

PRESENTASI TUGAS BESAR STRATEGI ALGORITMA

Penerapan algoritma Branch and Bound dan Greedy untuk mengelilingi tempat wisata terkenal di Bandung pada liburan akhir semester





PENGANTAR

Waktu liburan tidak lama lagi akan tiba dan salah satu cara untuk mengisi liburan adalah dengan mengunjungi tempat-tempat wisata. Di sini kami menggunakan algoritma BnB dan Greedy untuk mencari rute perjalanan terbaik.

ANGGOTA KELOMPOK

Khalillullah Al Faath
(1301204376)

Naufal Abdurrahman
Burhani (1301204008)

Ariqo Sukma Bahamis
Ali (1301200361)

ALGORITMA BNB

Algoritma Branch and Bound atau biasa disingkat sebagai BnB adalah strategi yang biasanya digunakan untuk menyelesaikan masalah optimasi dengan menggunakan pohon ruang status untuk pencarian solusi dan juga mempertimbangkan batas tertentu dalam konstruksi pohon ruang statusnya.



ALGORITMA GREEDY

Algoritma greedy adalah metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi yang mana pada algoritma greedy, persoalan dipecahkan secara langkah per langkah sedemikian sehingga pada setiap langkah:

1. mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip “take what you can get now!”)
2. dan “berharap” bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global (Munir, 2022).

KOMPONEN GREEDY

Himpunan Kandidat

Himpunan Solusi

Fungsi solusi

Fungsi seleksi

Fungsi kelayakan

Fungsi objektif

TSP

Travelling Salesman Problem atau biasa disingkat sebagai TSP adalah suatu permasalahan NP-hard dalam optimasi kombinatorik. Prinsip dari TSP ini adalah pertanyaan berikut:

“Diberikan larik yang terdiri atas kota-kota dan jarak antara pasangan kota-kota yang ada. Tentukan rute terpendek yang mana setiap kota dikunjungi tepat sekali dan kembali ke kota awal?”



TEMPAT WISATA

Objek wisata adalah suatu tempat yang menjadi kunjungan pengunjung karena mempunyai sumber daya, baik alami maupun buatan manusia, seperti keindahan alam atau pegunungan, pantai flora dan fauna, kebun binatang, dan lain-lain.



JARAK ANTAR TEMPAT WISATA DI BANDUNG

i/j	1	2	3	4	5	6	7	8	9	10
1	~	6.2	16.4	21.6	8.3	10.1	18.6	9.7	34	9.8
2	13.8	~	14.1	19.2	7.1	7.8	14	8.5	18.9	8.5
3	10.2	12.2	~	5.5	6.8	6.9	11.4	8.3	8.4	8.8
4	9.4	18.4	6.3	~	12.9	13	17	14.1	2.7	14.1
5	6.6	4.7	7	12.2	~	0.2	11.5	2.1	11.8	2.1
6	7.6	5.8	7.7	12.9	0.8	~	11.2	3.1	12.7	3.1
7	17.8	14	11.4	16.2	11.6	11.7	~	13.9	19.1	14.3
8	6.5	4.9	8.1	13.3	1.9	2.8	13.5	~	11.6	1.7
9	6.3	15.6	9	2.7	11.5	11.6	20.1	11.9	~	11.9
10	7.2	4	8.9	14,1	2.7	3.6	14.9	0.8	12.4	~

1.

Telkom University
2.

Trans Studio Bandung (Cibangkong, Batununggal)
3.

Dago Dream Park (Mekarwangi, Pagerwangi)
4.

Floating Market Lembang (Kabupaten Bandung Barat)
5.

Gedung Sate(Cihaur Geulis)
6.

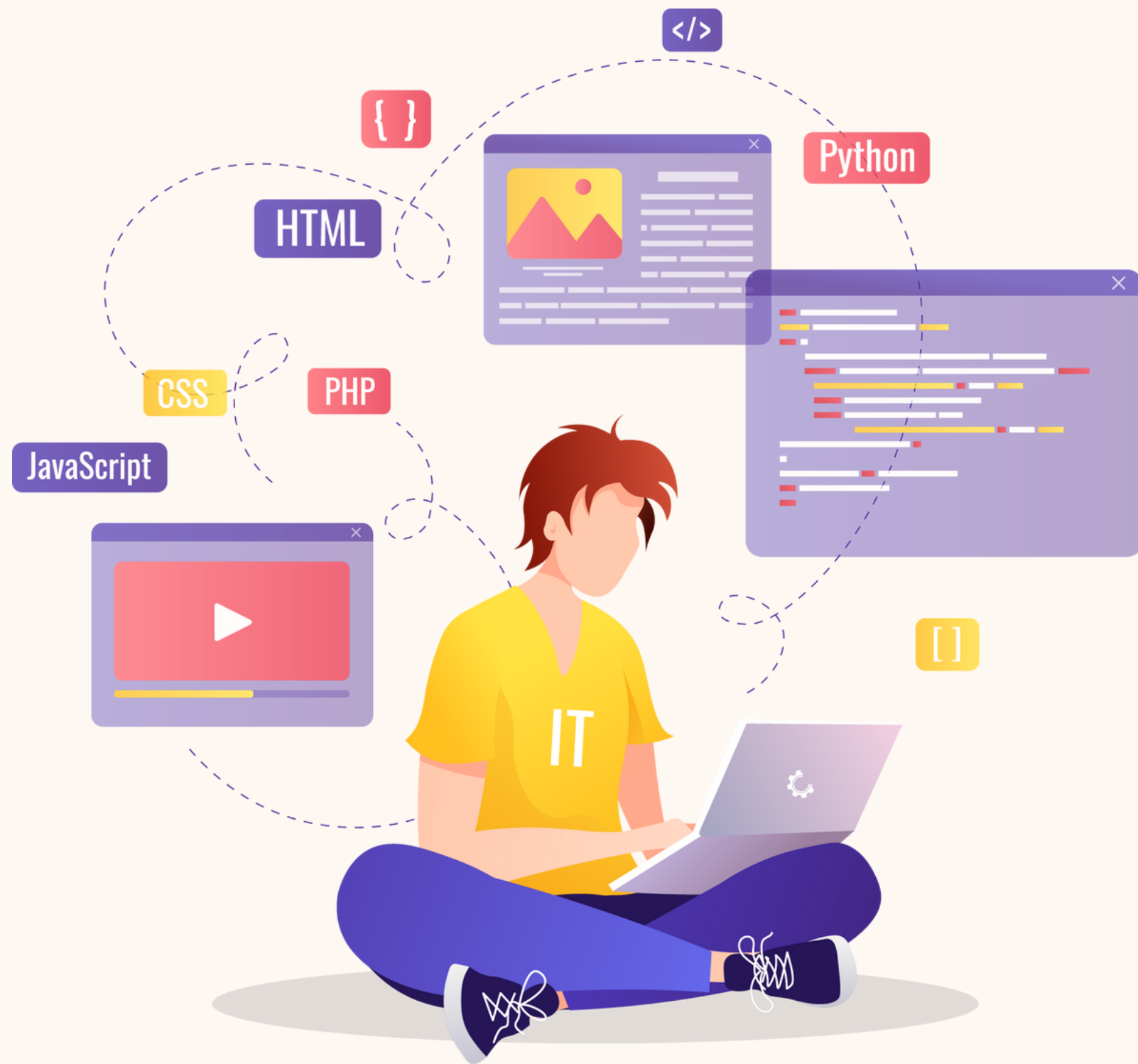
Museum Geologi Bandung (Cihaur Geulis, Kec. Cibeunying Kaler)
7.

Bukit Moko
8.

Taman Balai Kota Bandung (Kota Bandung)
9.

The Great Asia Afrika (Gudangkahuripan, Lembang, Kabupaten Bandung Barat)
10.

Jalan Braga (Kota Bandung)



IMPLEMENTASI PROGRAM

IMPORT MATRIKS JARAK DAN FUNGSI PENGUBAH NAMA

```
import pandas as pd
import numpy as np

df = pd.read_excel("https://github.com/
khalilullahalfaath/SA_Tubes/blob/
635eb73abab63587448b08bfe716e7703b5803d3/
Jarak%20antar%20tempat%20wisata%20di%20Bandung.xlsx?
raw=true")
```

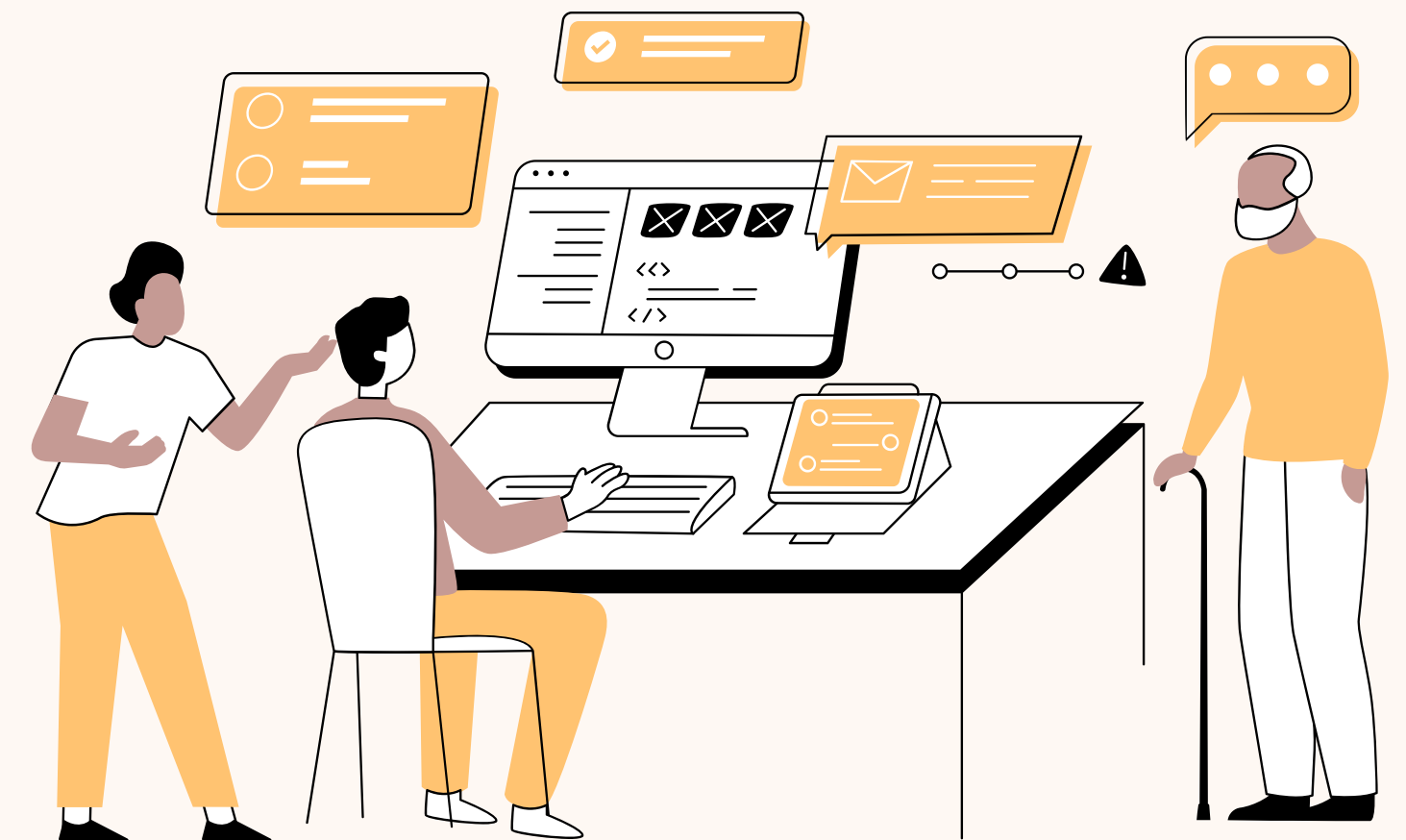
✓ 2.6s

Python


```
def changeName(rute):
    arrNama = ["Telkom University", "Trans Studio
Bandung", "Dago Dream Park", "Floating Market
Lembang", "Gedung Sate", "Museum Geologi Bandung",
"Bukit Moko", "Taman Balai Kota Bandung", "The
Great Asia Afrika", "Jalan Braga"]
    for i in range(N):
        if (i < N-1):
            print(arrNama[rute[i]], end = ' -> ')
        else:
            print(arrNama[rute[i+1]], end="\n\n")
```

✓ 0.2s

Python







```
import math
maxsize = float('inf')

# Function to copy temporary solution
# to the final solution
def copyToFinal(curr_path):
    final_path[:N + 1] = curr_path[:]
    final_path[N] = curr_path[0]

    Function to find the minimum edge cost
    having an end at the vertex i
    def firstMin(adj, i):
        min = maxsize
        for k in range(N):
            if adj[i][k] < min and i != k:
                min = adj[i][k]

        return min
```

```
# function to find the second minimum edge
# cost having an end at the vertex i
def secondMin(adj, i):
    first, second = maxsize, maxsize
    for j in range(N):
        if i == j:
            continue
        if adj[i][j] <= first:
            second = first
            first = adj[i][j]

        elif(adj[i][j] <= second and
            adj[i][j] != first):
            second = adj[i][j]

    return second
```

```

def TSPRec(adj, curr_bound, curr_weight,
          level, curr_path, visited):
    global final_res

    # base case is when we have reached level N
    # which means we have covered all the nodes once
    if level == N:

        # check if there is an edge from
        # last vertex in path back to the first
        # vertex
        if adj[curr_path[level - 1]][curr_path[0]]
           != 0:

            # curr_res has the total weight
            # of the solution we got
            curr_res = curr_weight + adj[curr_path
                                         [level - 1]]\
                                         [curr_path
                                         [0]]

            if curr_res < final_res:
                copyToFinal(curr_path)
                final_res = curr_res

```

```

        if curr_res < final_res:
            copyToFinal(curr_path)
            final_res = curr_res

        return

    # for any other level iterate for all vertices
    # to build the search space tree recursively
    for i in range(N):

        # Consider next vertex if it is not same
        # (diagonal entry in adjacency matrix and
        # not visited already)
        if (adj[curr_path[level-1]][i] != 0 and
            visited[i] == False):
            temp = curr_bound
            curr_weight += adj[curr_path[level - 1]]
                           [i]

```

```

# different computation of curr_bound
# for level 2 from the other levels
if level == 1:
    curr_bound -= ((firstMin(adj,
curr_path[level - 1]) +
                    firstMin(adj, i)) /
                    2)
else:
    curr_bound -= ((secondMin(adj,
curr_path[level - 1]) +
                    firstMin(adj, i))
                    / 2)

# curr_bound + curr_weight is the
actual lower bound
# for the node that we have arrived on.
# If current lower bound < final_res,
# we need to explore the node further
if curr_bound + curr_weight < final_res:
    curr_path[level] = i
    visited[i] = True

```

```

# call TSPRec for the next level
TSPRec(adj, curr_bound,
curr_weight,
        level + 1, curr_path,
        visited)

# Else we have to prune the node by
resetting
# all changes to curr_weight and
curr_bound
curr_weight -= adj[curr_path[level - 1]]
[i]
curr_bound = temp

# Also reset the visited array
visited = [False] * len(visited)
for j in range(level):
    if curr_path[j] != -1:
        visited[curr_path[j]] = True

```

```
def TSP(adj):

    # Calculate initial lower bound for the root
    # node
    # using the formula 1/2 * (sum of first min +
    # second min) for all edges. Also initialize
    # the
    # curr_path and visited array
    curr_bound = 0
    curr_path = [-1] * (N + 1)
    visited = [False] * N

    # Compute initial bound
    for i in range(N):
        curr_bound += (firstMin(adj, i) +
                       secondMin(adj, i))

    # Rounding off the lower bound to an integer
    curr_bound = math.ceil(curr_bound / 2)

    # We start at vertex 1 so the first vertex
    # in curr_path[] is 0
    visited[0] = True
    curr_path[0] = 0
```

Driver code

Adjacency matrix for the given graph

```
adj = np.array(df)
```

```
N = 10
```

final_path[] stores the final solution

i.e. the // path of the salesman.

```
final_path = [None] * (N + 1)
```

visited[] keeps track of the already

visited nodes in a particular path

```
visited = [False] * N
```

Stores the final minimum weight

of shortest tour.

```
final_res = maxsize
```

```
TSP(adj)
```


TSP(adj)

```
print("Implementasi Branch and Bound")
print("-----")
print("Minimum cost :", final_res)
print("Path Taken : ", end = ' ')
for i in range(N + 1):
    if (i < N):
        print(final_path[i]+1, end = ' -> ')
    else:
        print(final_path[i]+1, end="\n\n")
changeName(final_path)
```





Komponen-komponen greedy:

1. Himpunan kandidat: Himpunan tempat wisata yang dapat dipilih
2. Himpunan solusi: Himpunan tempat wisata secara berurut dari kota asal dan kembali ke kota asal dengan jarak terpendek.

3. Fungsi seleksi

Memilih kota dengan jarak terpendek

4. Fungsi kelayakan

Memilih tempat wisata tepat sekali kecuali untuk kampus.

5. Fungsi objektif

Meminimasi rute perjalanan dari semua tempat wisata



```

def findMinRoute(tsp):
    sum = 0
    counter = 0
    i = 0
    j = 0
    mn = 999999999
    visitedRouteList = {}
    visitedRouteList[0] = 1
    route = [0] * len(tsp)
    while i < len(tsp) and j < len(tsp[i]):
        if counter >= len(tsp[i]) - 1:
            break
        if j != i and j not in visitedRouteList:
            if tsp[i][j] < mn:
                mn = tsp[i][j]
                route[counter] = j + 1
        j += 1
        if j == len(tsp[i]):
            sum += mn
            mn = 999999999
            visitedRouteList[route[counter] - 1] = 1
            j = 0
            i = route[counter] - 1

```

```

        i = route[counter] - 1
        counter += 1
        #routes += [np.where(tsp == mn)]
    i = route[counter - 1] - 1
    for j in range(0, len(tsp)):
        if i != j and tsp[i][j] < mn:
            mn = tsp[i][j]

    sum += mn
    print("Implementasi Greedy")
    print("-----")
    print("Minimum route: ", sum)
    rute = []
    for key, value in visitedRouteList.items():
        rute += [key]
    rute.append(0)
    N = 10
    print("Path Taken : ", end = ' ')
    for i in range(N + 1):
        if (i < N):
            print(rute[i]+1, end = ' -> ')
        else:
            print(rute[i]+1, end = "\n\n")
    changeName(rute)

```

Implementasi Branch and Bound

Minimum cost : 56.1

Path Taken : 1 -> 2 -> 5 -> 10 -> 8 -> 6 -> 7 -> 3 -> 4
-> 9 -> 1

Telkom University -> Trans Studio Bandung -> Gedung Sate
-> Jalan Braga -> Taman Balai Kota Bandung -> Museum
Geologi Bandung -> Bukit Moko -> Dago Dream Park ->
Floating Market Lembang -> Telkom University

HASIL OUTPUT

Implementasi Greedy

Minimum route: 66.9

Path Taken : 1 -> 2 -> 5 -> 6 -> 8 -> 10 -> 3 -> 4 -> 9
-> 7 -> 1

Telkom University -> Trans Studio Bandung -> Gedung Sate
-> Museum Geologi Bandung -> Taman Balai Kota Bandung ->
Jalan Braga -> Dago Dream Park -> Floating Market Lembang
-> The Great Asia Afrika -> Telkom University

IV. ANALISIS

A. Perhitungan analisis kompleksitas algoritma dalam $T(n)$

1) Strategi Branch and Bound

Pada konstruksi pohon ruang status dengan pendekatan BnB, setidaknya ada dua hal yang perlu untuk diperhatikan:

a) Level pohon

Level pohon pada kasus TSP adalah sebanyak n sesuai dengan jumlah tempat.

b) Jumlah anak

Jumlah anak simpul pada kasus TSP dengan Branch and Bound dalam kondisi worst-case sama dengan pendekatan brute-force yaitu $n!$. Namun, bisa berubah seiring dengan pruning state space tree tergantung dengan nilai lower boundnya.

2) Strategi Greedy

Jika pengurutan diabaikan, maka pada algoritma Greedy hanya melakukan pengecekan sebanyak $T(n) = n(n-1)+\dots+1$

B. Kelas efisiensi dalam bentuk $O(n)$

1) Branch and Bound

Pada BnB kompleksitas waktunya dalam $O(n)$ adalah $O(n \cdot n!)$.

2) Greedy

Pada greedy, kompleksitas waktunya dalam $O(n)$ adalah $O(n^2)$

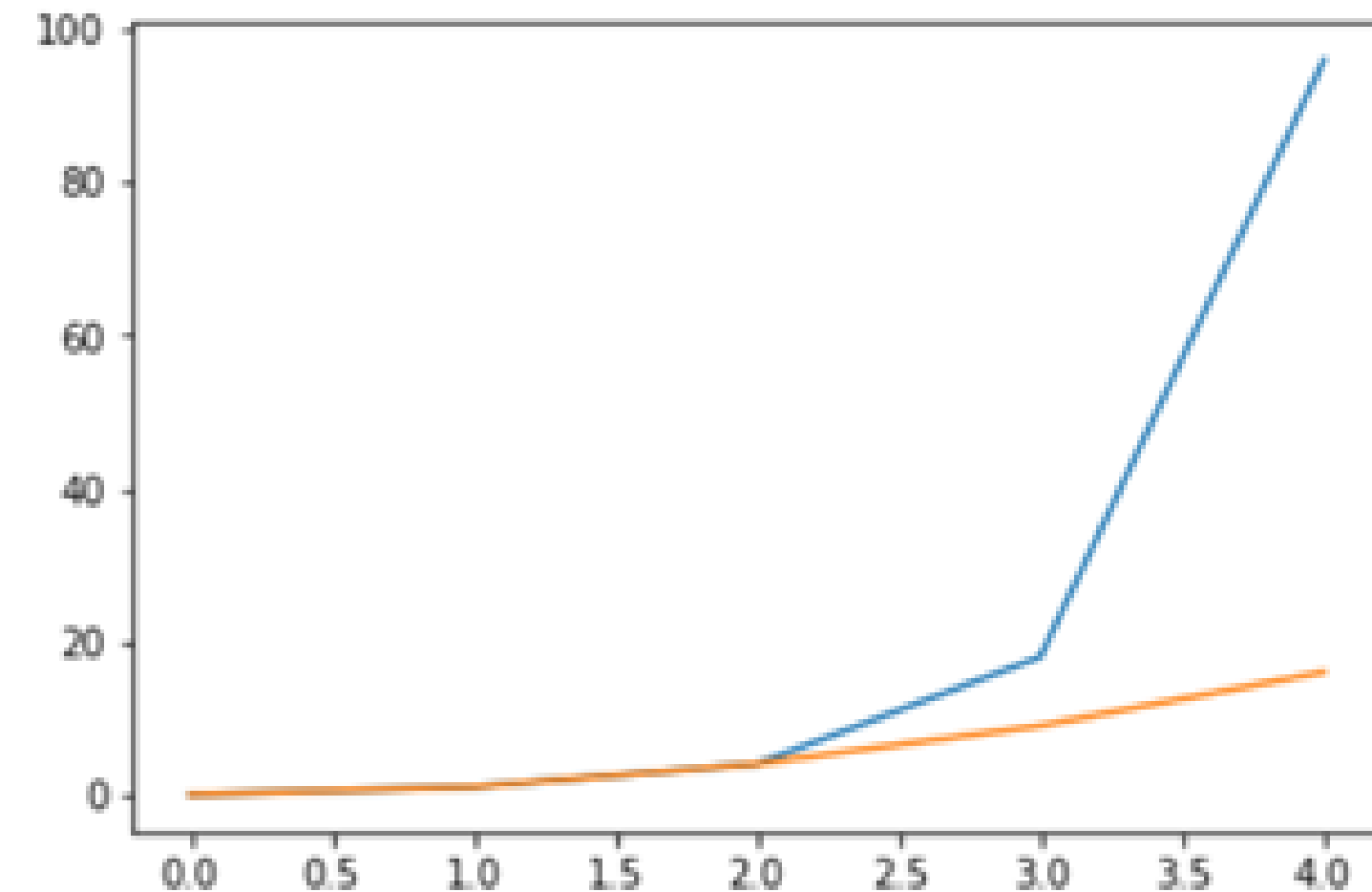
ANALISIS KOMPLEKSITAS



PERBANDINGAN KOMPLEKSITAS WAKTU

C. Perbandingan kompleksitas

Dapat dilihat bahwa kompleksitas pada greedy lebih mangkus daripada branch and bound. Namun, penyelesaian pada greedy tidak selalu optimal atau global optima.



Keterangan:

Biru: Kompleksitas BnB

Kuning: kompleksitas greedy

REFERENSI

- [1] Daniells, L. (2020, April 21). *The Travelling Salesman Problem*. Retrieved from lancaster.ac.uk: <https://www.lancaster.ac.uk/stor-i-student-sites/libby-daniells/2020/04/21/the-travelling-salesman-problem/>
- [2] GeeksforGeeks admin. (2022, Maret 25). <https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/>. Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/>
- [3] GeeksforGeeks admin. (2022, January 2022). *Travelling Salesman Problem | Greedy Approach*. Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/travelling-salesman-problem-greedy-approach/>
- [4] Munir, R. (2022, January). *Algoritma Greedy (Bagian 1)*. Retrieved June 23, 2022, from informatika.stei.itb.ac.id: [http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-\(2022\)-Bag3.pdf](http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf)
- [5] Wulandari, G. S., & Sa'adah, S. (2021). Buku Pengantar Strategi Algoritma. In G. S. Wulandari, & S. Sa'adah, *Buku Pengantar Strategi Algoritma* (1 ed., p. 133). Bandung, Jawa Barat, Indonesia: CV. Karya Bakti Makmur Indonesia. Retrieved June 21, 2022, from <https://drive.google.com/file/d/1OLaEBWRhIkWD0HlqbBOKTk3t26p4sfs/view>

REFERENSI





THANK
YOU

TERIMA
KASIH

