3 words)
↳ converted to program what data type it takes is unknown so, we represent it as word.

Note : we can also get into deeper analysis (machine code)

x = a*5 + b*6; → 1 unit of time

(actually it may take more : 2 multiplications, 1 addition and 1 assignment)

converted to machine code is different.

Frequency Count Method

① Sum of finding of all elements in an array

algorithm Sum (A, n) ⑤
(array)
{
S = 0; → 1

(i has changed 5 times) 5 < 5 (stop)

i=0
i=1
i=2
i=3
i=4
i=5 (stop)

for ($\underbrace{i=0}_{1}$; $\underbrace{i<n}_{n+1}$; $\underbrace{i++}_{n}$) → n+1 (2n+2)
{
S = S + A[i]; → n (as the for loop executes 'n' times)
}
return S; → 1
}

2n+3.
degree of polynomial.

Solution :- A

8	3	9	7	2
0	1	2	3	4

A = 5

$f(n) = 2n+3$
$f(n) = O(n)$ 'order of n'.

space complexity :- A, n, S, i
A ↓ n words
n, S, i → 1 word.
(n+3)
= 3+n = $O(n)$
$S(n) = O(n)$

MEGHANA G RAJ
ASSISTANT PROFESSOR

② Finding the sum of matrices (time) $n \times m$

3×3

algorithm add (A, B, n)
{
for (i=0; i<n; i++) $\rightarrow n+1$
{
for (j=0; j<n; j++) $\rightarrow n * (n+1)$
{
c[i, j] = A[i, j] + B[i, j]; $\rightarrow n \times n$
}
}
}

$f(n) = 2n^2 + 2n + 1$

$f(n) = n^2$.

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

for loop executes $\rightarrow n+1$
remaining statements inside the for loop executes 'n' times.

space :- variables :- a b c n i j

$$\underbrace{\begin{matrix} a & b & c \\ \downarrow & \downarrow & \downarrow \\ n^2 & n^2 & n^2 \end{matrix}}_{\text{matrices (2-d arrays } n \times n = n^2)}$$

$n, i, j \rightarrow 1$ scalar variable (simple)

Space function is $3n^2 + 3$
degree $O(n^2)$

③ multiplication of two matrices

(nested loops)

```
algorithm multiply (A, B, n)
{
  for (i=0; i<n; i++)                          → n+1              (n+1)
  {
    for (j=0; j<n; j++)                        → n*n+1            (n^2+n)
    {
      C[i,j]=0;                                → n*n              (n^2)
      for (k=0; k<n; k++)                      → n*n*n+1          (n^3+n^2)
      {
        C[i,j] = C[i,j] + A[i,k] * B[k,j];     → n*n*n            (n^3)
      }
    }
  }
}
```

$f(n) = 2n^3 + 3n^2 + 2n + 1$

$= O(n^3)$

Space :- $\underbrace{A \quad B \quad C}_{n^2 + n^2 + n^2}$ $\underbrace{n \quad i \quad j \quad k}_{\text{scalar } 4}$ $= 3n^2 + 4 = O(n^2)$

time $O(n^3)$

space $O(n^2)$

④ transpose of matrix.

```
transpose (a, n)
{
  for (i=0; i<n; i++)                 ⟶ n+1
  {
    for (j=i+1; j<n; j++)             ⟶ n(n+1)/2 = (n^2+n)/2
    {
      swap (a[i][j], a[j][i]);        ⟶ n * n = n^2
    }
  }
}
```

$\underbrace{i<n}_{(n+1)}$

$j = 1, 2 \cdots n$

↓

(sum of first 'n' natural numbers)

$\cancel{f(n) = 2n^2+2n+1}$

$O(n^2)$ $\cancel{O(n^2)}$

⑤

`i=1;`	⟶	1	
`sum=0;`	⟶	1	
`while (i<=n)`	⟶	$n+1$	
`{`			
`j=1;`	⟶	$\underline{n}$	
`while (j<=n)`	⟶	$\underline{n}\ (n+1)$	n^2+n
`{`			
`sum = sum+1;`	⟶	$\underline{n} * n$	n^2
`j = j+1;`	⟶	$\underline{n} * n$	n^2
`}`			
`i = i+1;`	⟶	$\underline{n}$	n
`}`			

$f(n) = 3n^2 + 4n + 3$

$O(n^2)$

Commonly Used rates of growth.

$n!$

$\downarrow$

2^n

$\downarrow$

n^2

$\downarrow$

$n\log n$

$\downarrow$

$\log(n!)$

$\downarrow$

n

$\downarrow$

$2^{\log n}$

$\downarrow$

$\log\log n$

$\downarrow$

1

decreasing rates of growth

Time Complexity - Problem type 1

① for (i=0 ; i<n ; i++) → n+1

{

Stament ; → n

}

(0 → n) ascending order.

O(n)

② for (i=n ; i>0 ; i--) → n+1

{

stament; → n

}

(n→0) decreasing order.

O(n)

③ for (i=1; i<n; i=i+2)

{

statement; → n/2

}

O(n)

④ for (i=1; i<n; i=i+20)

{

statement; → n/20

}

O(n)

(5)

```
for (i=0; i<n; i++)          → n+1
{
  for (j=0; j<n; j++)        → n(n+1)
  {
      stmt;                  → n×n
  }
}
```

$O(n^2)$

(6)

```
for (i=0; i<n; i++)
{
  for (j=0; j<i; j++)  → 0<1 ✓
  {                      1<1 ×
     stmt;
  }
}
```

total $= 0+1+2+\cdots+n = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$

$O(n^2)$

tracing

i	j	no of times stmt executed
0	0	0
1	0 ✓ 1 ×	1 → (break)
2	0 1 2 ×	2 → (break)
3	0 1 2 3 ×	3 → (break)
n		n

(incrementing) for (i=0; i<n; i++)	$O(n)$
for (i=0; i<n; i=i+2)	$O\left(\frac{n}{2}\right) = O(n)$ $\quad \frac{n}{200} = O(n)$ only
(decrementing) for (i=n; i>1; i--)	$O(n)$
for (i=1; i<n; i=i*2)	$O(\log_2 n)$
for (i=1; i<n; i=i*3)	$O(\log_3 n)$
for (i=n; i>1; i=i/2)	$O(\log_2 n)$