TUGAS 2

Nama : Abdul Rahman
NIM : 23/528808/SPA/00987
Prodi : Doktor Ilmu Komputer
MK : Komputabilitas dan Kompleksitas (MII227001)
Dosen : Bapak Faizal Makhrus, S.Kom., M.Sc., Ph.D.

Rekursi adalah teknik dalam pemrograman di mana sebuah fungsi memanggil dirinya sendiri sebagai bagian dari proses penyelesaian masalah. Masalah yang diselesaikan dengan rekursi biasanya dipecah menjadi sub-masalah yang lebih kecil dengan pola atau struktur yang serupa dengan masalah asli.

Kompleksitas **Rekursi O(n)**	
`def recursive_sum(n):`	Mendefinisikan fungsi `recursive_sum` yang menerima satu parameter n.
`global operation_count`	Mengakses variabel global `operation_count` untuk menghitung total pemanggilan rekursif.
`operation_count += 1`	Menambah `operation_count` setiap kali fungsi dipanggil. Mengukur jumlah total operasi rekursif.
`if n == 1:`	Memeriksa apakah `n=1`, yang merupakan base case untuk menghentikan rekursi. Base case, berhenti di sini tanpa mempengaruhi kompleksitas O(n).
`return 1`	Jika kondisi base case terpenuhi, mengembalikan nilai 1 tanpa memanggil fungsi lagi.
`return n + recursive_sum(n - 1)`	Jika `n>1`, memanggil `recursive_sum` dengan `n−1` yang memulai rekursi. Pemanggilan ini terjadi n kali sehingga kompleksitasnya `O(n)`.

Kompleksitas **Rekursi O(n log n)**	

`def merge_sort(arr):`
`  if len(arr) > 1:`
`  mid = len(arr) // 2`
`  left = merge_sort(arr[:mid])`
`  right = merge_sort(arr[mid:])`
`  return merge(left, right)`
`  else:`
`  return arr`
	Fungsi `merge_sort` secara rekursif membagi array menjadi dua bagian hingga ukurannya menjadi 1. Setiap pembagian ini memakan waktu **O(log n)**, karena array dibagi setengahnya pada setiap langkah hingga array terkecil (base case) tercapai.

`def merge(left, right):`
`  result = []`
`  i = j = 0`
`  while i < len(left) and j < len(right):`
`  if left[i] < right[j]:`
`  result.append(left[i])`
`  i += 1`
`  else:`
`  result.append(right[j])`
`  j += 1`
`  result.extend(left[i:])`
`  result.extend(right[j:])`
`  return result`
	Fungsi merge menggabungkan dua sub-array yang sudah diurutkan (left dan right) menjadi satu array yang terurut. Setiap elemen dari kedua array dibandingkan dan dimasukkan ke dalam array hasil. Karena setiap elemen diproses satu kali, penggabungan memiliki kompleksitas **O(n)**.

Kesimpulan : O(log n) berasal dari proses pembagian array menjadi dua hingga mencapai ukuran 1.**O(n)** berasal dari proses penggabungan elemen pada setiap level rekursi. Kombinasi kedua langkah ini memberikan kompleksitas total **O(n log n)**.	

Kompleksitas **Rekursi O(n^2)**	
`def A_with_operations(n, count=0):`	Mendefinisikan fungsi rekursif dengan dua parameter: n dan count.
`if n > 1:`	Kondisi pengecekan apakah `n > 1`. Jika benar, rekursi akan terus berjalan.
`count += n`	Setiap iterasi loop menambah nilai n ke total operasi. Ini membutuhkan waktu **O(n)** pada setiap level rekursi.
`for i in range(1, n+1): pass`	Loop ini berjalan sebanyak n kali, artinya kompleksitas waktu dari langkah ini adalah **O(n)**.
`return A_with_operations(n-1, count)`	Setelah menyelesaikan iterasi, fungsi akan memanggil dirinya sendiri dengan `n-1`. Kedalaman rekursi adalah **n**.
`else: return count + 1`	Kondisi dasar (base case) adalah ketika `n <= 1`. Pada saat ini, rekursi berhenti, dan total operasi dikembalikan.
`Kesimpulan` : Total operasi adalah: $T(n) = O(n)+O(n-1)+O(n-2)+...+O(1)=O(n^2)$	

Kompleksitas Rekursi O(n)

```python
import time

# Inisialisasi counter untuk menghitung operasi
operation_count = 0

# Fungsi rekursi sesuai dengan A(n)
def A(n):
    global operation_count
    operation_count += 1  # Menghitung setiap kali fungsi dipanggil
    if n > 1:
        A(n - 1)
    else:
        return 1  # Base case

# Fungsi untuk menghitung jumlah operasi dan waktu eksekusi
def measure_operations_and_time(n):
    global operation_count
    operation_count = 0  # Setel ulang penghitung operasi
    start_time = time.time()  # Waktu mulai
    A(n)  # Memanggil fungsi A(n)
    end_time = time.time()  # Waktu selesai

    # Menghitung waktu yang dibutuhkan
    time_taken = end_time - start_time

    return operation_count, time_taken

# Daftar nilai n yang ingin diuji
n_values = [10, 50, 100, 500, 1000]

# Menampilkan hasil untuk setiap n
for n in n_values:
    operations, time_taken = measure_operations_and_time(n)
    print(f"n = {n}: Jumlah operasi = {operations}, Waktu = {time_taken:.10f} detik")
```

```
n = 10: Jumlah operasi = 10, Waktu = 0.0000061989 detik
n = 50: Jumlah operasi = 50, Waktu = 0.0000061989 detik
n = 100: Jumlah operasi = 100, Waktu = 0.0000097752 detik
n = 500: Jumlah operasi = 500, Waktu = 0.0001218319 detik
n = 1000: Jumlah operasi = 1000, Waktu = 0.0002210140 detik
```

Waktu yang dibutuhkan untuk menjalankan fungsi

n	T(n) dalam detik
10	0,0000014305
50	0,0000061989
100	0,0000097752
500	0,0001218319
1000	0,0002210140

Kompleksitas Rekursi O(n log n)

```python
# Fungsi rekursif merge sort
def merge_sort(arr):
    global operation_count
    if len(arr) > 1:
        mid = len(arr) // 2
        left = merge_sort(arr[:mid])
        right = merge_sort(arr[mid:])

        return merge(left, right)
    else:
        return arr

# Fungsi untuk menghitung jumlah operasi dan waktu eksekusi
def measure_operations_and_time(n):
    global operation_count
    operation_count = 0  # Setel ulang penghitung operasi
    arr = list(range(n, 0, -1))  # Array terbalik untuk merge sort
    start_time = time.time()  # Waktu mulai
    merge_sort(arr)  # Memanggil fungsi merge_sort
    end_time = time.time()  # Waktu selesai

    # Menghitung waktu yang dibutuhkan
    time_taken = end_time - start_time

    return operation_count, time_taken

# Daftar nilai n yang ingin diuji
n_values = [10, 50, 100, 500, 1000]

# Menampilkan hasil untuk setiap n
for n in n_values:
    operations, time_taken = measure_operations_and_time(n)
    print(f"n = {n}: Jumlah operasi = {operations}, Waktu = {time_taken:.10f} detik")
```

```
n = 10: Jumlah operasi = 19, Waktu = 0.0000488758 detik
n = 50: Jumlah operasi = 153, Waktu = 0.0001351833 detik
n = 100: Jumlah operasi = 356, Waktu = 0.0003910065 detik
n = 500: Jumlah operasi = 2272, Waktu = 0.0019698143 detik
n = 1000: Jumlah operasi = 5044, Waktu = 0.0050439835 detik
```

Waktu yang dibutuhkan untuk menjalankan fungsi

n	T(n) dalam detik
10	0,0000488758
50	0,0001351833
100	0,0003910065
500	0,0019698143
1000	0,0050439835

Kompleksitas Rekursi O(n^2)

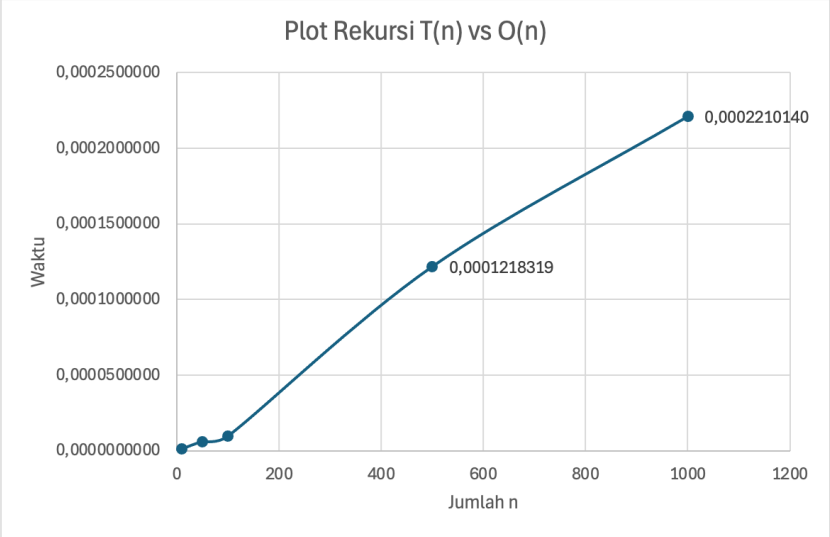
```python
import sys
import time
import matplotlib.pyplot as plt
import pandas as pd

# Let's modify the code to include the total number of operations and format the output similarly.
def A_with_operations(n, count=0):
    if n > 1:
        count += n  # Increment the operation count by the number of iterations
        for i in range(1, n+1):
            pass  # "do something" placeholder
        return A_with_operations(n-1, count)
    else:
        return count + 1  # Base case, add the final operation for n=1

n_values = [10, 50, 100, 500, 1000]
results = []

for n in n_values:
    start_time = time.time()
    operations = A_with_operations(n)
    end_time = time.time()
    elapsed_time = end_time - start_time
    results.append({
        "n": n,
        "Jumlah Operasi": operations,
        "Waktu (detik)": elapsed_time
    })

df_operations = pd.DataFrame(results)

# Display the DataFrame
print(df_operations)
```

```
      n  Jumlah Operasi  Waktu (detik)
0    10              55       0.000013
1    50            1275       0.000044
2   100            5050       0.000125
3   500          125250       0.003503
4  1000          500500       0.053896
```

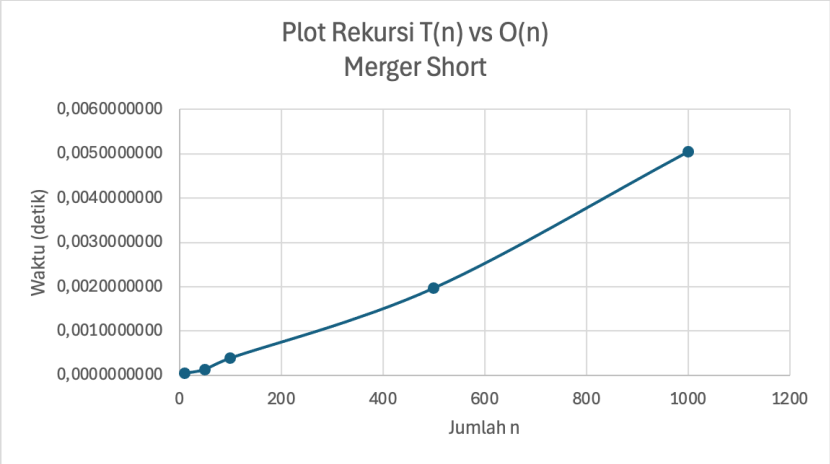
Waktu yang dibutuhkan untuk menjalankan fungsi

n	T(n) dalam detik
10	0,000013
50	0,000044
100	0,000125
500	0,003503
1000	0,053896

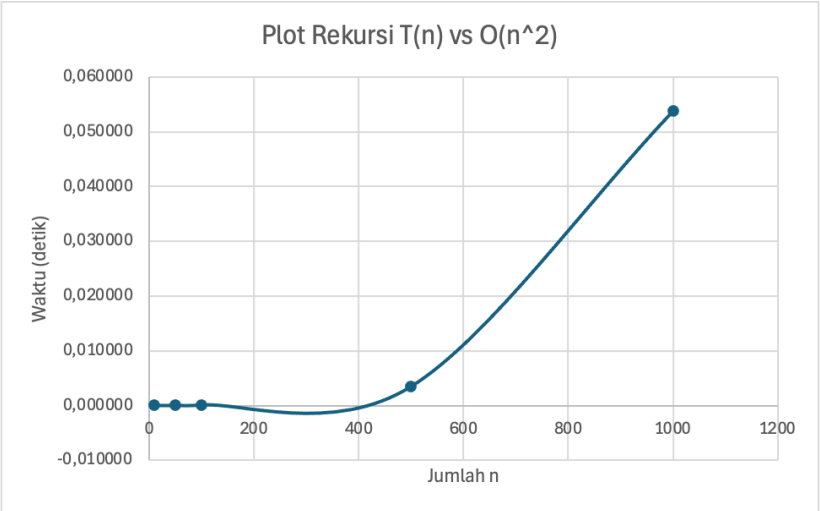
Plot Kompleksitas Rekursi O(n)



Plot Kompleksitas Rekursi O(n log n)



Plot Kompleksitas O(n^2)




Plot Kompleksitas Rekursi
0,0600000000
0,0500000000
0,0400000000
0,0300000000
0,0200000000
0,0100000000
0,0000000000
-0,0100000000
Waktu (Detik)
0
200
400
600
800
1000
1200
Jumlah (n)
0,053896
0,0050439835
0,0002210140
time linear
time n log n
time kuadran


Plot Kompleksitas Rekursi
550000
450000
350000
250000
150000
50000
-50000
-150000
Jumlah Operasi
0
200
400
600
800
1000
1200
Jumlah n
Plot Area
operasi linear
operasi n log n
operasi kuadran