Complexity Analysis

By

Suvam Basak

Email: suvambasak22@iitk.ac.in
Web: suvambasak.github.io

Which one is most efficient approach?

Approach 3

```python
def gcd(m: int, n: int):
    common_factors = 1
    for f in range(2, min(m, n)+1):
        if m % f == 0 and n % f == 0:
            common_factors = f

    return common_factors
```

Approach 4

```python
def gcd(m: int, n: int):
    cf = min(m, n)
    while cf > 0:
        if m % cf == 0 and n % cf == 0:
            return cf
        cf -= 1
```

Asymptotic analysis

- Estimation of CPU time and main memory space required to complete the execution of the algorithm
  - Time complexity:
    - Frequency count/Sum of frequency
  - Space complexity:
    - Extra space consumption for the execution of the algorithm
    - Not the input size

Asymptotic analysis: Time complexity

```
def gcd(m: int, n: int):
    factors_m = []-----------------------→ 1
    factors_n = []-----------------------→ 1
    for i in range(1, m+1):---------------→ m
        if m % i == 0:--------------------→ m
            factors_m.append(i)-----------→ m
    for i in range(1, n+1):---------------→ n
        if n % i == 0:--------------------→ n
            factors_n.append(i)-----------→ n
    common_factors = []------------------→ 1
    for f in factors_m:------------------→ m
        if f in factors_n:---------------→ mxn
            common_factors.append(f)------→ mxn
    return common_factors[-1]------------→ 1
```

TC = 4m+3n+2(mn)+4

TC = $\Theta(mn)$

Asymptotic analysis: Space complexity

```
def gcd(m: int, n: int):
    factors_m = []------------------------→ m
    factors_n = []------------------------→ n
    for i in range(1, m+1):---------------→ 1
        if m % i == 0:
            factors_m.append(i)
    for i in range(1, n+1):
        if n % i == 0:
            factors_n.append(i)
    common_factors = []-------------------→ m
    for f in factors_m:
        if f in factors_n:
            common_factors.append(f)
    return common_factors[-1]
```

SC = 2m + n + 1

SC = Θ(m)

Time complexity: $\Theta(n)$

```
for (i=1; i<=n; i++)
     printf("Text");
```

```
for (i=n; i>=1; i--)
     printf("Text");
```

Time complexity?

```
for (i=1; i<=n; i=i+5)
     printf("Text")
```

Time complexity?

```
for (i=1; i<=n; i=i+5)
    printf("Text")
```

Exit when: i = $1+(k \times 5) <= n$

➔ $k <= (n-1)/5$

TC: $\Theta(n)$

1. i=1
2. i=1+5
3. i=1+(2x5)
4. i=1+(3x5)
5. i=1+(4x5)
6. ...
7. i=1+(k x 5)

Time complexity: $\Theta(n)$

```
for (i=1; i<=n; i++)
    printf("Text");
```

```
for (i=n; i>=1; i--)
    printf("Text");
```

```
for (i=1; i<=n; i=i+5)
    printf("Text");
```

```
for (i=n; i>=1; i=i-5)
    printf("Text");
```

Time complexity?

```
for (i=1; i<=n; i=i*2)
    printf("Text");
```

Time complexity?

```
for (i=1; i<=n; i=i*2)
      printf("Text");

for (i=n; i>=1; i=i/2)
      printf("Text");
```

1. i=1
2. i=2
3. $i=2^2$
4. $i= 2^3$
5. $i= 2^4$
6. …
7. $i= 2^k$

Exit when: $i = 2^k <= n$

➔ $k = \log n$

TC: $\Theta(\log n)$

Time complexity?

```
for (i=2; i<=n; i=i²)
     printf("Text");

for (i=n; i>=1; i=i^(1/2))
     printf("Text");
```

Time complexity?

```
for (i=2; i<=n; i=i²)
        printf("Text");

for (i=n; i>=1; i=i^(1/2))
        printf("Text");
```

1. $i=2$
2. $i=2^2$
3. $i=2^{2^\wedge 2}$
4. $i= 2^{2^\wedge 3}$
5. $i= 2^{2^\wedge 4}$
6. …
7. $i= 2^{2^\wedge k}$

Exit when: $i = 2^{(2^\wedge k)} <= n$

➔ $2^k = \log n$

➔ $k = \log \log n$

TC: $\Theta(\log \log n)$

Time complexity?

```
for (i=1; i<=n; i++)
     for (j=1; j<=n; j++)
          printf(“Text”);
```

```
for (i=1; i<=n; i++)
     for (j=1; j<=n; j=j*2)
          printf(“Text”);
```

Time complexity?

```
for (i=1; i<=n; i++) ------------------→n
        for (j=1; j<=n; j++) -----------→ n
                printf("Text");---------→ n²
```

TC: $\Theta(n^2)$

```
for (i=1; i<=n; i++) -----------------→ n
        for (j=1; j<=n; j=j*2) -------→ log n
                printf("Text");-------→ n log n
```

TC: $\Theta(n \log n)$

Time complexity?

```
def fact(n: int) -> int:
    if n == 0 or n == 1:
        return 1
    else:
        return n * fact(n-1)
```

Space complexity

1. fact(5)
2. fact(4)
3. fact(3)
4. fact(2)
5. fact(1)

Maximum recursion depth (STACK)

$\Theta(n)$

Time complexity?

```
def fact(n: int) -> int:
    if n == 0 or n == 1:
        return 1
    else:
        return n * fact(n-1)
```

- $T(n) = T(n-1) + C$, $n>1$
- $T(n) = 1$, $n<=1$

$(n\text{-}k) = 1$
$K = (n\text{-}1)$

Time complexity?

```
def fact(n: int) -> int:
    if n == 0 or n == 1:
        return 1
    else:
        return n * fact(n-1)
```

- $T(n) = T(n-1) + C$, $n>1$
- $T(n) = 1$, $n<=1$

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

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

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

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

…

$= T(n-k) + k.C$

$= T(n-(n-1)) + (n-1)\ C$

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

$= 1+ C\ (n-1) \rightarrow \Theta(n)$

$(n-k) = 1$
$K = (n-1)$

Time complexity?

```
def fact(n: int) -> int:
    if n == 0 or n == 1:
        return 1
    else:
        return n * fact(n-1)
```

- $T(n) = T(n-1) + C$, $n>1$
- $T(n) = 1$, $n<=1$

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

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

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

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

.

$= T(n-k) + k.$

\# function calls

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

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

$= 1+ C\,(n-1) \rightarrow \Theta(n)$

Time complexity?

```
def rec(n):
    if n <=1:
        return 1
    return n + rec(n-1) + rec(n-1)
```

Time complexity?

```
def rec(n):
    if n <=1:
        return 1
     return n + rec(n-1) + rec(n-1)
```

- $T(n) = 2T(n-1) + C$, $n>1$
- $T(n) = 1$, $n<=1$

Time complexity?

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

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

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

$= 2^2T(n-2)+2C+C$

$= 2^2 [2T(n-3)+C]+2C+C$

$= 2^3T(n-3)+2^2C+2C+C$

$= 2^kT(n-k) +(2^k+\ldots+2^2+2+1)C$

$= 2^{n-1}T(n-(n-1)) +(2^{n-1}+\ldots+2^2+2+1)C$

$= 2^{n-1}T(1) +(2^{n-1}+\ldots+2^2+2+1)C \rightarrow \Theta(2^n)$

$(n-k) = 1$

$k = (n-1)$

Time complexity?

```
def rec(n):
    if n <=1:
        return 1
     for i in range(n):
         //code//
     return n + rec(n-1)
```

Time complexity?

```
def rec(n):
    if n <=1:
        return 1
     for i in range(n):
         //code//
     return n + rec(n-1)
```

- $T(n) = 1, n<=1$
- $T(n) = T(n-1)+n, n > 1$

Time complexity?

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

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

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

$= T(n-3) +(n-2) + (n-1) + n$

..

$= T(n-k) + (n-(k-1)) + (n-(k-2)) + \ldots + (n-2) + (n-1) + n$

$= T(n-(n-1)) + (n-((n-1)-1)) + (n-((n-1)-2)) + \ldots + (n-2) + (n-1) + n$

$= 1 + 2 + 3 + 4 + \ldots + n$

$= n(n+1)/2 \rightarrow \Theta(n^2)$

$(n-k) = 1$

$k = (n-1)$