**TUGAS 1 DESAIN DAN ANALISIS ALGORITME**

**Perbandingan Kompleksitas Algoritma Pengurutan**

**(Selection Sort dan Quick Sort)**

****

Disusun oleh :

Cahya Aprilia Putranti (19102080)

S1 IF 07 MM4

**PRODI S1 TEKNIK INFORMATIKA**

**FAKULTAS INFORMATIKA**

**INSTITUT TEKNOLOGI TELKOM PURWOKERTO**

**2022**

**Dasar teori**

Dalam beberapa konteks pemrograman, algoritma adalah spesifikasi dari urutan langkah-langkah untuk melakukan tugas tertentu. Untuk menyelesaikan suatu masalah, tidak cukup hanya dengan menemukan algoritma yang hasil pemecahan masalahnya benar. Ini berarti bahwa algoritma mengembalikan output yang diinginkan dari sejumlah input yang diberikan. Sebagus apapun algoritmanya, jika memberikan hasil yang salah, sudah pasti itu bukan algoritma yang bagus. Oleh karena itu, algoritma yang baik menggunakan algoritma yang efektif, efisien, terarah, dan terstruktur.Pilih algoritme yang kesesuaiannya dapat diukur dari segi waktu eksekusi algoritme dan jejak memori.

Selection sort adalah suatu metode pengurutan yang membandingkan elemen yang sekarang dengan elemen berikutnya sampai ke elemen yang terakhir. Jika ditemukan elemen lain yang lebih kecil dari elemen sekarang maka dicatat posisinya dan langsung ditukar. Dalam arti lain Selection Sort adalah pengurutan hard split/easy join dengan cara membagi larik menjadi dua buah upalarik yang tidak sama ukurannya

Metode selection sort adalah melakukan pemilihan dari suatu nilai yang terkecil dan kemudian menukarnya dengan elemen paling awal, lalu membandingkan dengan elemen yang sekarang dengan elemen berikutnya sampai dengan elemen terakhir, perbandingan dilakukan terus sampai tidak ada lagi pertukaran data

Quick Sort adalah sebuah algoritma sorting dari model Divide and Conquer yaitu dengan cara mereduksi tahap demi tahap sehingga menjadi 2 bagian yang lebih kecil. Algoritma pengurutan Quicksort dapat juga di sebut algoritma pengurutan yang terkenal dan tercepat (sesuai namanya).

Metode Quick Sort merupakan algoritma yang sangat cepat dibandingkan dengan algirtma sorting lainnya, karena algoritma quick sort ini melakukan sorting dengan membagi masalah menjadi sub masalah dan sub masalah dibagi lagi menjadi sub-sub masalah sehingga sorting tersebut menjadi lebih cepat walaupun memakan ruang memori yang besar.

**Implementasi**

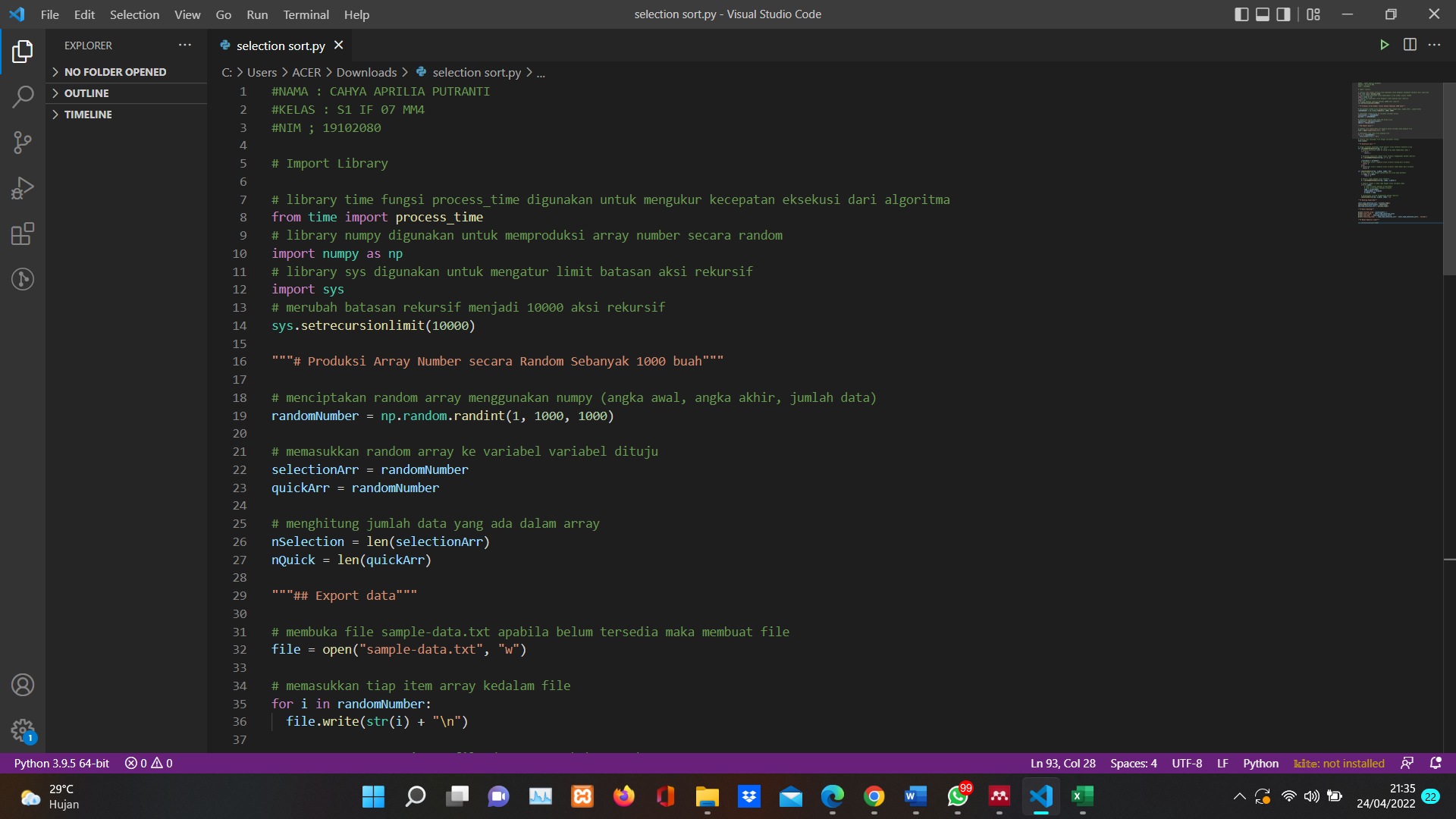
Cara kerja program :

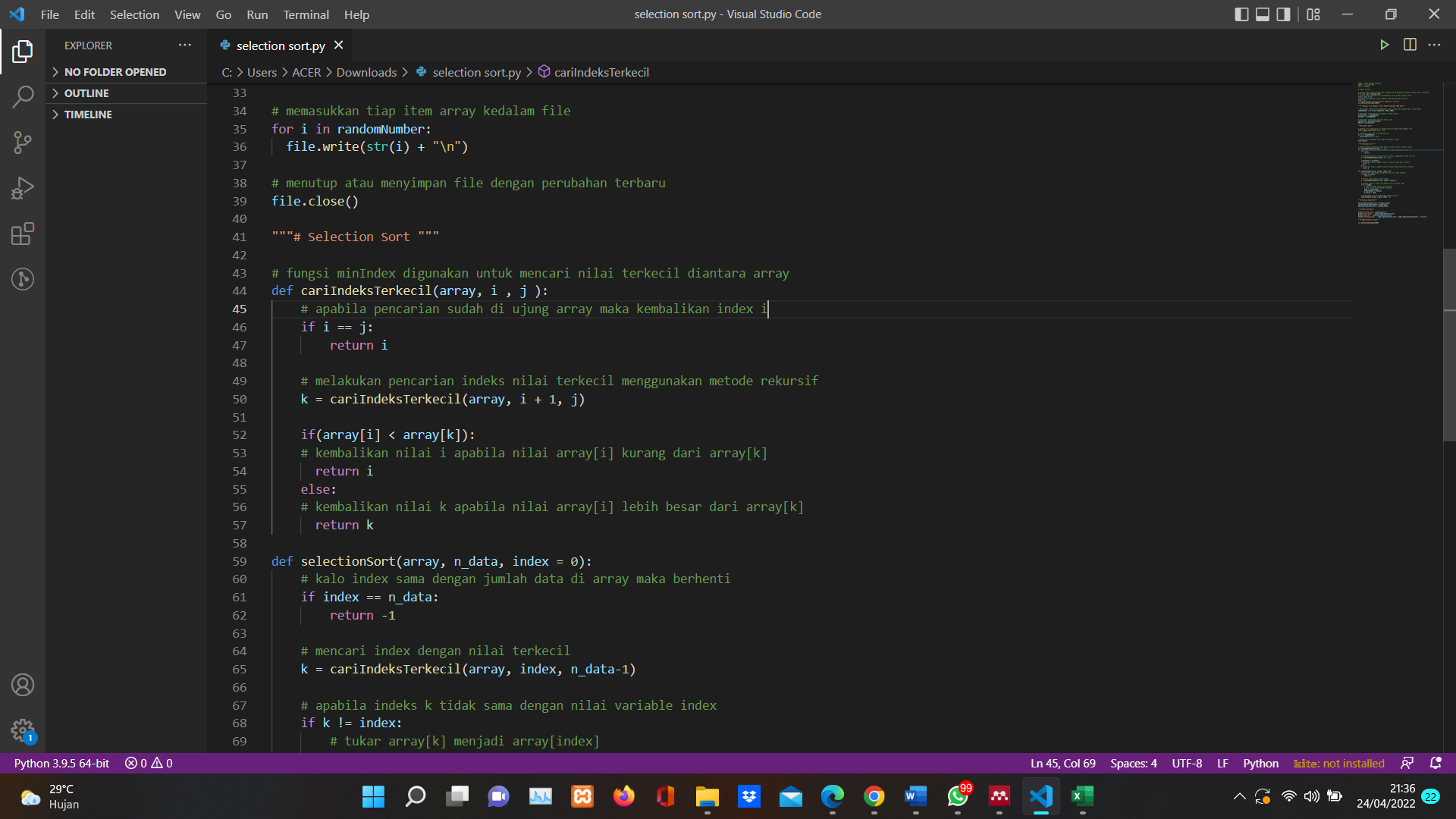
* Selection sort

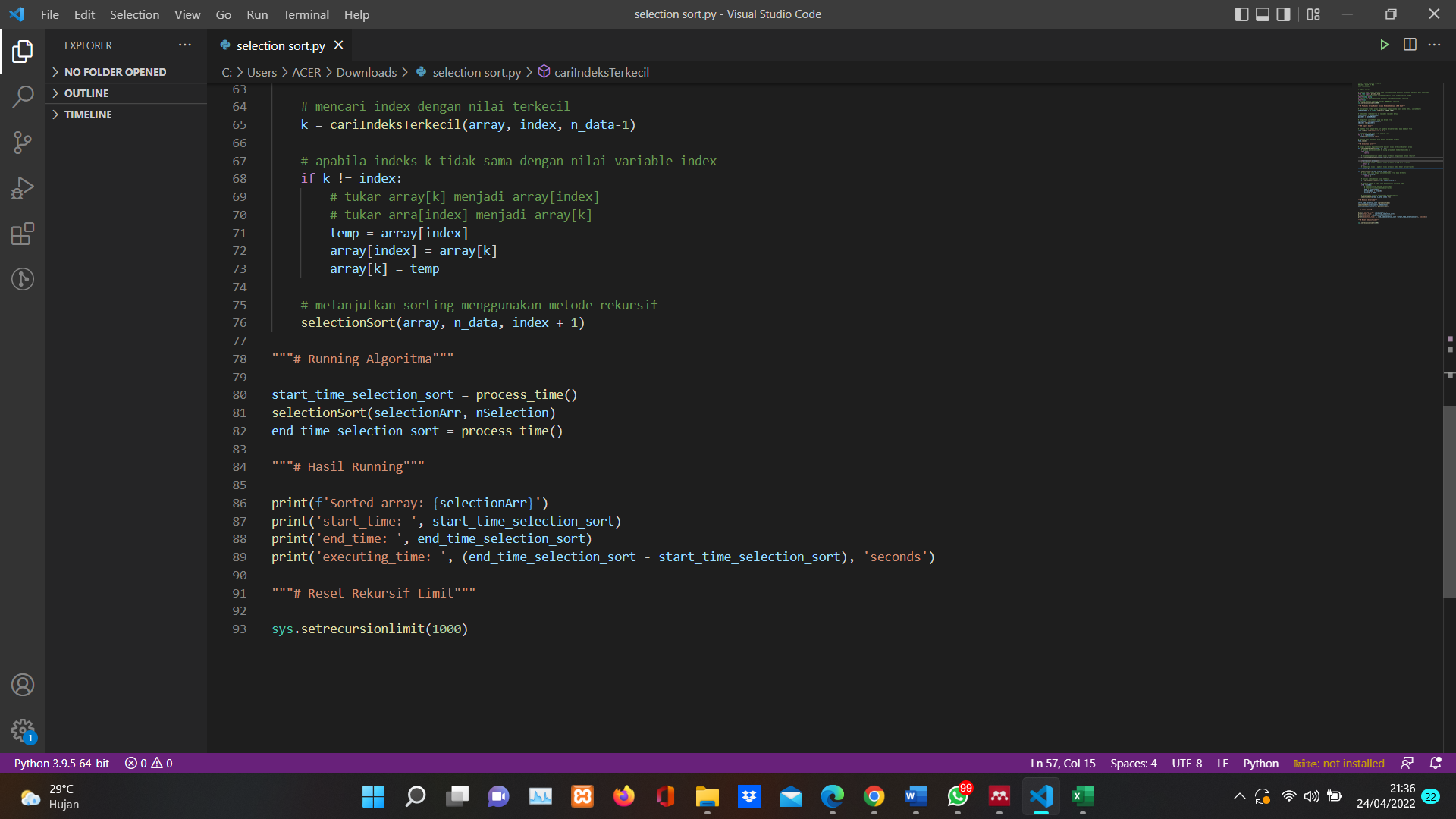
1. Pseudocode

|  |
| --- |
| Program Selection\_Sort  Deklarasi:  // pendeklarasian program menggunakan beberapa functions  // serta function yang bertipe data interger  randomNumber, selectionArr: Integer[]  nSelection, startTime, endTime: Integer  SubProgram carIndeksTerkecil  Parameter:  // variabel yang bertipe data interger  array: Integer[]  i, j: Integer  Deklarasi:  k: Integer  // algoritma yang digunakan menggunakan selection sort  Algoritma:  // disini merupakan algoritma yang digunakan pada program yang dibuat  if i == j then  output i  endif  k <- carIndeksTerkecil(array, i + i, j)  if array[i] < array[k] then  output i  else  output k  endif    SubProgram selectionSort  Parameter:  // pendeklrasian variable menggunakan tipe data integer  array: Integer[]  n\_data, index: Integer  Deklarasi:  k, temp: Integer  index <- 0  // disini merupakan pendeklarasian algoritam untuk program yang dibuat  Algoritma:  if index == n\_data then  output -1  endif  // mencari indeks diambil dari parameter diatas  k <- carIndeksTerkecil(array, index, n\_data)  if k != index then  temp <- array[index]  array[index] <- array[k]  array[k] <- temp  endif  selectionSort(array, n\_data, index + 1)    Algoritma:  randomNumber <- input(array)  selectionArr <- randomNumber  startTime <- input(time)  selectionSort(selectionArr)  endTime <- input(time)  output(selectionArr)  output(startTime)  output(endTime)  output(endTime - startTime) |

1. Screenshoot Program





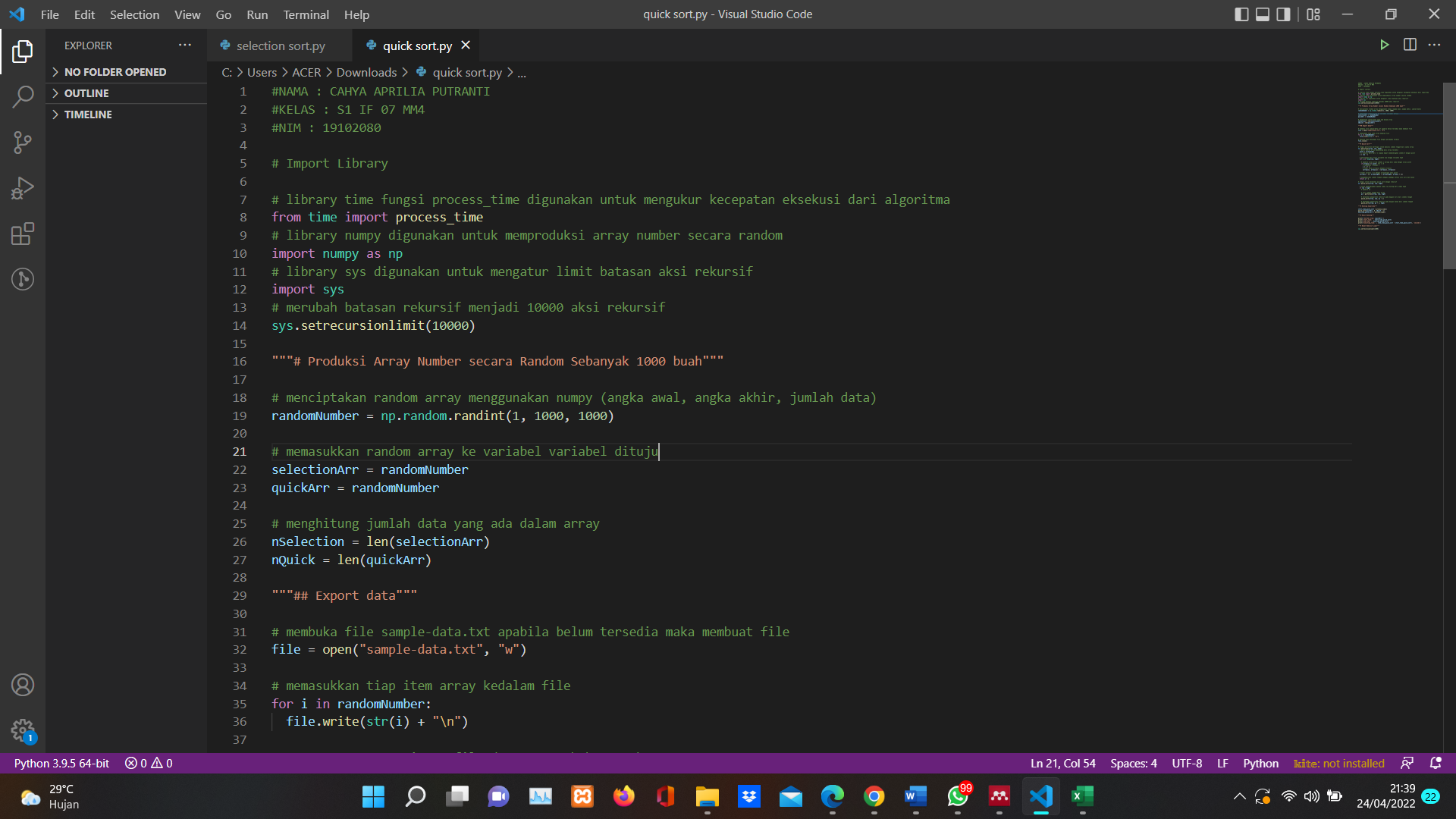


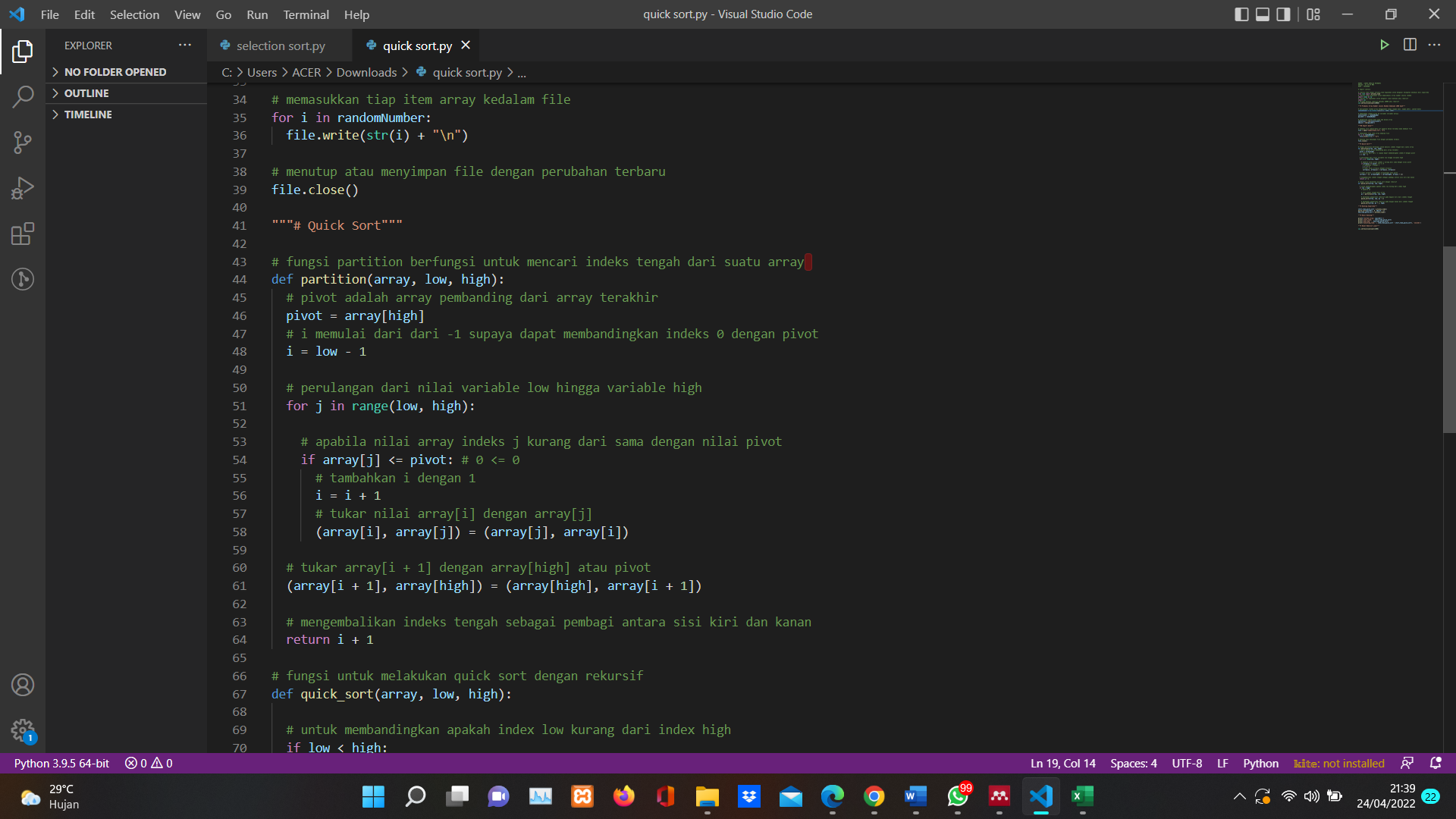
* Quick sort

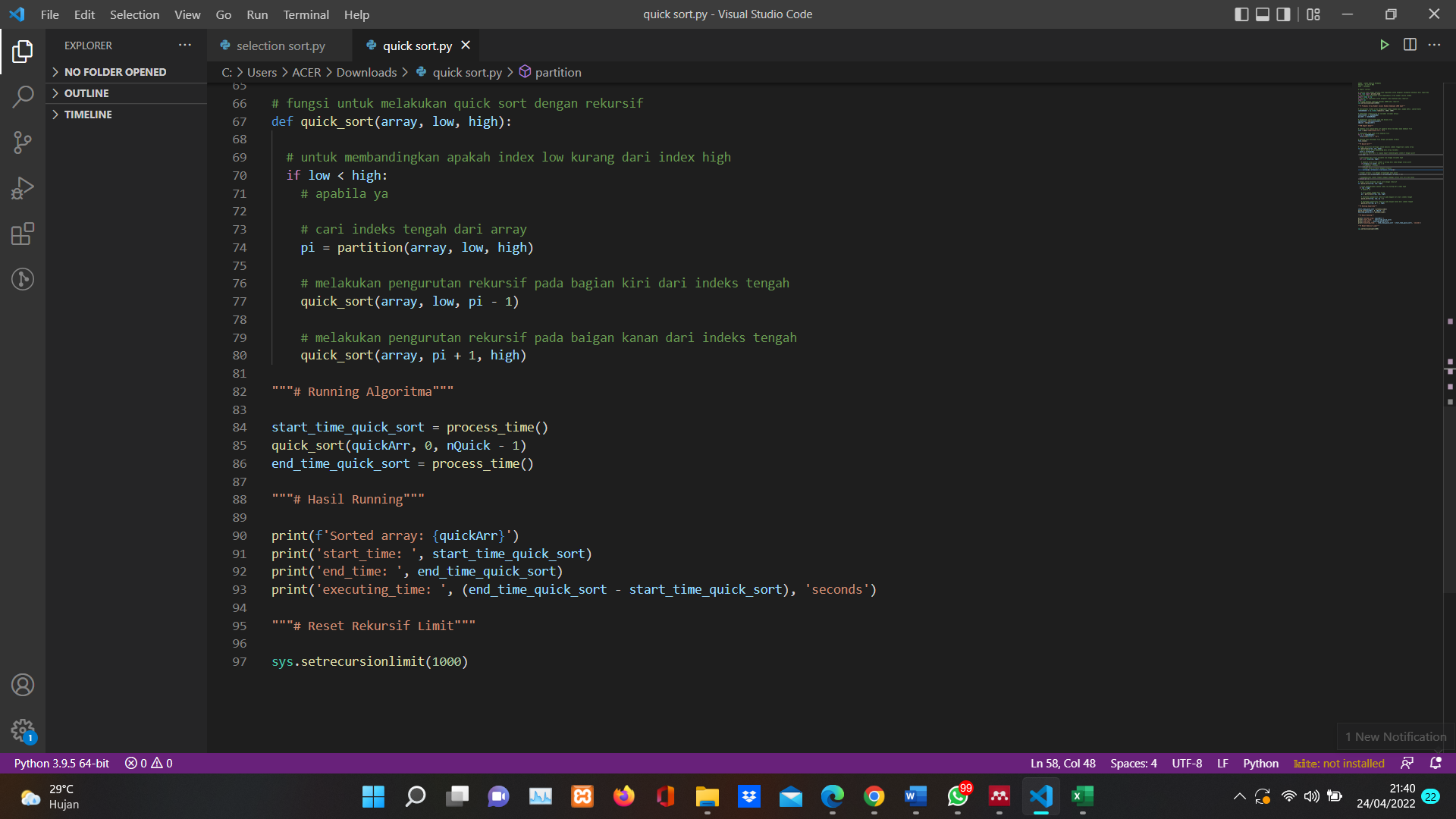
1. Pseudocode

|  |
| --- |
| Program quick\_Sort  Deklarasi:  // pendeklarasian program menggunakan beberapa functions  // serta semua function yang bertipe data interger  randomNumber, quickArr: Integer[]  nQuick, startTime, endTime: Integer  // variabel yang bertipe data interger  SubProgram partition  Parameter:  array: Integer[]  low, high: Integer  // pendeklrasian variable berupa I, pivot, temp, yang bertipe data integer  Deklarasi:  i, pivot, temp: Integer  // disini menggunakan algorita quict sort yang digunakan pada program ini  Algoritma:  pivot <- array[high]  i <- low - 1  // dan perulangan yang digunakan untuk program ini adalah menggunakan perulangan for  for j in low to high do  if array[j] <= pivot then  i <- i + 1  temp <- array[i]  array[i] <- array[j]  array[j] <- temp  endif  endfor  temp <- array[i + 1]  array[i + 1] <- array[high]  array[high] <- temp  output i + 1  SubProgram quicksort  // parameter variable semuanya bertipe integer  Parameter:  array: Integer[]  low, high: Integer  // pendeklrasian variable pi bertipe data integer  Deklarasi:  pi: Integer  // dan ini perupakan algoritma yang variablenya diambil dari pendeklrasian diatas yang menggunakan quick sort  Algoritma:  if low < high then  pi <- partition(array, low, high)  quick\_sort(array, low, pi - 1)  quick\_sort(array, pi + 1, high)  endif  Algoritma:  randomNumber <- input(array)  quickArr <- randomNumber  startTime <- input(time)  quickSort(quickArr)  endTime <- input(time)  output(quickArr)  output(startTime)  output(endTime)  output(endTime - startTime) |

1. Screenshoot Program







**Pengujian**

Dalam pengujian dini, pengungi menggunakan algoritma selection sort dan quick sort sebegai perbandingannya. Data yang digunakan untuk pengujian yaitu data dummy atau random data yang kemudian dimasukan dalam array. Untuk pengujian yang pertama menggunakan inputan data 100 kemudian di running dan di dapatkan waktu hasil eksekusi sebesar 0.003 detik pada selection sort dan 0.005 detik pada quick sort, kemudian pada pengujian kedua menggunakan inputan data 500 dan di dapatkan waktu hasil eksekusi sebesar 0,5 detik pada selection sort dan 0,6 detik pada quick sort, dan yang terakhir yaitu dengan menginputkan data sebesar 1000 kemudian di running dan di dapatkan hasil eksekusi sebesar 0.2 detik pada selection sort dan 0.26 detik pada quick sort.

Untuk spesifikasi hardware saya menggunakan device laptop acer aspire dengan processor Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz 1.19 GHz, RAM (4,00 GB), dan system type (64-bit operating system, x64-based processor) dan untuk software yang saya gunakan yaitu windows 11, menggunakan Bahasa pemrograman python, compiler python dan IDE google collab.

|  |  |  |
| --- | --- | --- |
| n | Waktu eksekusi algoritma Selection | Waktu eksekusi algoritma Quick |
| 100 | 0.0037 seconds | 0.0027 seconds |
| 500 | 0.0807 seconds | 0.0741 seconds |
| 1000 | 0.2046 seconds | 0.2698 seconds |

Tabel eksekusi waktu masing masing algoritma

Keterangan:

n = jumlah inputan

waktu eksekusi dalan detik

* Algoritma selection sort

Grafik selection sort

* Algoritma quick sort

Grafik quick sort

**Analisis Hasil Pengujian**

* Selection sort

Algoritma ini mempunyai dua for loop, satu loop di dalam loop yang lainnya. Banyaknya perbandingan yang harus dilakukan untuk siklus pertama adalah n, perbandingan yang harus dilakukan untuk siklus yang kedua n-1, dan seterusnya. Sehingga jumlah keseluruhan perbandingan adalah n(n+1)/2-1 perbandingan.

Kompleksitas Waktu nya O(n2) karena ada dua loop bersarang. Hal yang baik tentang sortir seleksi adalah tidak pernah membuat lebih dari O(n) swap dan dapat berguna ketika penulisan memori adalah operasi yang mahal.

* Quick sort

Kompleksitas efisiensi quicksort dari algoritma quicksort sangat di pengaruhi oleh pilihan elemen pivot. Pemilihan pivot menentukan jumlah dan ukuran partisi di setiap fase rekrusif. Waktu yang dibutuhkan oleh QuickSort secara umum dapat dituliskan sebagai berikut.

T(n) = T(k) + T(n-k-1) + (n)

Dua istilah pertama untuk dua panggilan rekursif, istilah terakhir untuk proses partisi. k adalah jumlah elemen yang lebih kecil dari pivot. Waktu yang dibutuhkan oleh QuickSort tergantung pada array input dan strategi partisi.

1. Kasus terburuk:

Kasus terburuk terjadi ketika proses partisi selalu memilih elemen terbesar atau terkecil sebagai pivot. Jika kita mempertimbangkan strategi partisi di atas di mana elemen terakhir selalu dipilih sebagai pivot, kasus terburuk akan terjadi ketika array sudah diurutkan dalam urutan naik atau turun. Berikut ini adalah pengulangan untuk kasus terburuk.

T(n) = T(0) + T(n-1) + (n)

yang ekuivalen dengan

T(n) = T(n-1) + (n)

Solusi dari pengulangan di atas adalah **(n2).**

1. Kasus terbaik:

Kasus terbaik terjadi ketika proses partisi selalu memilih elemen tengah sebagai pivot. Berikut ini adalah pengulangan untuk kasus terbaik.

T(n) = 2T(n/2) + O(n)

Solusi dari pengulangan di atas adalah **O(nlogn).**

1. Kasus Rata-rata:

Untuk melakukan analisis kasus rata-rata, kita perlu mempertimbangkan semua kemungkinan permutasi array dan menghitung waktu yang dibutuhkan oleh setiap permutasi yang tidak terlihat mudah. Kita dapat memperoleh gambaran kasus rata-rata dengan mempertimbangkan kasus ketika partisi menempatkan elemen (n/9) dalam satu himpunan dan elemen (9n/10) dalam himpunan lain. Berikut rekurensi untuk kasus ini.

T(n) = T(n/9) + T(9n/10) + (n)

Solusi dari pengulangan di atas juga **(nlogn)**

Perbandingan hasil perhitungan kompleksitas algoritma selection dan quick sort yaitu pada algoritma selection sort (O(n2 )membutuhkan waktu lebih lama untuk mengurutkan data sedangkan quick sort tidak membutuhkan waktu yang lama untuk mengurutkan datanya, hal ini sesuai dengan kompleksitas algoritmanya. Dan dapat di simpulkan Algoritma Quick Sort lebih cepat dalam melakukan pengurutan data jika dibandingkan dengan Selection Sort.

**Referensi**

[1] Rahayuningsih, Panny Agustia. "Analisis Perbandingan Kompleksitas Algoritma Pengurutan Nilai (Sorting)." *EVOLUSI: Jurnal Sains dan Manajemen* 4.2 (2016).

[2] Al Rivan, Muhammad Ezar. "Perbandingan Kecepatan Gabungan Algoritma Quick Sort dan Merge Sort dengan Insertion Sort, Bubble Sort dan Selection Sort." *Jurnal Teknik Informatika dan Sistem Informasi* 3.2 (2017).