LAPORAN TUGAS KECIL II STRATEGI ALGORITMA

MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA *DIVIDE AND CONQUER*

Logo

Description automatically generated

DISUSUN OLEH

NAMA: HAZIQ ABIYYU MAHDY

NIM: 13521170

KELAS: K02

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

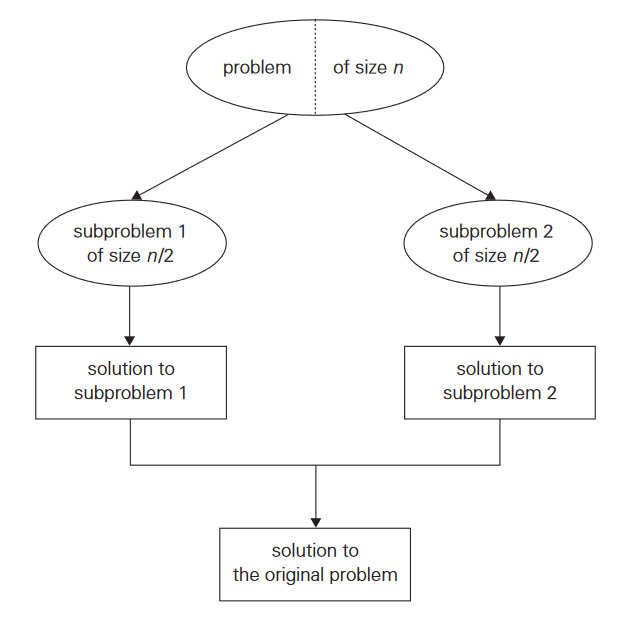
INSTITUT TEKNOLOGI BANDUNG

**BAB I**

**DESKRIPSI PERSOALAN**

Algoritma *divide and conquer* adalah jenis algoritma yang menyelesaikan suatu permasalahan dengan membagi permasalahan tersebut menjadi upa-permasalahan yang lebih kecil. Secara umum, algoritma *divide and conquer* terdiri atas tiga langkah.

1. *Divide*, yaitu membagi permasalahan menjadi beberapa upa-permasalahan yang memiliki karakteristik yang sama dengan permasalahan semula. Idealnya, masing-masing upa-permasalahan memiliki ukuran yang sama.
2. *Conquer*, yaitu menyelesaikan upa-permasalahan, biasanya secara rekursif.
3. *Combine*, yaitu menggabungkan solusi dari upa-permasalahan untuk mendapatkan solusi permasalahan semula jika diperlukan.



**Gambar 1.1** Ilustrasi algoritma *divide and conquer*

Pada tugas kecil ini, penulis mengembangkan algoritma *divide and conquer* dalam bahasa Python untuk mencari pasangan titik terdekat pada ruang 3D. Pendekatan *divide and conquer* tersebut kemudian dibandingkan dengan pendekatan *brute force* dari segi waktu eksekusi dan jumlah operasi yang dilakukan. Misalkan terdapat n buah titik pada ruang 3D. Setiap titik P di dalam ruang dinyatakan dengan koordinat P = (x, y, z). Jarak dua buah titk P1 = (x1, y1, z1) dan P2 = (x2, y2, z2) dihitung dengan rumus Euclidean berikut.

Pada tugas ini juga diberikan bonus berupa *plotting* titik-titik pada grafik (bonus 1) serta generalisasi program untuk dapat menemukan pasangan titik terdekat dalam R­­­N  (bonus 2).

Masukan program:

1. Banyaknya titik (n)
2. Dimensi (untuk bonus 1)
3. Titik-titik (dibangkitkan secara acak)

Luaran program:

1. Pasangan titik yang jaraknya terdekat dan nilai jaraknya
2. Banyaknya operasi perhitungan rumus Euclidean
3. Waktu riil dalam detik serta spesifikasi komputer yang digunakan
4. Penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya (untuk bonus 2).

Pada program ini, digunakan beberapa *library* sebagai berikut.

1. math, untuk melakukan operasi sqrt()
2. numpy, untuk membangkitkan bilangan acak (pada fungsi random.uniform()) serta untuk struktur data NDArray
3. matplotlib, untuk melakukan *plotting* titik-titik pada grafik
4. time, untuk menghitung waktu eksekusi program.

**BAB II**

**ALGORITMA PENYELESAIAN MASALAH**

Diberikan array P[0..n-1] yang berisi titik-titik [p0, p1, .., pn-1] dengan pi = (xi, yi,, zi). Jika pencarian pasangan titik terdekat dilakukan dengan algoritma brute force, berikut adalah langkah-langkah yang dilakukan.

1. Simpan nilai jarak minimum sebesar tak hingga. Nilai ini kemudian akan diganti dengan jarak titik terdekat yang ada di P.
2. Lakukan iterasi terhadap seluruh titik-titik yang ada di P. Untuk setiap titik pi, lakukan iterasi dari i+1 sampai n-1 dan lakukan perhitungan jarak Euclidean dengan seluruh titik [pi+1 .. pn-1]. Jika terdapat pasangan titik yang jaraknya lebih kecil dibandingkan jarak minimum sebelumnya, maka pasangan titik tersebut di-*assign* sebagai pasangan terdekat dan jaraknya di-*assign* sebagai jarak minimum.
3. Setelah iterasi selesai, kembalikan pasangan titik terdekat dan jarak minimum.

Banyaknya operasi yang diperlukan untuk algoritma *brute force* adalah sebagai berikut.

Untuk menyelesaikan permasalahan yang sama dengan pendekatan *divide and conquer*, maka dilakukan langkah-langkah berikut.

1. Titik-titik pada P diurutkan menaik berdasarkan nilai absis menggunakan algoritma *quicksort*
2. Kasus basis: apabila titik-titik yang ada pada P berjumlah dua atau tiga, maka lakukan pencarian pasangan titik terdekat dengan cara *brute force*.
3. Kasus rekurens: apabila titik-titik yang ada pada P berjumlah lebih dari tiga, ambil titik tengah yang dapat membagi *array* P menjadi dua bagian, yaitu partisi kiri dan pariti kanan. Titik tengah tersebut adalah pn div 2. Titik-titik [p0 .. pn div 2 -1] berada di sebelah kiri xn div 2 dan titik-titik [pn div 2 .. pn] berada di sebelah kanan xn div 2. Pada gambar dibawah, terdapat titik-titik [p0 .. p5], dengan p3 sebagai pembagi partisi. Bidang x = 2, yang merupakan absis dari titik p­3 membagi *array* menjadi dua partisi, dengan [p0, p1, p2] sebagai partisi kiri dan [p3, p4, p5] sebagai partisi kanan.

Chart

Description automatically generated

**Gambar 2.1** Ilustrasi pembagian titik-titik menjadi dua partisi

1. Lakukan pemanggilan fungsi *divide and conquer* untuk partisi kiri dan partisi kanan, kemudian cari jarak pasangan titik terdekat antara dua partisi tersebut, yaitu jarak terkecil di antara jarak terkecil pada partisi kiri dan jarak terkecil pada partisi kanan. Simpan jarak tersebut sebagai d dan simpan pula pasangan terdekat yang didapatkan.
2. Karena terdapat kemungkinan titik terdekat dipisahkan oleh bidang tengah, maka kita harus melakukan pengecekan terhadap titik-titik yang jarak absisnya dengan bidang tengah kurang dari atau sama dengan d. Daerah tersebut disebut *slab*.

Chart

Description automatically generated

**Gambar 2.2** Ilustrasi *slab*

1. Kita dapat melakukan pengecekan terhadap jarak pasangan titik-titik yang ada pada *slab*. Apabila jarak suatu pasangan titik lebih kecil dari jarak minimum yang disimpan sebelumnya, maka kita akan memperbarui nilai jarak minimum dan pasangan terdekat. Namun, untuk sebuah titik Q (Qx, Qy, Qz) di dalam *slab*, kita tidak perlu mencari jarak titik Q dengan seluruh titik lainnya pada *slab*. Terdapat paling banyak 18 titik yang perlu dicari jaraknya dengan titik Q, yaitu titik R (Rx, Ry, Rz) dengan | Qy - Ry | ≤ d dan | Qz - Rz | ≤ d. Apabila | Qy - Ry | > d atau | Qz - Rz | > d, maka jarak titik Q dan R > d, sehingga pasangan titik tersebut sudah pasti bukan merupakan pasangan terdekat. Perlu diingat bahwa 18 hanyalah batas atas jumlah titik yang perlu dihitung jaraknya dengan Q. Pada umumnya, jumlah titik yang perlu ditinjau kurang dari 18. Misalkan Q sebuah titik pada bagian kiri *slab*, banyak titik maksimum yang perlu ditinjau dapat dilihat pada gambar berikut.

Chart, line chart

Description automatically generated

**Gambar 2.3** Banyaknya titik maksimum yang perlu ditinjau dari Q

1. Setelah melakukan pencarian pada *slab*, kita akan mendapatkan pasangan titik terdekat serta jarak pasangan titik tersebut.

Pada kasus basis, banyaknya operasi perhitungan jarak Euclidean yang diperlukan pada pemanggilan fungsi *divide and conquer* adalah konstan (satu perhitungan untuk n = 2 atau tiga perhitungan untuk n = 3). Sedangkan pada kasus rekurens, banyaknya operasi yang diperlukan pada pemanggilan fungsi ini adalah dua kali banyaknya operasi yang diperlukan untuk memanggil fungsi yang sama untuk ukuran n/2 (pencarian pada partisi kiri dan partisi kanan) ditambah banyaknya operasi yang diperlukan untuk pencarian pada *slab*. Untuk setiap titik pada *slab*, batas atas banyaknya perhitungan jarak Euclidean yang diperlukan adalah 18 (suatu konstanta), maka dapat dinyatakan bahwa banyaknya operasi yang diperlukan untuk pencarian pada *slab* adalah kn, dengan k adalah suatu konstanta. Sehingga, banyaknya operasi yang diperlukan dapat dinyatakan sebagai fungsi berikut.

Kompleksitas dalam notasi *Big-Oh* dapat dihitung dengan menggunakan teorema Master.

Text, letter

Description automatically generated

Diketahui a = 2, b = 2, dan d = 1. Karena a = bd, maka kompleksitas algoritma ini adalah

Dapat dibuktikan bahwa pada permasalahan ini, algoritma *divide and conquer* secara teoretis lebih sangkil dibandingkan algoritma *brute force*. Untuk membuktikannya lebih lanjut, penulis melakukan *plotting* jumlah operasi perhitungan jarak Euclidean yang diperlukan pada kedua algoritma. Pada gambar di bawah, sumbu-x melambangkan banyaknya masukan dan sumbu-y melambangkan banyaknya operasi yang diperlukan. Dapat dilihat bahwa pertumbuhan jumlah operasi pada algoritma *divide and conquer* (dilambangkan dengan titik merah) jauh lebih lambat dibandingkan dengan pertumbuhan jumlah operasi pada algoritma *brute force* (dilambangkan dengan titik biru).

**Chart, scatter chart

Description automatically generated**

**Gambar 2.4** Perbandingan algoritma *divide and conquer* dengan *brute force*

**BAB III**

**SOURCE CODE**

1. closest\_pair.py

|  |
| --- |
| import math  import numpy as np  import matplotlib.pyplot as plt    def generate\_random\_points(N, dimensions, threshold=1000):      """      Mengembalikan numpy.NDArray berisi N buah titik acak dengan dimensi sebesar dimensions.      Seluruh titik akan berada pada range      -1 \* threshold <= x < threshold,      -1 \* threshold <= y < threshold,      -1 \* threshold <= z < threshold,      dst.      """      arr\_points = np.empty(N, dtype=object)        for i in range(N):          arr\_points[i] = []          for \_ in range(dimensions):              value = round(np.random.uniform(-1 \* threshold, threshold), 2)              arr\_points[i].append(value)          arr\_points[i] = tuple(arr\_points[i])      return arr\_points    def euclidean\_dist(point\_a, point\_b, dimensions):      """      Mengembalikan jarak euclidean antara titik a dan titik b dengan dimensi sebesar dimensions      """      result = 0      for i in range(dimensions):          result += (point\_a[i] - point\_b[i])\*\*2      result = math.sqrt(result)      return result    def bf\_closest\_pair(arr\_points, dimensions):      """      Mengembalikan jarak terpendek, pasangan titik terdekat, serta banyaknya operasi      perhitungan jarak euclidean yang dilakukan dengan algoritma brute force      Prekondisi:      len(arr\_points) >= 2 dan dimensions >=1      """      closest\_pair = None      min\_dist = float('inf')      operations = 0      for i in range(0, len(arr\_points)):          for j in range(i+1, len(arr\_points)):              current\_dist = euclidean\_dist(arr\_points[i], arr\_points[j], dimensions)              operations += 1              if (current\_dist < min\_dist):                  min\_dist = current\_dist                  closest\_pair = (arr\_points[i], arr\_points[j])      return min\_dist, closest\_pair, operations    def dnc\_closest\_pair(arr\_points, dimensions):      """      Mengembalikan jarak terpendek, pasangan titik terdekat, serta banyaknya operasi      perhitungan jarak euclidean yang dilakukan dengan algoritma divide and conquer.      Prekondisi:      len(arr\_points) >= 2, dimensions >=1, dan arr\_points      sudah terurut berdasarkan absis      """      num\_points = len(arr\_points)      if (num\_points <= 3):          return bf\_closest\_pair(arr\_points, dimensions)      # Pengambilan titik tengah yang dapat membagi array menjadi partisi kiri dan kanan sama rata      # dengan elemen tengah array sebagai acuan.      mid = num\_points // 2      mid\_x = arr\_points[mid][0]      left\_arr\_points = arr\_points[:mid]      right\_arr\_points = arr\_points[mid:]      # Pencarian pasangan terdekat pada partisi kiri dan kanan secara rekursif      left\_min\_dist, left\_closest\_pair, left\_operations = dnc\_closest\_pair(left\_arr\_points, dimensions)      right\_min\_dist, right\_closest\_pair, right\_operations = dnc\_closest\_pair(right\_arr\_points, dimensions)      # Jarak terpendek sementara adalah min(left\_min\_dist, right\_min\_dist)      if (left\_min\_dist <= right\_min\_dist):          closest\_pair = left\_closest\_pair          min\_dist = left\_min\_dist      else:          closest\_pair = right\_closest\_pair          min\_dist = right\_min\_dist      # Titik-titik yang jarak absisnya dengan mid\_x <= min\_dist akan dimasukkan ke dalam slab      slab = []      slab\_width = min\_dist      slab\_operations = 0      for point in arr\_points:          if (abs(point[0] - mid\_x) <= slab\_width):              slab.append(point)      # Pencarian pasangan terdekat untuk titik-titik yang berada dalam slab      for i in range(0, len(slab)):          for j in range(i + 1, len(slab)):              # Jika terdapat sepasang titik di dalam slab yang tidak memenuhi kriteria close\_enough              # (lihat fungsi di bawah) maka tidak perlu dilakukan perhitungan              # jarak euclidean, karena sudah pasti jaraknya > min\_dist              if (close\_enough(slab[i], slab[j], slab\_width, dimensions)):                  current\_dist = euclidean\_dist(slab[i], slab[j], dimensions)                  slab\_operations += 1                  if (current\_dist < min\_dist):                      min\_dist = current\_dist                      closest\_pair = (slab[i], slab[j])        total\_operations = left\_operations + right\_operations + slab\_operations      return min\_dist, closest\_pair, total\_operations    def close\_enough(point\_a, point\_b, dist\_limit, dimensions):      """      Menentukan apakah titik a (x1, y1, z1) dan titik b (x2, y2, z2)      cukup dekat dengan kriteria berikut:      |x1 - x2| < dist\_limit,      |y1 - y2| < dist\_limit,      |z1 - z2| < dist\_limit,      dst.      Digunakan untuk mengurangi jumlah perhitungan pada algoritma divide and conquer      """      for i in range(1, dimensions):          if (abs(point\_a[i] - point\_b[i]) > dist\_limit):              return False        return True    def plot\_results(arr\_points, closest\_pair):      """      Melakukan plotting terhadap titik-titik yang ada pada arr\_points. pasangan titik      yang merupakan pasangan terdekat diberi warna merah, sedangkan yang lainnya      diberi warna hijau      """      fig = plt.figure(figsize = (16, 9))      ax = plt.axes(projection ="3d")      ax.set\_xlabel('X')      ax.set\_ylabel('Y')      ax.set\_zlabel('Z')        for point in arr\_points:          if point in closest\_pair:              ax.scatter3D(point[0], point[1], point[2], color = "red")          else:              ax.scatter3D(point[0], point[1], point[2], color = "green")        plt.show()    def quick\_sort\_by\_x(arr\_points, left, right):      """      Melakukan sorting pada arr\_points dengan algoritma quick sort      """      if (left < right):          pivot = arr\_points[right][0]          i = left - 1          for j in range(left, right):              if (arr\_points[j][0] <= pivot):                  i += 1                  arr\_points[i], arr\_points[j] = arr\_points[j], arr\_points[i]          pivotIdx = i + 1          arr\_points[pivotIdx], arr\_points[right] = arr\_points[right], arr\_points[pivotIdx]          quick\_sort\_by\_x(arr\_points, left, pivotIdx-1)          quick\_sort\_by\_x(arr\_points, pivotIdx + 1, right) |

1. main.py

|  |
| --- |
| import time  import closest\_pair as cp  print("====================== INPUTS =======================")  while (True):      num\_points = int(input("Number of points: "))      if (num\_points >=2):          break      print("Number of points must be greater than or equal to 2")  while (True):      dim = int(input("Dimensions: "))      if (dim >= 1):          break      print("Dimensions must be greater than or equal to 1")  print("=====================================================")  print("\n")  arr\_points = cp.generate\_random\_points(num\_points, dim, threshold=1000)  bf\_start\_time = time.time()  bf\_min\_dist, bf\_closest\_pair, bf\_operations = cp.bf\_closest\_pair(arr\_points, dim)  bf\_finish\_time = time.time()  print("==================== BRUTE FORCE ====================")  print("Min distance:", bf\_min\_dist)  print("Closest pair:", bf\_closest\_pair)  print("Execution time:", bf\_finish\_time - bf\_start\_time)  print("Total operations:", bf\_operations)  print("=====================================================")  print("\n")    dnc\_start\_time = time.time()  cp.quick\_sort\_by\_x(arr\_points, 0, len(arr\_points) - 1)  dnc\_min\_dist, dnc\_closest\_pair, dnc\_operations = cp.dnc\_closest\_pair(arr\_points, dim)  dnc\_finish\_time = time.time()  print("================ DIVIDE AND CONQUER =================")  print("Min distance:", dnc\_min\_dist)  print("Closest pair: ", dnc\_closest\_pair)  print("Execution time:", dnc\_finish\_time - dnc\_start\_time)  print("Total operations:", dnc\_operations)  print("=====================================================")  if (dim == 3):      ans = input("Plot results? (y/n): ")      if (ans == "y"):          cp.plot\_results(arr\_points, dnc\_closest\_pair) |

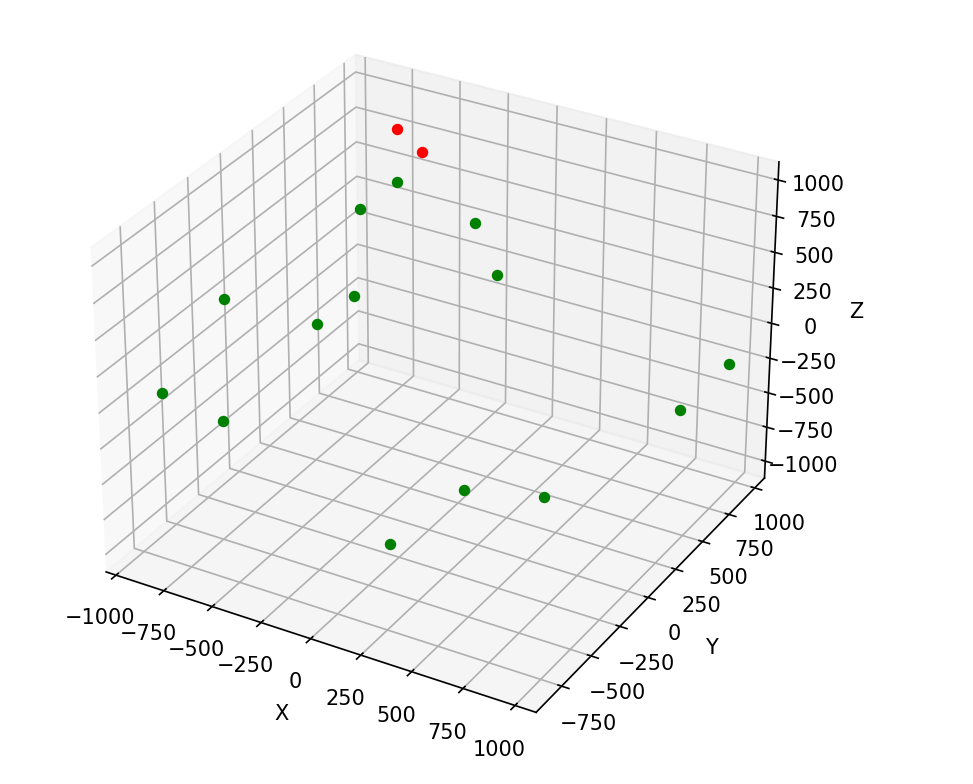
**BAB IV**

**KASUS UJI DAN CHECKLIST**

1. n = 16

Text

Description automatically generated



1. n = 64

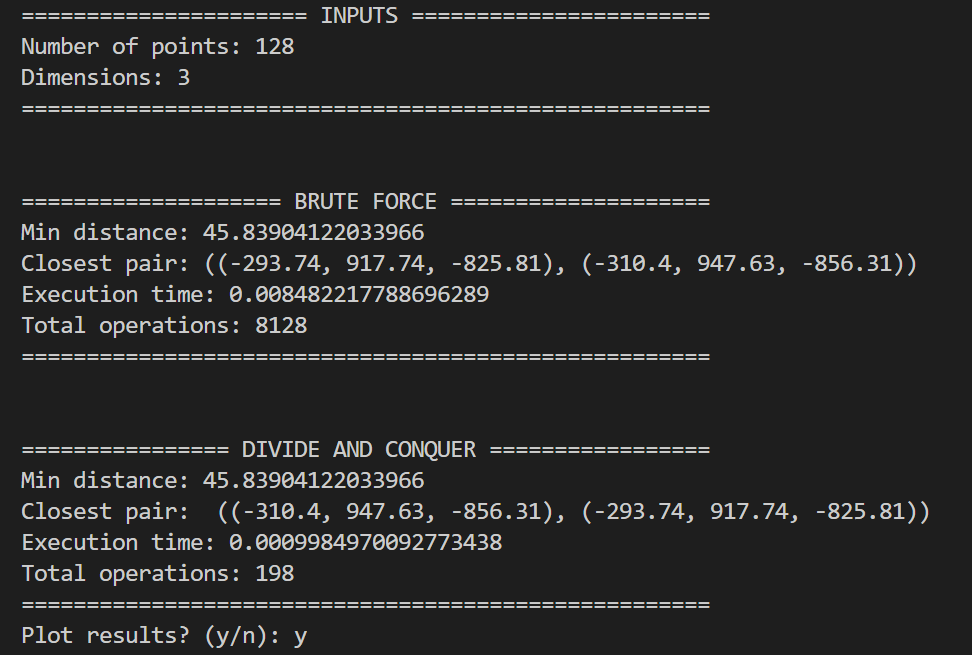
Text

Description automatically generated

Chart, scatter chart

Description automatically generated

1. n = 128



Chart, scatter chart

Description automatically generated

1. n = 1000

Text

Description automatically generated

Chart, scatter chart

Description automatically generated

1. dimensi ≠ 3

Text

Description automatically generated

**Text

Description automatically generated**

**Text

Description automatically generated**

**BAB V**

**LINK TO REPOSITORY**

<https://github.com/haziqam/Tucil2_13521170>

**DAFTAR PUSTAKA**

1. Levitin, A. (2014). *Introduction to the Design and Analysis of Algorithms: International Edition*. Pearson Education.
2. *3D scatterplot - Matplotlib 3.7.0 documentation*. Diakses dari <https://matplotlib.org/stable/gallery/mplot3d/scatter3d.html> pada 26 Februari 2023.
3. Munir, Rinaldi. 2021. Algoritma *Divide and Conquer* (Bagian 2). Bahan Kuliah IF2211 Strategi Algoritma. Diakses dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy(2021)-Bag1.pdf> pada 26 Februari 2023