LAPORAN TUGAS

DESAIN DAN ANALISIS ALGORITMA

KOMPLEKSITAS ALGORITMA SORTING BUBBLE SORT DAN MERGE SORT

Disusun oleh:

Muhammad Rahaji Jhaerol
19102144
S1 IF 07 SC1

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO

2022

Dasar Teori

A. Bubble Sort

Bubble sort adalah algoritma pengurutan sederhana. Algoritma pengurutan ini adalah algoritma berbasis perbandingan di mana setiap pasangan elemen yang berdekatan dibandingkan dan elemen-elemen tersebut ditukar jika tidak berurutan. Algoritme ini tidak cocok untuk kumpulan data besar karena kompleksitas kasus rata-rata dan terburuknya adalah (n2) di mana n adalah jumlah item[1].

B. Merge Sort

Merge sort adalah metode pengurutan yang menggunakan pola divide and conquer. Strateginya adalah dengan membagi sekelompok data yang akan diurutkan menjadi beberapa kelompok kecil terdiri dari maksimal dua nilai untuk dibandingkan dan digabungkan lagi secara keseluruhan.

Implementasi

A. Bubble Sort

Bubble Sort adalah metode pengurutan algoritma dengan cara melakukan penukaran data secara terus menerus sampai bisa dipastikan dalam suatu iterasi tertentu tidak ada lagi perubahan/penukaran. Algoritma ini menggunakan perbandingan dalam operasi antar elemennya[2].

Berikut ini adalah gambaran dari algoritma bubble sort:

1. Bandingkan nilai data ke-1 dan data ke-2
2. Jika data ke-1 lebih besar dari data ke-2 maka tukar posisinya
3. Kemudian data yg lebih besar tadi dibandingkan dengan data ke-3
4. Lakukan langkah nomer 2 hingga selesai.

Pseudocode in python

```
function bubbleSort(elements):
  # Perulangan dari ukuran array dari indeks terakhir[-1] ke indeks [0]
  for n in range(len(elements)-1, 0, -1):
    for i in range(n):
      if elements[i] > elements[i + 1]:
        # menukar data jika elemennya kurang dari elemen berikutnya
dalam array
        elements[i], elements[i + 1] = elements[i + 1], elements[i]
      end if
    end for
  end for
end function
```

B. Merge Sort

Algoritma pengurutan data merge sort dilakukan dengan menggunakan cara divide and conquer yaitu dengan memecah kemudian menyelesaikan setiap bagian kemudian menggabungkannya kembali. Pertama data dipecah menjadi 2 bagian dimana bagian pertama merupakan setengah (jika data genap) atau setengah minus satu (jika data ganjil) dari seluruh data, kemudian dilakukan pemecahan kembali untuk masing-masing blok sampai hanya terdiri dari satu data tiap blok.

Setelah itu digabungkan kembali dengan membandingkan pada blok yang sama apakah data pertama lebih besar daripada data ke-tengah+1, jika ya maka data ke-tengah+1 dipindah sebagai data pertama, kemudian data ke-pertama sampai ke-tengah digeser menjadi data ke-dua sampai ke-tengah+1, demikian seterusnya sampai menjadi satu blok utuh seperti awalnya. Sehingga metode merge sort merupakan metode yang membutuhkan fungsi rekursi untuk penyelesaiannya.

Pseudocode in python[3]

```
function merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m

    # buat temp arrays
    L = [0] * (n1)
    R = [0] * (n2)

    # Salin data ke temp arrays L[] and R[]
```

```
    for i in range(0, n1):
        L[i] = arr[l + i]
    end for

    for j in range(0, n2):
        R[j] = arr[m + 1 + j]
    end for

    # Merge temp arrays kembali ke arr[l..r]
    i = 0     # inisialisasi index pertama dari subarray
    j = 0     # inisialisasi index kedua dari subarray
    k = l     # Inisialisasi index merged dari subarray

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        end if
        else:
            arr[k] = R[j]
            j += 1
        k += 1
    end while

    # Salin elemen L[] yang tersisa, jika ada
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1
    end while
    # Salin elemen R[] yang tersisa, jika ada
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1
    end while
# l adalah untuk indeks kiri dan r adalah indeks kanan dari
# sub-array arr yang akan diurutkan
end function

function mergeSort(arr, l, r):
    if l < r:

        # Sama seperti (l+r)//2, tetapi menghindari overflow untuk
        # besar l dan h
        m = l+(r-l)//2

        # Urutkan bagian pertama dan kedua
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)
    end if
end function
```

Screenshot Program


```
# Muhammad Rahaji Jhaerol
# 19102144
# S1IF07SC1
# Kompleksitas Algoritma Sorting
# Bubble sort dan Merge sort
```

```
# spesifikasi hardware
# RAM: 8192MB
# CPU: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz (8 CPUs), ~1.8GHz
# OS: Windows 11 Home Single Language 64-bit (10.0, Build 22000) (22000.co_release.2106
# Bahasa Pemrograman: Python
# IDE: Google Colaboratory
```

```
# library yang digunakan
from random import randint # set nilai random
import time # set waktu
```

```
# deklarasi array kosong untuk menampung nilai time dan n
arrTimeBubbleSort = []
arrTimeMergeSort = []
arrNsort = []
```

```
# deklarasi array kosong untuk menampung nilai dari bubble and merge
arrBubble = []
arrMerge = []
# inputan nilai n sebagai panjang array/banyak data
n = int(input("Masukan nilai n : "))
b = 0
# perulangan dengan range n
for j in range(n):
  # inisialisasi data dengan nilai random dari 1-100
  data = randint(0, 100)
  b = data
  # menambahkan data ke array bubble/merge
  arrBubble.append(b)
  arrMerge.append(b)
```

```
Masukan nilai n : 1000
```

```
# Algoritma Sorting Bubble Sort
# reference: https://www.geeksforgeeks.org/python-program-for-bubble-sort/
def bubbleSort(elements):
  # Perulangan dari ukuran array dari indeks terakhir[-1] ke indeks [0]
  for n in range(len(elements)-1, 0, -1):
    for i in range(n):
      if elements[i] > elements[i + 1]:
        # menukar data jika elemennya kurang dari elemen berikutnya dalam array
        elements[i], elements[i + 1] = elements[i + 1], elements[i]
```

```
# Algoritma Merge Sort
```

```
# reference: https://www.geeksforgeeks.org/python-program-for-merge-sort/
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m

    # buat temp arrays
    L = [0] * (n1)
    R = [0] * (n2)

    # Salin data ke temp arrays L[] and R[]
    for i in range(0, n1):
        L[i] = arr[l + i]

    for j in range(0, n2):
        R[j] = arr[m + 1 + j]

    # Merge temp arrays kembali ke arr[l..r]
    i = 0     # inisialisasi index pertama dari subarray
    j = 0     # inisialisasi index kedua dari subarray
    k = l     # Inisialisasi index merged dari subarray

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
            k += 1

    # Salin elemen L[] yang tersisa, jika ada
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    # Salin elemen R[] yang tersisa, jika ada
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

# l adalah untuk indeks kiri dan r adalah indeks kanan dari
# sub-array arr yang akan diurutkan


def mergeSort(arr, l, r):
    if l < r:

        # Sama seperti (l+r)//2, tetapi menghindari overflow untuk
        # besar l dan h
        m = l+(r-l)//2

        # Urutkan bagian pertama dan kedua
        mergeSort(arr, l, m)
```

```
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)
```

```
# set waktu start
start_time_bubble = time.time()
print("Unsorted Array Bubble :")
print(arrBubble)
# panggil fungsi bubble sort
bubbleSort(arrBubble)
# waktu eksekusi
time1 = time.time() - start_time_bubble
# menambahkan waktu eksekusi bubble ke array
arrTimeBubbleSort.append(time1)
# menambahkan nilai n ke array
arrNsort.append(n)
print("Sorted Array Bubble :")
print(arrBubble)
print("runtime : {} second".format(time1))
```

```
Unsorted Array Bubble :
[60, 40, 70, 8, 44, 56, 23, 49, 10, 55, 86, 61, 0, 1, 18, 55, 98, 96, 26, 87, 60, 84,
Sorted Array Bubble :
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
runtime : 0.1079862117767334 second
```

```
# set waktu start
start_time_merge = time.time()
print("Unsorted Array Merge :")
print(arrMerge)
# panggil fungsi merge sort
mergeSort(arrMerge, 0, n-1)
# waktu eksekusi
time2 = time.time() - start_time_merge
# menambahkan waktu eksekusi merge ke array
arrTimeMergeSort.append(time2)
print("Sorted Array Merge :")
print(arrMerge)
print("runtime : {} second".format(time2))
```

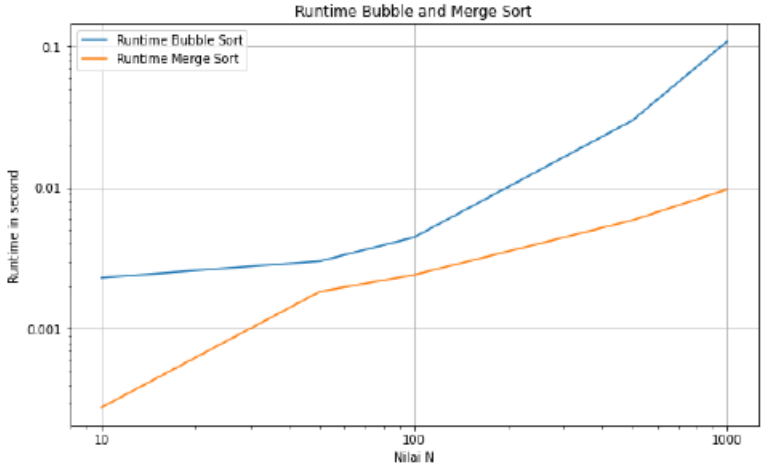
```
Unsorted Array Merge :
[60, 40, 70, 8, 44, 56, 23, 49, 10, 55, 86, 61, 0, 1, 18, 55, 98, 96, 26, 87, 60, 84,
Sorted Array Merge :
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
runtime : 0.009698867797851562 second
```

```
# library untuk menampilkan grafik
import matplotlib.pyplot as plt
from matplotlib.ticker import FormatStrFormatter
from matplotlib.ticker import ScalarFormatter

plt.figure(figsize=(10,6))
```

```
print("Runtime Bubble Sort : ", end="")
print(arrTimeBubbleSort)
print("Runtime Merge Sort : ", end="")
print(arrTimeMergeSort)
print(arrNsort)
plt.xlabel("Nilai N")
plt.ylabel("Runtime in second")
plt.title("Runtime Bubble and Merge Sort")
plt.plot(arrNsort, arrTimeBubbleSort, label = "Runtime Bubble Sort")
plt.plot(arrNsort, arrTimeMergeSort, label = "Runtime Merge Sort")
plt.yscale("log")
plt.xscale("log")
plt.gca().yaxis.set_major_formatter(FormatStrFormatter('%.5g'))
plt.gca().xaxis.set_major_formatter(FormatStrFormatter('%.5g'))
plt.legend(loc="upper left")
plt.grid(True)
plt.show()
```

```
Runtime Bubble Sort : [0.0023102760314941406, 0.0030286312103271484, 0.0044634342193
Runtime Merge Sort : [0.0002791881561279297, 0.0018436908721923828, 0.00242161750793
[10, 50, 100, 500, 1000]
```



Pengujian

Data yang digunakan dalam pengujian ini yaitu menggunakan data random dari 1-100 menggunakan library python ‘from random import randint’. Nilai random tersebut dimasukan ke dalam array kosong dan dilakukan looping sebanyak n inputan yang di masukan. Misal nilai n yang di input 10 maka akan menyimpan 10 nilai/indeks random dalam array. Inputan n atau jumlah data yang diuji dan nilainya sama antara kedua algoritma sorting bubble sort dan merge sort.

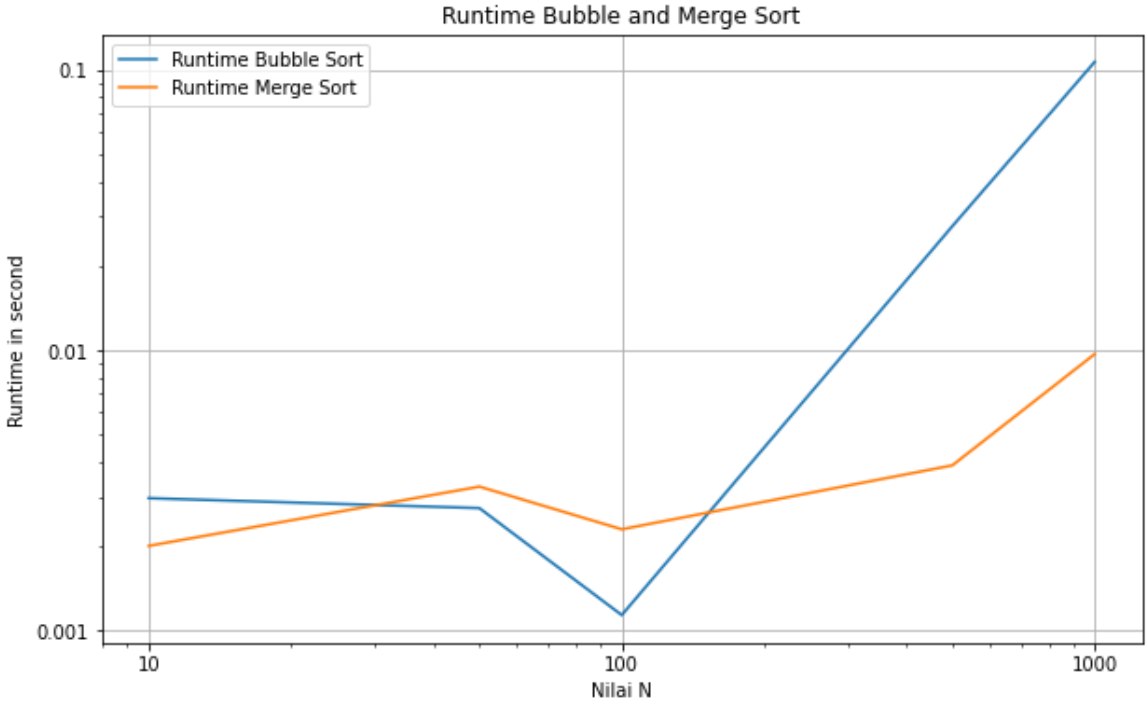
Perbandingan waktu eksekusi untuk masing-masing algoritma dengan n inputan

n	Waktu Eksekusi Bubble Sort	Waktu Eksekusi Merge Sort
10	0.0029685497283935547	0.0020067691802978516
50	0.002736330032348633	0.003264904022216797
100	0.0011367797851562 5	0.0022995471954345703
500	0.027702808380126953	0.0038864612579345703
1000	0.10679221153259277	0.009696006774902344

```
Unsorted Array Bubble :
[12, 76, 8, 50, 62, 69, 56, 47, 23, 98]
Sorted Array Bubble :
[8, 12, 23, 47, 50, 56, 62, 69, 76, 98]
runtime : 0.0029685497283935547 second

Unsorted Array Merge :
[12, 76, 8, 50, 62, 69, 56, 47, 23, 98]
Sorted Array Merge :
[8, 12, 23, 47, 50, 56, 62, 69, 76, 98]
runtime : 0.0020067691802978516 second
```

Gambar 1 Output program dengan 10 data



Gambar 2 Grafik perbandingan waktu eksekusi

Analisis Hasil Pengujian

A. Bubble Sort

Cara kerja bubble sort adalah dengan berulang-ulang melakukan traversal (proses looping) terhadap elemen-elemen struktur data yang belum diurutkan. Di dalam traversal tersebut, nilai dari dua elemen struktur data dibandingkan. Jika ternyata urutannya tidak sesuai dengan "pesanan", maka dilakukan pertukaran (swap). Algoritma sorting ini disebut juga dengan comparison sort dikarenakan hanya mengandalkan perbandingan nilai elemen untuk mengoperasikan elemennya.

Dalam proses bubble sort dapat ditulis:

$$T(n) = (N-1) + (N-2) + \cdots + 2 + 1 = \sum_{i-1}^{n-1} n - 1 = \frac{n(n-1)}{2}$$

n	$\boldsymbol{T(n) = \frac{n(n-1)}{2}}$	$\boldsymbol{n^2}$
10	45	100
50	1225	2.500
100	4950	10.000
500	124750	25.0000
1000	499500	1.000.000

Tabel 1 Perhitungan kompleksitas algoritma bubble sort

Dilihat dari tabel 2 diatas, semakin banyak data yang digunakan maka kompleksitas algoritma akan semakin besar.

B. Merge Sort

Kompleksitas algoritma merge sort adalah O(n log n). Secara umum, algoritma merge sort dapat diimplementasikan secara rekursif. Fungsi rekursif adalah sebuah fungsi yang didalam implementasinya memanggil dirinya sendiri[4].

Rumus merge sort dapat ditulis:

$$T(n) = 2T\frac{n}{2} + n = (n \log n)$$

n	$\boldsymbol{T(n) = 2T\frac{n}{2} + n}$	$\boldsymbol{n \log n}$
10	10T+10	10
50	50T+50	84,9485
100	100T+100	200
500	500T+500	1.349,4850
1000	1000T+1000	3.000

Tabel 2 Perhitungan kompleksitas algoritma merge sort

Dilihat dari tabel 3 diatas, semakin banyak data yang digunakan maka kompleksitas algoritma akan semakin besar.

C. Kesimpulan

Dari analisis hasil pengujian dapat dilihat bahwa pada kedua algoritma sorting bubble sort dan merge sort memiliki waktu komputasi kompleksitas yang semakin besar jika

data yang diinputkan semakin besar juga. Sehingga dapat dikatakan bahwa kompleksitas berbanding lurus dengan banyaknya data.

Dari kedua algoritma sorting tersebut dapat dilihat bahwa kompleksitas algoritma merge sort lebih baik dibandingkan bubble sort karena pada algoritma merge sort menghasilkan waktu eksekusi yang lebih cepat dari pada bubble sort. Pada tabel perhitungan kompleksitas juga menunjukkan bahwa algoritma merge sort memiliki hasil yang lebih baik dari pada algoritma bubble sort.

Referensi

[1] Tutorialspoint, “Data Structure - Bubble Sort Algorithm.” https://www.tutorialspoint.com/data_structures_algorithms/bubble_sort_algorithm (accessed Apr. 25, 2022).

[2] P. A. Rahayuningsih, “Analisis Perbandingan Kompleksitas Algoritma Pengurutan Nilai (Sorting),” *Evolusi*, vol. 4, no. 2, pp. 64–75, 2016, [Online]. Available: https://ejournal.bsi.ac.id/ejurnal/index.php/evolusi/article/view/702/577.

[3] GeeksforGeeks, “Python Program for Merge Sort.” https://www.geeksforgeeks.org/python-program-for-merge-sort/ (accessed Apr. 24, 2022).

[4] W. F. Wisudawan, “Kompleksitas Algoritma Sorting yang Populer Dipakai,” pp. 1–8, 2006.