PENERAPAN DAN PERBANDINGAN KOMPLEKSITAS WAKTU ALGORITMA *INSERTION SORT* DAN *QUICK SORT*

IKMAL NAWAWI THOHA
19102255
S1-IF-07-MM4

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2022

I. Dasar Teori

- *Algorithm Insertion Sort*

  Algorithm Insertion Sort didasarkan pada gagasan bahwa satu elemen dari elemen input dikonsumsi di setiap iterasi untuk menemukan posisinya yang benar, yaitu posisi yang dimilikinya dalam array yang diurutkan. Ini mengulangi elemen input dengan menumbuhkan array yang diurutkan pada setiap iterasi. [1]

  Ini membandingkan elemen saat ini dengan nilai terbesar dalam array yang diurutkan. Jika elemen saat ini lebih besar, maka ia meninggalkan elemen di tempatnya dan pindah ke elemen berikutnya jika tidak menemukan posisi yang benar dalam array yang diurutkan dan memindahkannya ke posisi itu. Ini dilakukan dengan menggeser semua elemen, yang lebih besar dari elemen saat ini, dalam array yang diurutkan ke satu posisi di depan. [2]

- *Algorithm Quick Sort*

  Algorithm quick sort adalah suatu algoritma pengurutan data yang menggunakan methode pemecahan data menjadi beberapa partisi, sehingga metode ini juga disebut dengan partition exchange sort. Untuk memulai literasi pengurutan, pertama-tama kamu harus memilih sebuah elemen dari data, yang kemudian elemen-elemen data tersebut akan diurutkan sedemikian rupa.

  Algoritma ini mengambil satu elemen secara acak (biasanya mengambil dari tengah) disebut dengan pivot yang kemudian disimpan pada semua elemen yang lebih kecil di sebelah kiri pivot dan semua yang lebih besar di sebelah kanan pivot. Methode ini dilakukan secara rekursif terhadap semua elemen di sebelah kiri dan kanan sampai semua elemen sudah terurut. [3]

II. Implementasi

- Spesifikasi *Hardware*
  - CPU : AMD Ryzen 5 5600U with Radeon Graphics 2.30 GHz
  - RAM : 16 GB

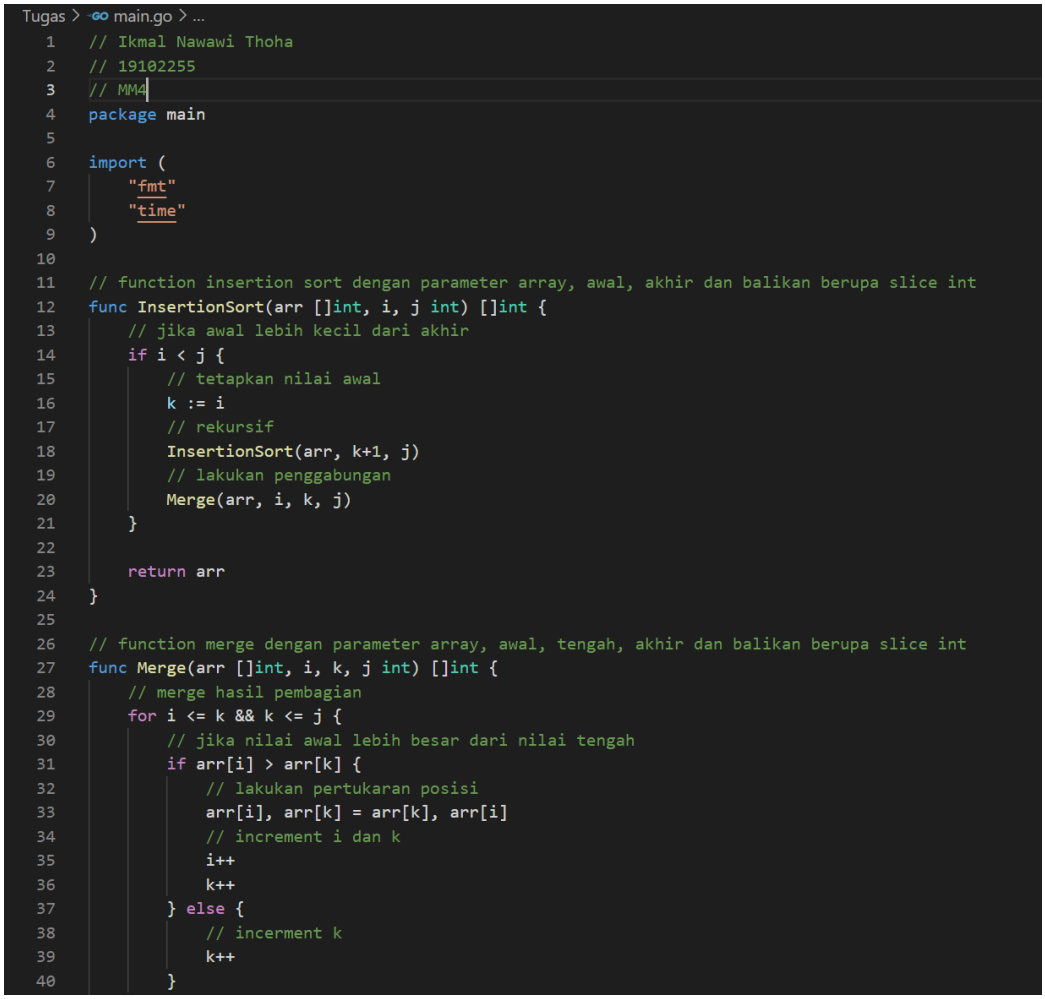
- Spesifikasi *Software*
  - *Operating System* : Windows 11
  - Bahasa Pemrograman : Golang v1.18.1
  - IDE : Visual Studio Code v1.66.2.0
  - Compiler : Python

- *Insertion Sort*
  - Pseudecode

```
prosedure InsertionSort (var arr: array of integer; i,j: integer);
{Mengurutkan element array arr dengan Insertion Sort
i dan j adalah indeks awal dan akhir array arr
Masukan : arr[i..j]
Luaran : arr[i..j] terurut
}

START
   // ketika i lebih kecil dari j
   if i < j then
      begin
         // inisiasi nilai awal
         k <- i
         // rekursif
         InsertionSort (arr, k+1, j)
         // merge
         Merge(arr, i, k, j)
      end

   return arr
END



prosedure Merge(var arr: array of integer; i,k,j: integer);
{Menggabngkan element array arr dengan Merge Sort
i, k, dan j adalah indeks awal, tengah, dan akhir array arr
Masukan : arr[i..k] dan arr[k+1..j]
Luaran : arr[i..j] terurut
}

START
   // looping element array arr
   for i <= k && k <= j do
      begin
         // jika element array arr[i] lebih besar dari element array arr[k]
         if arr[i] > arr[k] then
            begin
               // tukar element
               arr[i], arr[k] = arr[k], arr[i]
               // increment i
               i <- i + 1
```

```
            // incerment k
            k <- k + 1
        end
    else
        begin
            // increment k
            k <- k + 1
        end
  end
  return arr
END
```

- Screenshot Program *Insertion Sort*



```go
// Ikmal Nawawi Thoha
// 19102255
// MM4
package main

import (
    "fmt"
    "time"
)

// function insertion sort dengan parameter array, awal, akhir dan balikan berupa slice int
func InsertionSort(arr []int, i, j int) []int {
    // jika awal lebih kecil dari akhir
    if i < j {
        // tetapkan nilai awal
        k := i
        // rekursif
        InsertionSort(arr, k+1, j)
        // lakukan penggabungan
        Merge(arr, i, k, j)
    }

    return arr
}

// function merge dengan parameter array, awal, tengah, akhir dan balikan berupa slice int
func Merge(arr []int, i, k, j int) []int {
    // merge hasil pembagian
    for i <= k && k <= j {
        // jika nilai awal lebih besar dari nilai tengah
        if arr[i] > arr[k] {
            // lakukan pertukaran posisi
            arr[i], arr[k] = arr[k], arr[i]
            // increment i dan k
            i++
            k++
        } else {
            // incerment k
            k++
        }
```

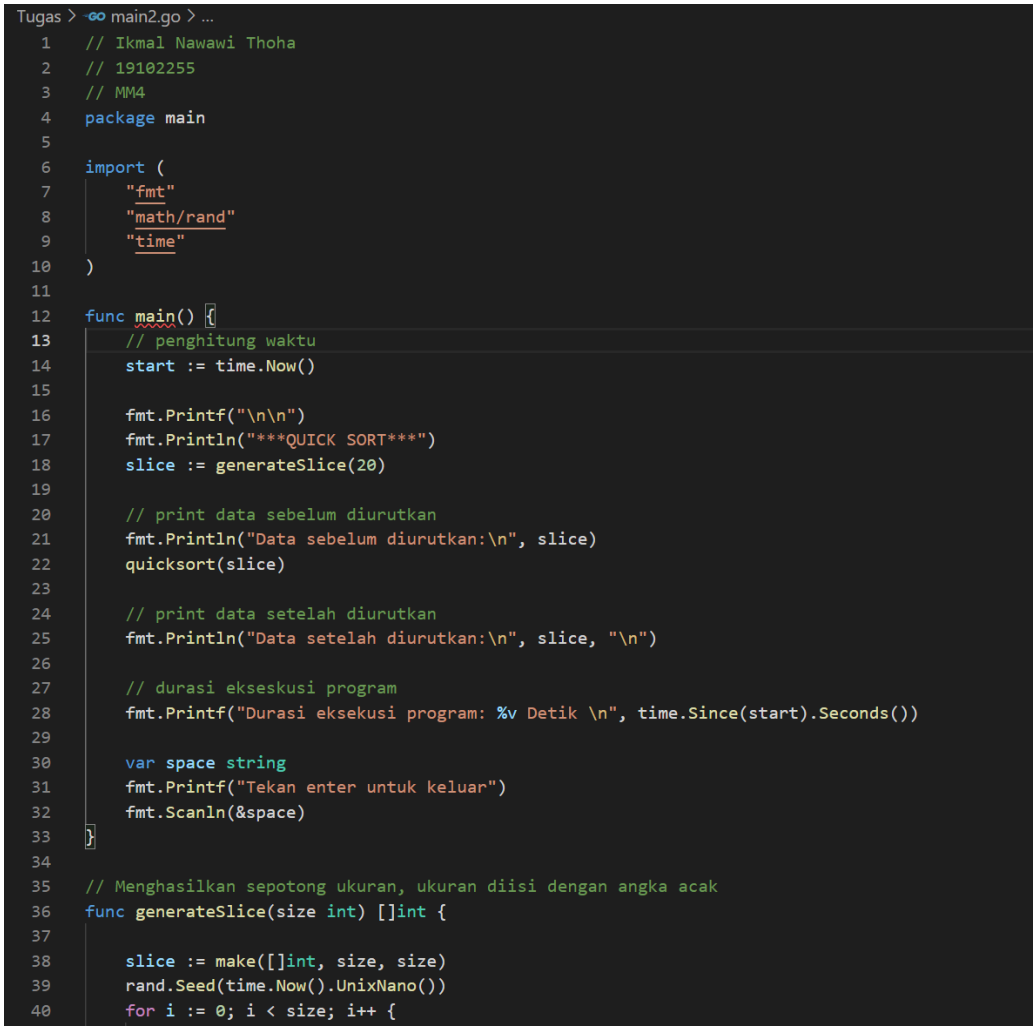
```
    }
    // kembalikan array
    return arr
}

func main() {
    // penghitung waktu
    start := time.Now()

    // my data
    data2 := []int{3, 5, 4, 6, 1, 2, 7, 9, 8}

    fmt.Printf("\n\n")
    fmt.Println("***INSERTION SORT***")
    fmt.Printf("\n\n")

    // print data sebelum diurutkan
    fmt.Printf("Data sebelum diurutkan: %v\n", data2)

    // urutkan data
    res := InsertionSort(data2, 0, len(data2)-1)

    // print data setelah diurutkan
    fmt.Printf("Data setelah diurutkan: %v\n\n\n\n", res)

    // durasi ekseskusi program
    fmt.Printf("Durasi eksekusi program: %v Detik \n", time.Since(start).Seconds())

    var space string
    fmt.Printf("Tekan enter untuk keluar")
    fmt.Scanln(&space)
}
```

- *Quick sort*
  - Pseudecode

```
Prosedure QuickSort (make([]int, size, size);
Menghasilkan sepotong ukuran, ukuran diisi dengan angka acak
func generateSlice(size int) []int {

START
  slice := make([]int, size, size)
  rand.Seed(time.Now().UnixNano())
  for i := 0; i < size; i++ {
        slice[i] = rand.Intn(999) - rand.Intn(999)
  }
  return slice
}

func quicksort(a []int) []int {
   if len(a) < 2 {
      return a
   }
END

Prosedure QuickSort (make([]int, size, size);
Menghasilkan sepotong ukuran, ukuran diisi dengan angka acak
func generateSlice(size int) []int {

STAR
   left, right := 0, len(a)-1

   pivot := rand.Int() % len(a)

   a[pivot], a[right] = a[right], a[pivot]

   for i, _ := range a {
      if a[i] < a[right] {
         a[left], a[i] = a[i], a[left]
         left++
      }
   }

   a[left], a[right] = a[right], a[left]

   quicksort(a[:left])
   quicksort(a[left+1:])
```

```
    return a
}
END
```

- Screenshot Program *Quick Sort*



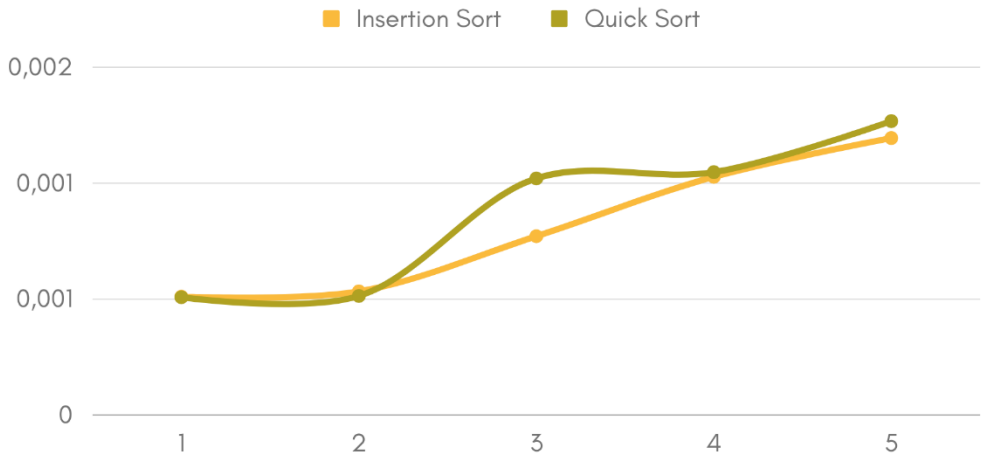
```
Tugas > main2.go > ...
1   // Ikmal Nawawi Thoha
2   // 19102255
3   // MM4
4   package main
5
6   import (
7       "fmt"
8       "math/rand"
9       "time"
10  )
11
12  func main() {
13      // penghitung waktu
14      start := time.Now()
15
16      fmt.Printf("\n\n")
17      fmt.Println("***QUICK SORT***")
18      slice := generateSlice(20)
19
20      // print data sebelum diurutkan
21      fmt.Println("Data sebelum diurutkan:\n", slice)
22      quicksort(slice)
23
24      // print data setelah diurutkan
25      fmt.Println("Data setelah diurutkan:\n", slice, "\n")
26
27      // durasi ekseskusi program
28      fmt.Printf("Durasi eksekusi program: %v Detik \n", time.Since(start).Seconds())
29
30      var space string
31      fmt.Printf("Tekan enter untuk keluar")
32      fmt.Scanln(&space)
33  }
34
35  // Menghasilkan sepotong ukuran, ukuran diisi dengan angka acak
36  func generateSlice(size int) []int {
37
38      slice := make([]int, size, size)
39      rand.Seed(time.Now().UnixNano())
40      for i := 0; i < size; i++ {
```

```
        slice[i] = rand.Intn(999) - rand.Intn(999)
    }
    return slice
}

func quicksort(a []int) []int {
    if len(a) < 2 {
        return a
    }

    left, right := 0, len(a)-1

    pivot := rand.Int() % len(a)

    a[pivot], a[right] = a[right], a[pivot]

    for i, _ := range a {
        if a[i] < a[right] {
            a[left], a[i] = a[i], a[left]
            left++
        }
    }

    a[left], a[right] = a[right], a[left]

    quicksort(a[:left])
    quicksort(a[left+1:])

    return a
}
```

III. Pengujian

Pada pengujian algoritma Insertion Sort dan Quick Sort, disini menggunakan data urut yang kemudian akan dimasukan ke dalam script. Data yang digunakan adalah random number. Pengujian menggunakan random number bertujuan untuk memudahkan dalam melakukan test waktu estimasi pada algoritma yang akan diuji. Kemudian melakukan 5 kali pengujian dengan maksimal limit 999 data dan range data yang digunakan. Berikut hasil pengujian yang telah dilakukan :

N	Waktu Estimasi Insertion Sort	Waktu Estimasi Quick Sort
1	0.0005108	0.0005083
2	0.000511	0.0010292
3	0.0010273	0.0010253
4	0.0005145	0.0005108
5	0.0005131	0.0010541



IV. Analisis Hasil Pengujian

- Algoritma Insertion Sort

  Berdasarkan pseudocode algoritma insertionsort di atas terdapat perulangan for dimana j adalah indeks 2 hingga Panjang array A. key adalah variabel yang menyimpan nilai j dari Array A. I adalah lokasi atau indeks yang berada di sebelah kiri indeks j. Setelah itu masuk ke perulangan While untuk membandingkan nilai kedua indeks i dan key serta memindahkan nilai hingga ke lokasi yang tepat. Apabila i lebih besar dari 0 dan A[i] lebih besar dari key, maka akan dilakukan perpindahan ruang dengan dilakukan penambahan + 1 pada indeks i. Ini menyatakan bahwa nilai indeks I berpindah ke kanan. Sementara itu kembali indeks iuntuk dilakukan komparasi (perulangan while) lagi.Jika i – 1 sama dengan 0 maka break, lalu key menempati ruang kosong yang berada di sebelah kiri.

- Algoritma Quick Sort

  Terdapat 3 jenis kompleksitas waktu dari quicksort :

  1. Kasus terburuk (worst case), yaitu terjadi bila terbentuk partisi dengan komposisi sub-masalah antara n – 1 elemen dan 0 elemen. Dengan demikian pemanggilan fungsi secara rekursif dengan array berukuran 0 akan langsung kembali, $T(0) = \Theta(1)$, sehingga berlaku: $T(n) = T(n – 1) + cn = O(n2)$.
  2. Kasus terbaik (best case), yaitu terjadi bila terbentuk partisi dengan dengan komposisi seimbang, dengan ukuran masing-masing tidak lebih dari n/2. Sehingga didapat: $T(n) = 2T(n/2) + cn = na + cn \log n = O(n \log n)$.
  3. Kasus rata-rata (average case), yaitu terjadi dari perimbangan poros antara terbaik dan terburuk, yang dalam prakteknya lebih mendekati kasus terbaik ketimbang terburuk. Sehingga didapat: $Tavg(n) = O(n \log n)$.

V. Kesimpulan

a. Kelemahan utama insertion sort adalah algoritma ini membutuhkan minimal passing/iterasi sebanyak n-1, sedangkan quick sort dapat kurang dari n -1. Pada kasus best case, algoritma exchange sort lebih unggul daripada insertion sort, sehubungan dengan kompleksitas yang lebih rendah yaitu nilai O(1) dibandingkan dengan O(n).4.Pada kasus worst case, algoritma insertion sort lebih unggul terhadap Quick sort, sehubungan dengan kompleksitas yang lebih rendah yaitu nilaiO (n), dibandingkan dengan O (n2).5. Algoritma insertion sort secara teknis lebih mudah diterapkan dibandingkan dengan Quick sort, berkaitan dengan panjangnya instruksi yang diperlukan.

b. Algoritma Quick Sort dapat kita ketahui sebagai algoritma yang handal dalam melakukan pengurutannya dari besarnya waktu asimptotik yang diperlukan apabila diberikan n buah masukan. Dimana kompleksitas algoritma dari algoritma ini memiliki 3 kasus yaitu O(n 2log n) untuk kasus terbaik, O(n2) untuk kasus terburuk, dan O(n 2log n) untuk kasus rata-rata. Kompleksitas tersebut dipengaruhi karena pemilihan pivot. Oleh karena itu proses pemilihan pivot perlu dipertimbangkan. Pivot yang terkadang digunakan yaitu elemen tengah dari tabel yang akan diurut.

VI. Referensi

[1] hackerearth, "hackerarth," [Online]. Available: https://www.hackerearth.com/practice/algorithms/sorting/selection-sort/tutorial/. [Accessed 18 April 2022].

[2] hackerearth, "hackerearth," [Online]. Available: https://www.hackerearth.com/practice/algorithms/sorting/insertion-sort/tutorial/. [Accessed 18 April 2022].

[3] "Membuat progam quick sort java – MAHIR CODING – M.HIKARU.N.Z." https://jagoprogramminx.wordpress.com/2020/01/12/membuat-progam-quick-sort-java/ (accessed Apr. 25, 2022).