TUGAS 3

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.

Membuktikan kompleksitas waktu untuk hiring problem, selection sort, merge sort dan bucket sort.

Kompleksitas Hiring Problem = O(n log n)

1. **hiring(n):** Fungsi utama yang mengiterasi kandidat sebanyak n kali. Setiap kali ada kandidat yang lebih baik dari best, ia diproses dengan fungsi hire().
2. **hire(candidate, hired):** Fungsi ini menggunakan bisect.insort, yang menyisipkan elemen dalam daftar terurut dengan kompleksitas O(log n).
3. Simulasi Waktu: Untuk beberapa nilai n, waktu eksekusi diukur untuk melihat bagaimana waktu bertambah seiring ukuran n bertambah, yang harus mengikuti **O(n log n)**.

Kompleksitas Selection Sort = $O(n^2)$

1. **Loop Eksternal (i)**: Mengulang dari awal hingga elemen kedua terakhir array (i < n - 1), di mana setiap iterasi memastikan bahwa elemen pada indeks i adalah elemen terkecil dari sisa array.
2. **Loop Internal (j)**: Mulai dari i + 1 hingga akhir array untuk mencari elemen terkecil di antara indeks i ke n-1.
3. **Penukaran (Swap)**: Setelah indeks min_index ditentukan, elemen pada arr[i] dan arr[min_index] ditukar untuk meletakkan elemen terkecil di posisi i.

Kompleksitas Merge Sort = O(n log n)

1. **Base Case**: Jika panjang arr lebih dari 1, array tersebut akan dibagi menjadi dua sub-array: left_half dan right_half.
2. Rekursi: Fungsi **merge_sort** dipanggil secara rekursif pada kedua sub-array.
3. Penggabungan **(Merge):** Dua sub-array yang telah diurutkan digabungkan menjadi satu array besar dengan memastikan elemen disusun dalam urutan yang benar.

Kompleksitas Bucket Sort = O(n)

1. **bucket_sort(arr)**: Fungsi utama yang membagi elemen-elemen arr ke dalam beberapa bucket berdasarkan rentang nilai elemen dan mengurutkan setiap bucket dengan insertion_sort.
2. **insertion_sort(arr)**: Fungsi untuk mengurutkan elemen dalam setiap bucket menggunakan Insertion Sort.
3. Pengukuran Waktu Eksekusi: Mengukur waktu eksekusi Bucket Sort untuk berbagai ukuran array dan menampilkan hasilnya dalam bentuk plot.

Penjelasan Hiring Problem

```
import bisect
import random
import time
import matplotlib.pyplot as plt
import pandas as pd

# Define the hiring function with O(log n) complexity in hire()
def hiring(n):
    best = 0
    candidates = [random.randint(1, n * 10) for _ in range(n)]
    hired = []

    for i in range(n):
        if best < candidates[i]:
            hire(candidates[i], hired)
        best = max(best, candidates[i])

def hire(candidate, hired):
    # O(log n) insertion with binary insertion
    bisect.insort(hired, candidate)

# Testing with different n values
n_values = [3 * 10**3, 5 * 10**3, 10**4, 5 * 10**4, 10**5]
times = []

for n in n_values:
    start_time = time.time()
    hiring(n)
    end_time = time.time()
    times.append(end_time - start_time)

# Plotting
plt.plot(n_values, times, marker='o')
plt.xlabel('n (number of candidates)')
plt.ylabel('Execution Time (seconds)')
plt.title('Execution Time of Hiring Function with O(n log n) Complexity')
plt.grid(True)
plt.show()

# Displaying the time results for each n value in a table format
time_data = pd.DataFrame({
    "n (number of candidates)": n_values,
    "Execution Time (seconds)": times
})
print(time_data)
```

Fungsi `hiring` memanggil `hire` dengan kompleksitas O(log n) untuk setiap elemen dalam daftar candidates, sehingga menghasilkan kompleksitas total O(n log n).

Bagian Code	Kompleksitas	Penjelasan
`candidates = [random.randint(1, n * 10) for _ in range(n)]`	O(n)	Membuat daftar kandidat dengan panjang n, di mana setiap kandidat diberi nilai acak dalam rentang tertentu.
`for i in range(n):`	O(n)	Iterasi melalui seluruh daftar kandidat, satu per satu.
`if best < candidates[i]:`	O(1)	Perbandingan konstan untuk menentukan apakah kandidat saat ini lebih baik daripada kandidat terbaik sejauh ini.
`hire(candidates[i], hired)`	O(logn)	Memanggil fungsi `hire` untuk menambahkan kandidat ke dalam daftar `hired` secara terurut.
`bisect.insort(hired, candidate)`	O(logn)	Penyisipan elemen dalam daftar `hired` yang sudah diurutkan, menggunakan pencarian biner untuk menemukan posisi penyisipan, menghasilkan kompleksitas O(logn) per operasi.

Penjelasan Selection Sort

```
import time
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Selection Sort Function
def selection_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        # Swap elements
        arr[i], arr[min_index] = arr[min_index], arr[i]

# Testing with different n values
n_values = [100, 500, 1000, 2000, 3000]
times = []

for n in n_values:
    # Generate a random array of size n
    arr = np.random.randint(0, 10000, n)
    # Measure the time for selection sort
    start_time = time.time()
    selection_sort(arr)
    end_time = time.time()
    times.append(end_time - start_time)

# Plotting the results
plt.plot(n_values, times, marker='o')
plt.xlabel('n (array size)')
plt.ylabel('Execution Time (seconds)')
plt.title('Execution Time of Selection Sort with O(n^2) Complexity')
plt.grid(True)
plt.show()

# Displaying the execution times for each array size (n) tested
time_data_selection_sort = pd.DataFrame({
    "Array Size (n)": n_values,
    "Execution Time (seconds)": times
})
print(time_data_selection_sort)
```

Kode ini menampilkan grafik waktu eksekusi untuk berbagai ukuran array (nilai n). Grafik ini seharusnya menunjukkan kenaikan eksponensial dalam waktu eksekusi seiring pertumbuhan ukuran array, yang mengonfirmasi kompleksitas $O(n^2)$.

Bagian Code	Kompleksitas	Penjelasan
`for i in range(n - 1):`	O(n)	Loop luar untuk iterasi melalui elemen-elemen dalam array, menentukan posisi di mana elemen minimum berikutnya harus ditempatkan.
`min_index = i`	O(1)	Inisialisasi indeks minimum ke posisi saat ini dalam array.
`for j in range(i + 1, n):`	O(n)	Loop dalam untuk menemukan elemen terkecil dalam sisa array. Ini berjalan sebanyak n−i iterasi pada setiap loop luar, sehingga total waktu menjadi $O(n^2)$.
`if arr[j] < arr[min_index]:`	O(1)	Membandingkan elemen saat ini dengan elemen pada indeks minimum. Ini adalah operasi konstan yang dilakukan dalam loop dalam.
`arr[i], arr[min_index] = arr[min_index], arr[i]`	O(1)	Menukar elemen pada indeks saat ini dengan elemen terkecil yang ditemukan. Operasi ini juga konstan, tetapi dieksekusi O(n) kali untuk semua elemen.

Loop luar berjalan n kali, dan loop dalam berjalan n−i kali untuk setiap iterasi, yang menghasilkan **$O(n^2)$** secara keseluruhan.

Penjelasan Merge Sort

```
import time
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Merge Sort Function
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        # Recursive calls to sort both halves
        merge_sort(left_half)
        merge_sort(right_half)

        # Merging the sorted halves
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        # Copying any remaining elements of left_half, if any
        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        # Copying any remaining elements of right_half, if any
        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1

# Testing with different n values
n_values_merge = [3 * 10**3, 5 * 10**3, 10**4, 5 * 10**4, 10**5]
times_merge = []

for n in n_values_merge:
    # Generate a random array of size n
    arr = np.random.randint(0, 10000, n)
    # Measure the time for merge sort
    start_time = time.time()
    merge_sort(arr)
    end_time = time.time()
    times_merge.append(end_time - start_time)

# Plotting the results
plt.plot(n_values_merge, times_merge, marker='o')
plt.xlabel('n (array size)')
plt.ylabel('Execution Time (seconds)')
plt.title('Execution Time of Merge Sort with O(n log n) Complexity')
plt.grid(True)
plt.show()

# Displaying the execution times for each array size (n) tested
time_data_merge_sort = pd.DataFrame({
    "Array Size (n)": n_values_merge,
    "Execution Time (seconds)": times_merge
})
print(time_data_merge_sort)
```

Kode ini menampilkan tabel yang berisi ukuran array (n) dan waktu eksekusi yang diperlukan untuk setiap ukuran, sehingga kita bisa melihat bagaimana waktu eksekusi bertambah sesuai dengan prediksi teoretis O(n log n) dari Merge Sort.

Bagian Code	Kompleksitas	Penjelasan
`mid = len(arr) // 2`	O(1)	Membagi array menjadi dua bagian. Ini adalah operasi konstan yang dilakukan sekali per pemanggilan rekursif.
`left_half = arr[:mid]` `right_half = arr[mid:]`	O(n) pada setiap tingkat rekursi	Membuat salinan array bagian kiri dan kanan. Kompleksitas salinan adalah O(n) per tingkat rekursi, karena setiap elemen harus disalin.
`merge_sort(left_half)` `merge_sort(right_half)`	O(logn) tingkat rekursi	Setiap pemanggilan rekursif membagi array menjadi dua bagian. Tingkat rekursi mencapai hingga n log n kali, di mana n adalah panjang array.
`while i <len(left_half)` `and j <len(right_half)`	O(n) pada setiap tingkat rekursi	Menggabungkan kedua bagian. Pada setiap tingkat rekursi, ini membutuhkan iterasi melalui semua elemen dalam array yang sedang diurutkan.
`while i <len(left_half)` `while j <len(right_half)`	O(n) pada setiap tingkat rekursi	Menyalin sisa elemen dari bagian kiri atau kanan jika ada yang tertinggal setelah iterasi utama. Ini memastikan semua elemen dipindahkan ke array hasil.

Penjelasan Bucket Sort

```
import random
import time
import matplotlib.pyplot as plt
import pandas as pd

# Function for Bucket Sort
def bucket_sort(arr):
    if len(arr) == 0:
        return arr

    # Determine minimum and maximum values in the array
    min_value = min(arr)
    max_value = max(arr)

    # Bucket count and size
    bucket_count = len(arr)
    buckets = [[] for _ in range(bucket_count)]

    # Distribute elements into buckets
    for num in arr:
        index = int((num - min_value) / (max_value - min_value + 1) * (bucket_count - 1))
        buckets[index].append(num)

    # Sort each bucket and merge
    sorted_array = []
    for bucket in buckets:
        insertion_sort(bucket)
        sorted_array.extend(bucket)

    return sorted_array

# Helper function for insertion sort
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

# Testing with different n values
n_values_bucket = [10**3, 5 * 10**3, 10**4, 5 * 10**4, 10**5]
times_bucket = []

for n in n_values_bucket:
    # Generate a random array of size n
    arr = [random.uniform(0, 10000) for _ in range(n)]
    # Measure the time for bucket sort
    start_time = time.time()
    bucket_sort(arr)
    end_time = time.time()
    times_bucket.append(end_time - start_time)

# Plotting the results
plt.plot(n_values_bucket, times_bucket, marker='o')
plt.xlabel('n (array size)')
plt.ylabel('Execution Time (seconds)')
plt.title('Execution Time of Bucket Sort with O(n + k) Complexity')
plt.grid(True)
plt.show()

# Displaying the execution times for each array size (n) tested
time_data_bucket_sort = pd.DataFrame({
    "Array Size (n)": n_values_bucket,
    "Execution Time (seconds)": times_bucket
})
print(time_data_bucket_sort)
```

Distribusi elemen dalam bucket idealnya merata, sehingga Insertion Sort pada tiap bucket efisien (mendekati O(n)).

Bagian Code	Kompleksitas	Penjelasan
`min_value = min(arr)`	O(n)	Mencari nilai minimum dalam array membutuhkan iterasi melalui semua elemen dalam array.
`max_value = max(arr)`	O(n)	Sama seperti min, mencari nilai maksimum juga memerlukan iterasi melalui semua elemen dalam array.
`buckets = [ [] for _ in range(bucket_count) ]`	O(b) (biasanya dianggap O(n))	Membuat bucket dengan jumlah yang sama dengan ukuran array. Biasanya bbb dianggap mendekati nnn, sehingga kompleksitasnya mendekati O(n).
`Loop for num in arr: ...`	O(n)	Membagi setiap elemen ke dalam bucket sesuai dengan indeks yang dihitung. Ini melibatkan satu iterasi penuh melalui semua elemen array.
`Loop for bucket in buckets: ...`	O(n) dalam kasus rata-rata, tetapi $O(n^2)$ dalam kasus terburuk	Setiap bucket diurutkan menggunakan Insertion Sort. Jika semua elemen berada dalam satu bucket (kasus terburuk), kompleksitas menjadi $O(n^2)$, tetapi rata-rata mendekati O(n) jika bucket seimbang.

Kompleksitas Hiring Problem = O(n log n)

```python
import bisect
import random
import time
import matplotlib.pyplot as plt
import pandas as pd

# Define the hiring function with O(log n) complexity in hire()
def hiring(n):
    best = 0
    candidates = [random.randint(1, n * 10) for _ in range(n)]
    hired = []

    for i in range(n):
        if best < candidates[i]:
            hire(candidates[i], hired)
        best = max(best, candidates[i])

def hire(candidate, hired):
    # O(log n) insertion with binary insertion
    bisect.insort(hired, candidate)

# Testing with different n values
n_values = [10**3, 5 * 10**3, 10**4, 5 * 10**4, 10**5]
times = []

for n in n_values:
    start_time = time.time()
    hiring(n)
    end_time = time.time()
    times.append(end_time - start_time)

# Plotting
plt.plot(n_values, times, marker='o')
plt.xlabel('n (number of candidates)')
plt.ylabel('Execution Time (seconds)')
plt.title('Execution Time of Hiring Function with O(n log n) Complexity')
plt.grid(True)
plt.show()
```

Waktu yang dibutuhkan untuk menjalankan fungsi

Number of Candidates	Waktu (detik)
3000	0.002143
5000	0.012656
10000	0.016199
50000	0.065999
100000	0.098946

Kompleksitas Selection Sort = $O(n^2)$

```python
import time
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Selection Sort Function
def selection_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        # Swap elements
        arr[i], arr[min_index] = arr[min_index], arr[i]

# Testing with different n values
n_values = [100, 500, 1000, 2000, 3000]
times = []

for n in n_values:
    # Generate a random array of size n
    arr = np.random.randint(0, 10000, n)
    # Measure the time for selection sort
    start_time = time.time()
    selection_sort(arr)
    end_time = time.time()
    times.append(end_time - start_time)

# Plotting the results
plt.plot(n_values, times, marker='o')
plt.xlabel('n (array size)')
plt.ylabel('Execution Time (seconds)')
plt.title('Execution Time of Selection Sort with O(n^2) Complexity')
plt.grid(True)
plt.show()
```

Waktu yang dibutuhkan untuk menjalankan fungsi

Array Size (n)	Waktu (detik)
100	0.001865
500	0.074750
1000	0.170721
2000	0.530680
3000	1.138418

Kompleksitas Merge Sort = O(n log n)

```python
import time
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Merge Sort Function
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        # Recursive calls to sort both halves
        merge_sort(left_half)
        merge_sort(right_half)

        # Merging the sorted halves
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        # Copying any remaining elements of left_half, if any
        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        # Copying any remaining elements of right_half, if any
        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1

# Testing with different n values
n_values_merge = [10**3, 5 * 10**3, 10**4, 5 * 10**4, 10**5]
times_merge = []
```

Waktu yang dibutuhkan untuk menjalankan fungsi

Array Size (n)	Waktu (detik)
3000	0.035471
5000	0.106400
10000	0.097764
50000	0.454301
100000	0.832151

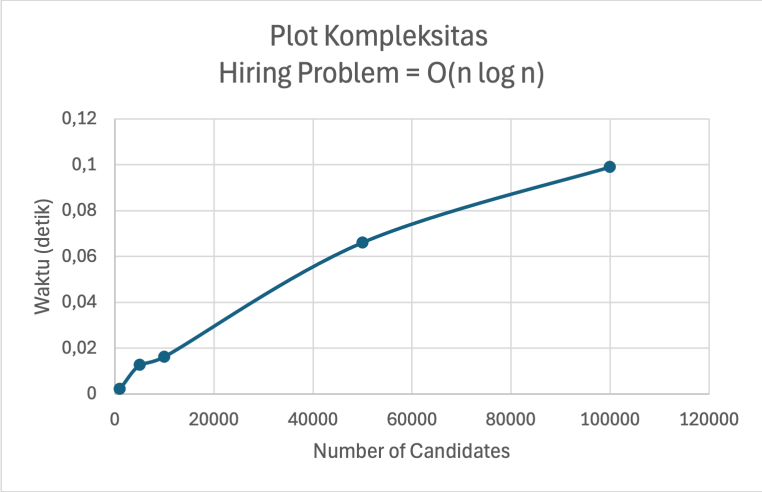
Kompleksitas Bucket Sort= O(n)

```python
# Function for Bucket Sort
def bucket_sort(arr):
    if len(arr) == 0:
        return arr

    # Determine minimum and maximum values in the array
    min_value = min(arr)
    max_value = max(arr)

    # Bucket count and size
    bucket_count = len(arr)
    buckets = [[] for _ in range(bucket_count)]

    # Distribute elements into buckets
    for num in arr:
        index = int((num - min_value) / (max_value - min_value + 1) * (bucket_count - 1))
        buckets[index].append(num)

    # Sort each bucket and merge
    sorted_array = []
    for bucket in buckets:
        insertion_sort(bucket)
        sorted_array.extend(bucket)

    return sorted_array

# Helper function for insertion sort
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

# Testing with different n values
n_values_bucket = [10**3, 5 * 10**3, 10**4, 5 * 10**4, 10**5]
times_bucket = []
```

Waktu yang dibutuhkan untuk menjalankan fungsi

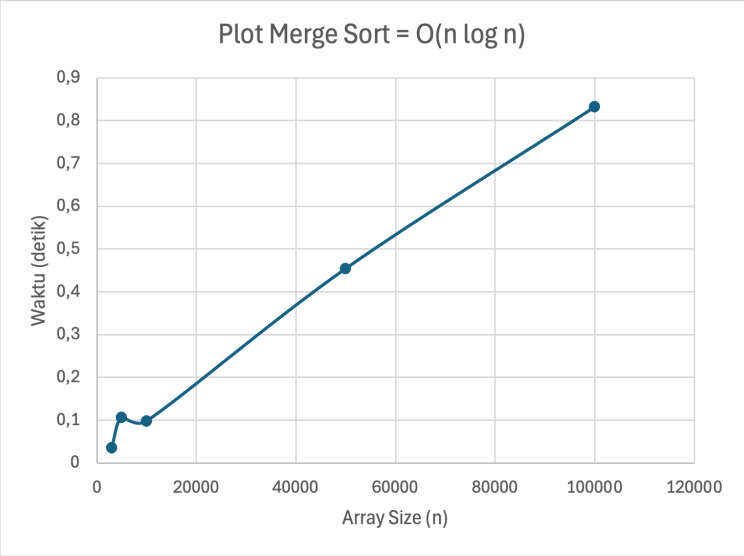
Array Size (n)	Waktu (detik)
3000	0.005079
5000	0.006648
10000	0.012478
50000	0.098200
100000	0.202243

Plot Kompleksitas Hiring Problem = O(n log n)



Plot Kompleksitas Selection Sort = O(n^2)

Plot Kompleksitas Merge Sort = O(n log n)



Plot Kompleksitas Bucket = O(n)