# TUGAS 1

# DESAIN DAN ANALISIS ALGORITMA

# PERBANDINGAN KOMPLEKSITAS ALGORITMA PENGURUTAN BUBBLE & MERGE

**Dosen Pengampu:**

Trihastuti Yuniati, S.Kom., M.T.


**Oleh:**

Andre Citro Febriliyan Lanyak

19102274

S1IF-07-SC1



**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS INFORMATIKA**

**INSTITUT TEKNOLOGI TELKOM PURWOKERTO**


**2022**

## 1. Dasar Teori

### 1.1 Bubble Sort

Bubble sort, kadang-kadang disebut sebagai sinking sort, adalah sebuah algoritme pengurutan sederhana yang berulang kali menelusuri *list*, membandingkan elemen yang berdekatan dan menukarnya jika urutannya salah. Dimana Bubble sort akan melintasi list berulang sampai list diurutkan. Bubble sort, termasuk kedalam jenis algoritma *comparison-based sorting*. Bubble sort dinamai berdasarkan cara kerajnya dimana elemen yang lebih kecil atau lebih besar (tergantung ascending atau descending) mengapung ke atas list seperti sebuah "gelembung" air.[1]

### 1.2 Merge Sort

Merge sort adalah sebuah algoritma pengurutan yang efisien, bertujuan umum, dan berjenis *comparison-based sorting*. Sebagian besar implementasi menghasilkan pengurutan yang stabil, yang berarti bahwa urutan elemen yang setara adalah sama dalam input dan output. Merge sort adalah algoritma divide-and-conquer yang ditemukan oleh John von Neumann pada tahun 1945. Secara konseptual Merge sort berfungsi sebagai berikut:

1. Bagi list yang tidak disortir menjadi n sublist, dimana masing-masing berisi satu elemen (listsatu elemen dianggap sudah diurutkan).
2. Gabungkan sublist berulang kali untuk menghasilkan sublist baru yang diurutkan hingga hanya tersisa satu sublist. Ini akan menjadi daftar yang diurutkan.[2]

## 2. Implementasi

### 2.1 Spesifikasi Hardware dan Software

2.1.1 RAM : 16384 MB

2.1.2 CPU : AMD Ryzen 7 5800H With Radeon Graphics (16 CPUs), ~3.2GHz

2.1.3 OS : Windows 10 Home Single Language 64-bit (10.0, Build 19044)

2.1.4 Bahasa Pemrograman : Python

2.1.5 IDE : Visual Studio Code

## 2.2 Bubble Sort

### 2.2.1 Pseudocode

```
//Deklarasi Fungsi/Prosedur dari Bubble sort
procedure bubbleSort(A : list of sortable items)

    //Deklarasi variable n berisi panjang dari array A
    n := length(A)

    //Lakukan Perulangan sehingga array terurutkan
    repeat
        swapped := false
        //Lakukan Perulangan sebanyak jumlah N-1
        inclusive
        for i := 1 to n-1 inclusive do
            /* if this pair is out of order */
            //Lakukan pertukaran jika elemen lebih besar
            if A[i-1] > A[i] then
                /* swap them and remember something
                changed */
                swap(A[i-1], A[i])
                swapped := true
            end if
        end for
    until not swapped
end procedure
```

### 2.2.2 Screenshot Program

```
def sortbubble(arr):
    #Deklarasi variabel n yang berisi panjang dari array
    n = len(arr)

    #Iterasi seluruh elemen dari array
    for i in range(n-1):
        #Lalui array 0 sampai n-i-1
        for j in range(0, n-i-1):
            #Tukar posisi jika elemen yang ditemukan lebih
            besar
            #Lalu lanjutkan ke elemen selanjutnya
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

## 2.3 Merge Sort

### 2.3.1 Pseudocode

```
//Deklarasi Fungsi/Prosedur dari Merge sort
procedure MergeSort(A : list of sortable items)

    //Deklarasi list kosong left dan right
    left, right := list

    //Percabangan untuk mengantisipasi inputan cuma 1
    element
    if length(A)≤ do
        return A
    else do
        //Deklarasi varaibel middle yang merupakan elemen
        paling tengah atau median dari list
        middle := length(A) / 2

        //Perulangan untuk memasukkan value yang ada di
        kanan dan kiri dari middle ke listnya masing
        masing
        for each x in A up to middle do
            add x to left
        for each x in A after middle do
            add x to right

        //Panggilkembali fungsi MergeSort secara rekursif
        untuk meyelesi masalah perurutan ini
        left :=  MergeSort(left)
        right :=  MergeSort(right)

        //Gabungkan list left dan right dan return value
        dari result yang merupakan list terurut
        result := Merge(left, right)
    end if
    return result
end procedure
```

### 2.3.2 Screenshot Program

```
#Deklarasi fungsi mergeSort
def mergeSort(array):
    if len(array) > 1:

        #Deklarasi varaiabel middle yang merupakan nilai
        median dari jumlah element dari array
        middle = len(array)//2

        #Deklarai subarray L(left) dan R(right) berisi data
        dari sebelah kiri dan kanan dari element middle
```

```
        L = array[:middle]
        R = array[middle:]

        #Secara rekursif panggil fungsi mergeSort untuk
mengurutkan array L dan R
        mergeSort(L)
        mergeSort(R)

        #Deklarasi varaiabel i, j, k
        i = j = k = 0

        #Sampai kita mencapai salah satu ujung L atau R,
        pilih yang lebih besar di
        #antara elemen L dan R dan tempatkan mereka di
        posisi yang benar di A[p..r]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                array[k] = L[i]
                i += 1
            else:
                array[k] = R[j]
                j += 1
            k += 1

        # Saat kita kehabisan elemen di L atau M, ambil
        elemen
        # yang tersisa dan masukkan ke A[p..r]
        while i < len(L):
            array[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            array[k] = R[j]
            j += 1
            k += 1
```

## 2.4 Fungsi Main

### 2.4.1 Screenshot Program

#### 2.4.1.1 Fungsi plotGraph

```
#Deklarasi Fungis untuk membuat grafik
def plotGraph(bubbleTime, mergeTime, arrLen):
    plt.figure(figsize=(10,6))
    print("Runtime Bubble Sort : ", end="")
```

```
    print(bubbleTime)
    print("Runtime Merge Sort : ", end="")
    print(mergeTime)
    print(arrLen)
    plt.xlabel("Nilai N")
    plt.ylabel("Runtime in second")
    plt.title("Runtime Bubble and Merge Sort")
    plt.plot(arrLen, bubbleTime, label = "Runtime Bubble
                                          Sort")
    plt.plot(arrLen, mergeTime, 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()
```

**2.4.1.2 Fungsi writeTxt**

```
#Deklarasi Fungsi untuk membuat file.txt
def writeTxt(lists):
    for count, list in enumerate(lists):
        fileName = "file{}.txt".format(count)
        with open(fileName, 'w') as f:
            f.write(str(list))
```

**2.4.1.3 Fungsi Main**

```
#Fungsi main
if __name__ == '__main__':
    #Deklarasi list kosong untuk menampung data-data yang
akan digunakan
    bubbleTotalRuntime = []
    mergeTotalRuntime = []
    nArr = []
    dataArr = []

    #Deklarasi perulagan while untuk menerima masukan user
    berupa jumlah element untuk mempopulasi list acak
    status = True
    while status == True:
        # Populasi kan array dengan 1000 angka random
        n = input("Masukan nilai n (untuk berhenti masukkan
```

```
                N): ")
        if n == "N":
            status = False
            break

        nArr.append(int(n))
        arr = populate(int(n))
        dataArr.append(arr)

        #Deklarasi arrLen berisi panjang dari list arr
        arrLen = len(arr)

        #Copy list arr ke varabel bubbleArr dan mergerArr
        bubbleArr = arr.copy()
        mergerArr = arr.copy()

        #Panggil Bubble dan Merge sort dan hitung runtime
        nya.
        bStart = time.time()
        sortbubble(bubbleArr)
        bEnd = time.time()

        mStart = time.time()
        mergeSort(mergerArr)
        mEnd = time.time()

        #Hitung Runtime nya
        bubbleTime = bEnd - bStart
        mergeTime = mEnd - mStart

        #Append hasil runtime ke dalam list
        bubbleTotalRuntime.append(bubbleTime)
        mergeTotalRuntime.append(mergeTime)

        print(f"Runtime of the Bubble sort is {bubbleTime}")
        print(f"Runtime of the Merge sort is {mergeTime}")

    print(bubbleTotalRuntime, mergeTotalRuntime)

    #Plot grafik untuk membandingkan runtime nya
    plotGraph(bubbleTotalRuntime, mergeTotalRuntime, nArr)
    #Panggil Fungis wirteTxt untuk menyimpan array acak yang
    digunakan
    writeTxt(dataArr)
```

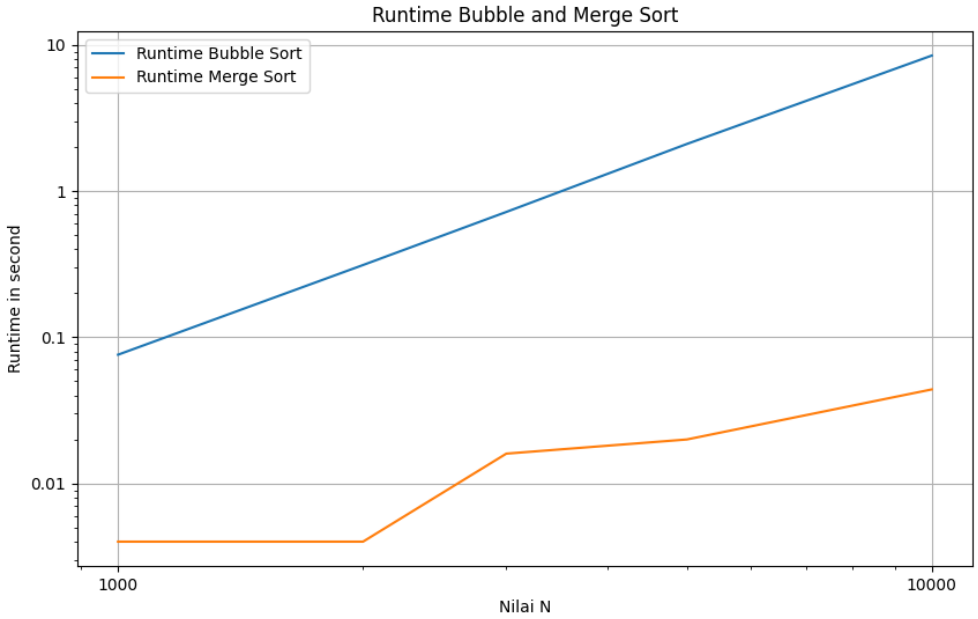
## 3. Pengujian

Pada pengujian kali ini saya mengguanakan bahasa pemrograman Python dengan bantuan library numpy untuk membuat list acak. Tujuan dari pengujian kali ini adalah membandingkan kecepatan runtime dari buah algoritma pengurutan, algortima Bubble sort dan Merge sort. Pengujian kali ini data yang digunakan adalah data acak yang dibuat dengan menggunakan fungsi randint yang merupakan fungsi bawaan numpy, dengan jumlah data sebanyak 1000, 2000, 3000, 5000, dan 10000 untuk total 5 sample. Isi data acak tiap sample adalah sama dengan nol sampai dengan jumlah total data, contohnya jika total data 1000 maka range data nya mulai dari nol sampai 999 (0-999).

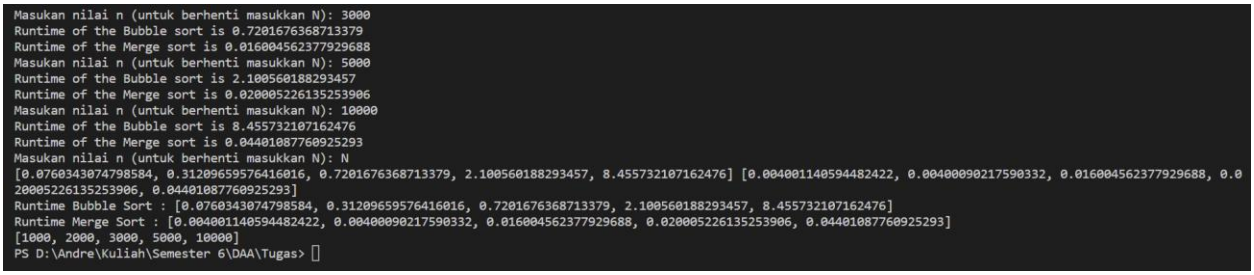
**Tabel 1. Perbandingan algoritma untuk tiap sample**

| n | Waktu runtime Bubble Sort(detik) | Waktu runtime Merge Sort(detik) |
|---|---|---|
| 1000 | 0.0760343074798584 | 0.004001140594482422 |
| 2000 | 0.31209659576416016 | 0.00400090217590332 |
| 3000 | 0.7201676368713379 | 0.016004562377929688 |
| 5000 | 2.100560188293457 | 0.020005226135253906 |
| 10000 | 8.455732107162476 | 0.04401087760925293 |

**Gambar 1. Grafik perbandingan hasil hasil eksekusi**



**Gambar 2. Hasil run program**




## 4. Analisis Hasil Pengujian

### 4.1 Bubble Sort

Bubble sort terdiri dari dua buah perulangan. Dimana bubble sort berulang-ulang kali melakukan perulangan/traversal ke komponen-komponen array yang belum diurutkan. Didalam perulangan tersebut bubble sort akan melakukan perbandingan antara dua buah elemen yang bersampingan, jika elemen yang disebelah kanan lebih besar maka posisi kedua elemen tersebut akan ditukar(swap).

$$T(n) = (N-1)(N-2) + \cdots + 2 + 1$$

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

**Tabel 2. Perhitungan kompleksitas algoritma Bubble sort**

| n | $T(n) = \frac{n(n-1)}{2}$ | $n^2$ |
|---|---|---|
| 1000 | 499500 | 1000000 |
| 2000 | 1999000 | 4000000 |
| 3000 | 4498500 | 9000000 |
| 5000 | 12497500 | 25000000 |
| 10000 | 49995000 | 100000000 |

### 4.2 Merge Sort

Merge Sort memiliki kompleksitas O(n log n). Merge sort umumnya diimplementasikan secara rekursif. Dimana secara rekursif merge sort akan di panggil untuk memecah array menjadi dua subarray hingga array tidak dapat di pecah lagi. Kemudian subarray pecahan akan di dibandingkan untuk mengurutkan array nya.

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

**Tabel 3. Perhitungan kompleksitas algoritma Merge sort**

| n | $T(n) = 2T\frac{n}{2} + n$ | $n \log n$ |
|---|---|---|
| 1000 | 1000T+1000 | 3000 |
| 2000 | 2000T+2000 | 6602.06 |
| 3000 | 3000T+3000 | 10431.36 |
| 5000 | 5000T+5000 | 18494.85 |
| 10000 | 10000T+10000 | 40000 |

### 4.3 Kesimpulan

Dapat dilihat dari hasil hasil pengujian Merge sort jauh lebih cepat dibandingkan Bubble sort. Hasil perhitungan kompleksitas masing masing algoritma juga menghasilkan hasil yang kurang lebih selaras dengan Gambar 1. Selain itu seiring jumlah data bertambah maka proses eksekusinya semakin bertambah, khusunya pada Bubble sort dapat kita lihat pada Tabel 2. dan Tabel 3. Untuk melakukan sort pada data 1000 element Merge sort memiliki kompleksitas sebesar 3000 sedangkan Bubble sort 1000000. Ini dikarenakan Bubble sort memiliki kompleksitas $O(n^2)$ dan Merge sort yang memiliki kompleksitas O(n log n). Jadi dapat saya simpulkan bahwa Merge sort jauh lebih cepat dibandingkan dari Bubble sort.

## 5. Referensi

[1] Wikipedia contributors. (2022b, April 25). *Bubble sort*. Wikipedia. Retrieved April 26, 2022, from https://en.wikipedia.org/wiki/Bubble_sort

[2] Wikipedia contributors. (2022, April 19). *Merge sort*. Wikipedia. Retrieved April 26, 2022, from https://en.wikipedia.org/wiki/Merge_sort