IMPLEMENTASI GREEDY DAN BRUTE FORCE PADA PENCARIAN BIAYA MINIMUM DENGAN TABEL SEIMBANG

TUGAS BESAR 1 MATEMATIKA LANJUT

Oleh

Aditya Sudyana (23222063)
Atikasari (23222049)
Ananda Sadam Firdaus (23222040)
Lukman Eka Arifandhi (23222075)
Naufal Fahmi Zakiuddin (23222070)

(Program Studi Magister Teknik Elektro)

INSTITUT TEKNOLOGI BANDUNG

November 2022PROGRAM OPTIMUM TRANSPORTATION COST DENGAN VOGELS APPROXIMATION DAN BRUTEFORCE

Rincian program optimum transportation cost degan vogels approximation dan bruteforce telah diupload di github pada link berikut :

GitHub - aditsud/greedy_bruteforce: Untuk Tugas Matematika Lanjut
https://github.com/aditsud/greedy_bruteforce

Dengan program demo pada link berikut :

Greedy Brute Force by Olaf (greedy-olaf.herokuapp.com)
https://greedy-olaf.herokuapp.com/

ANALISIS TERHADAP KOMPLEKSITAS ALGORITMA YANG DIGUNAKAN

Analisis kompleksitas algoritma yang digunakan diawali dengan menggambarkan diagram alir dari tiap fungsi. Aplikasi ini terdapat dua fungsi besar yang dipanggil dari proses yang utama sebagai berikut :

1. Proses *greedy* merupakan proses dalam proses awal dalam menentukan optimum cost. Pada proses ini diawali dengan menentukan penalty dari baris atau kolom yaitu dengan mengurangi biaya terkecil dan yang terkecil kedua. Penalty terbesar akan menjadi kandidat pengambilan cost. Begitu seterusnya hingga semua supply dan demand terpenuhi. Dengan begitu proses greedy dapat digambarkan dalam flowchart pada gambar 2. Dalam flowchart terlihat bahwa proses greedy juga terbagi atas fungsi fungsi yang lain. Antara lain :
    a. determinePenalty : digunakan untuk menghitung penalty setiap baris dan kolom. Pada flowchart terlihat bahwa kerumitan pada algoritma ini adalah $2n^3+2n^2$ pada kondisi terburuk, $(2 + 2\log n)\ n^2$ pada kondisi terbaik dan $(2 + 2\log n)\ n^2$ pada kondisi rata-rata
    b. getAreaofInterest : mendapatkan area baris atau kolom yang menjadi kandidat dalam mendapatkan cost terbaik. Pada algoritma tersebut terlihat bahwa kerumitannya adalah n^2+2n pada kondisi terburuk, $(2+2\log n)\ n$ pada kondisi terbaik dan $(2+2\log n)\ n$ pada kondisi rata-rata
    c. findFirstLowCost : digunakan untuk mendapatkan cost terbaik pada area yang menjadi kandidat. Pada algoritma ini terlihat bahwa kerumitannya adalah $3n^2$ dalam kondisi terburuk, $2n^2+n(2\log n)$ pada kondisi terbaik, dan $2n^2+n(2\log n)$ pada kondisi rata-rata.
    d. checkRowColLeft : digunakan untuk melihat sisa baris dan kolom. Pada fungsi ini kerumitannya 4n baik dalam kondisi terbaik, terburuk maupun rata-rata
    e. satisfyTheSupplyandDemand : menentukan nilai vogel optimal dengan membandingkan supply dan demand tersisa. Dalam fungsi ini terlihat bahwa kerumitannya adalah 1 baik dalam kondisi terbaik, terburuk, maupun rata-rata
    f. countTheVogelCost : Menghitung total cost. Pada fungsi ini terlihat bahwa kerumitannya adalah n^2

    Dari semua fungsi tersebut, algoritma greedy ini memiliki kerumitan :

- Kondisi terburuk : $(2n^3+2n^2 + n^2+2n + 3n^2 + 4n + 1) * n + n^2 = 2n^4+6n^3+7n^2+n$. Maka kompleksitas waktu asimptotiknya adalah $O(n^4)$
- Kondisi terbaik : $((2 + 2\log n)\ n^2 + (2+2\log n)\ n + 2n^2+n(2\log n) + 4n + 1) *n + n^2 = (4 + 2\log n)\ n^3 + (7 + 2\ {}^2\log n)\ n^2 + n$. Maka kompleksitas waktu asimptotiknya adalah $O(n^3)$
- Kondisi rata-rata : $((2 + 2\log n)\ n^2 + (2+2\log n)\ n + 2n^2+n(2\log n) + 4n + 1) *n + n^2 = (4 + 2\log n)\ n^3 + (7 + 2\ {}^2\log n)\ n^2 + n$. Maka kompleksitas waktu asimptotiknya adalah $O(n^3)$

2. Proses algoritma *brute force* diperhitungkan melalui fungsi calcBruteForce(). Dibutuhkan dua parameter untuk menjalankan fungsi ini matriks_cost dan matriks_vogels.

   Matriks *cost* merupakan matriks awal saat angka diinput dalam tabel *supply-demand*. Matriks vogel didefinisikan sebagai matriks yang berisi nilai dari hasil setetlah algoritma *Vogel Approximate* dijalankan. Terdapat tiga fungsi utama yang akan dijalankan dalam algoritma utama BruteForce(),

   - getSquare()

     Digunakan untuk menemukan pola kotak atau L di dalam matriks vogel, yang akan disimpan dalam larik square_list[]. kerumitan fungsi ini adalah n^2 dalam kondisi terbaik, n^7 dalam kondisi terburuk, dan n^4 pada kondisi rata rata

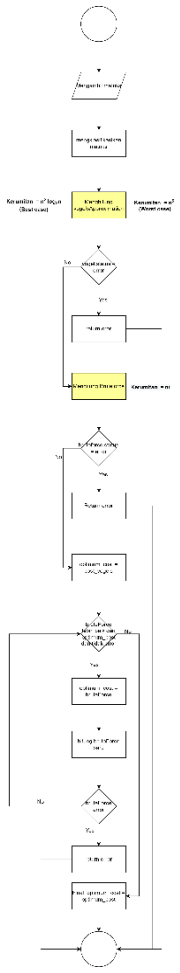
   - sort()

     Digunakan untuk mengurutkan item yang ada di dalam array. Kerumitan fungsi ini adalah 1 dalam kondisi terbaik, $2n^2$ pada kondisi terburuk, 2n (2logn) dalam kondisi rata-rata.

     Pada flowchart terlihat bahwa kompleksitass pada algoritma bruteforce adalah n! Dalam kondisi terburuk karena n dikombinasikan dengan n, sedangkan dalam kondisi terbaik adalah 1 karena perhitungan di vogel tidak ditemukan lagi pola pada bruteforce sehingga dianggap perhitungan di vogel telah optimum.

     Dari kedua algoritma besar tersebut adalah n^4 + n! Pada kondisi terburuk, dan n^4 pada kondisi terbaik dengan n^4 + n! Pada kondisi rata-rata. Algoritma ini terlihat sangat kompleks jika dilihat dari kerumitannya, tetapi sebelum memasuki brute force terlebih dahulu dijalankan algoritma vogel untuk mendekati nilai cost terbaik. Sehingga iterasi pada algoritma brute force diperkirakan tidak lebih dari 3 iterasi.

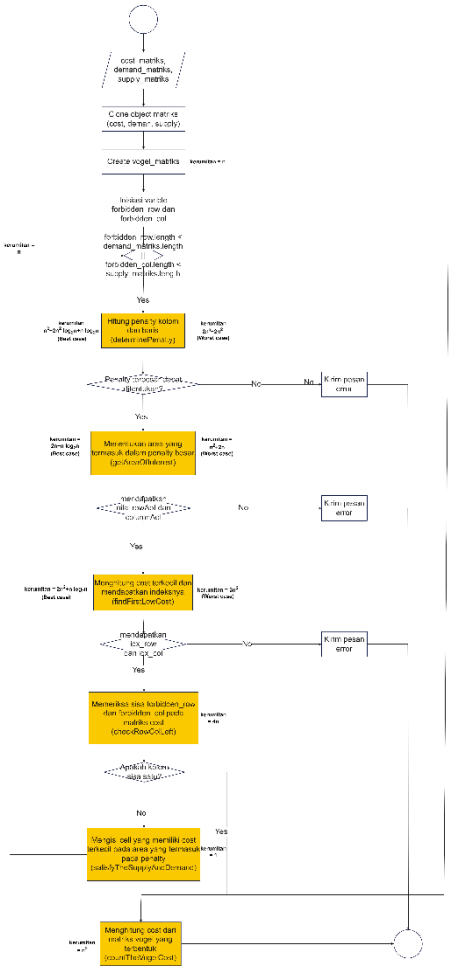
Flowchart algoritma yang digunakan pada bagian utama adalah sebagai berikut :



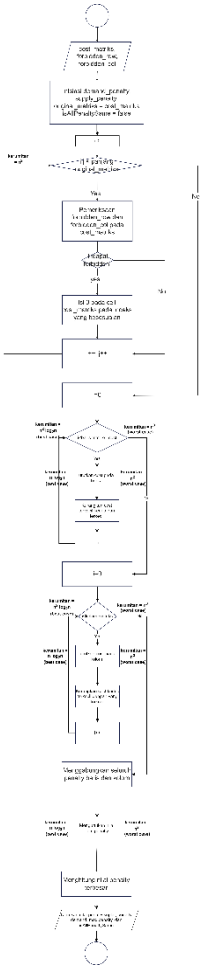
Gambar 1. Flowchart main

I. Greedy

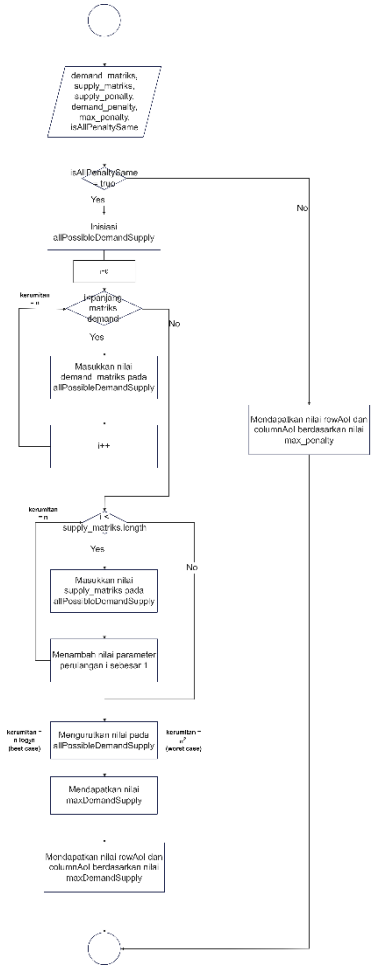
Rincian pada algoritma greedy sebagai berikut:



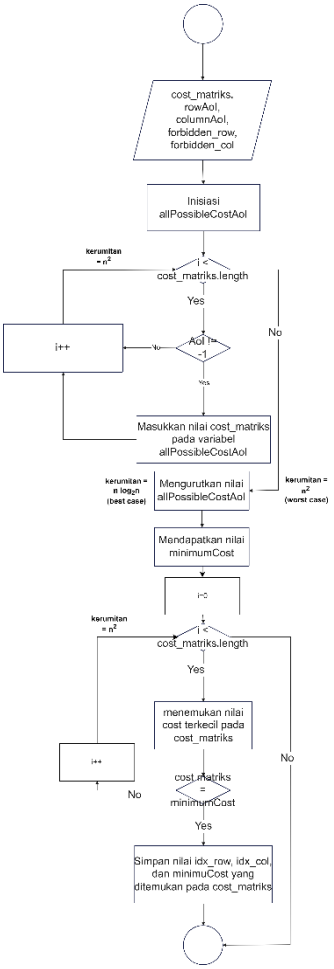
Gambar 2. Flowchart greedy – utama



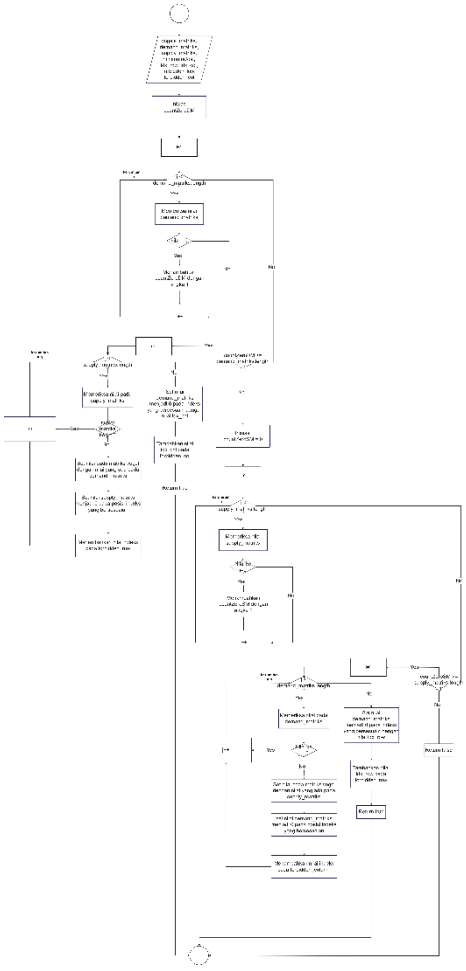
Gambar 3. DeterminePenalty



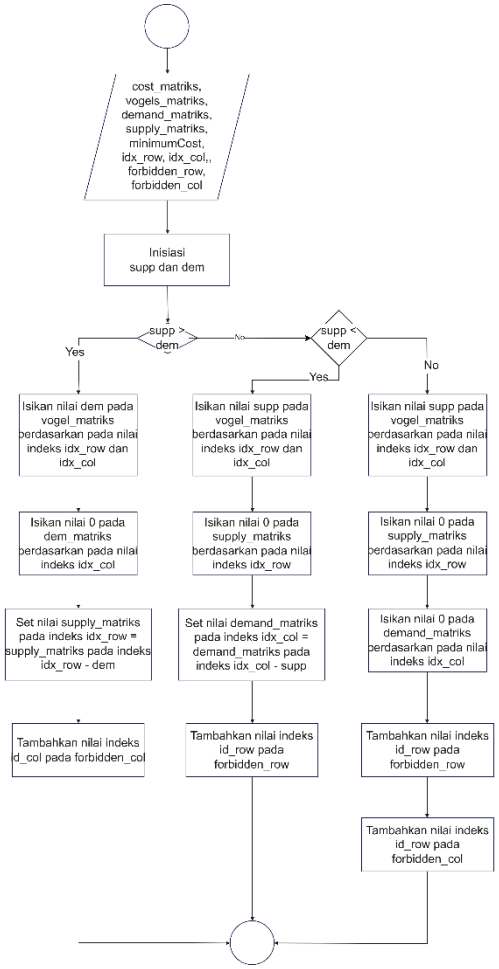
Gambar 4. getAreaofInterest



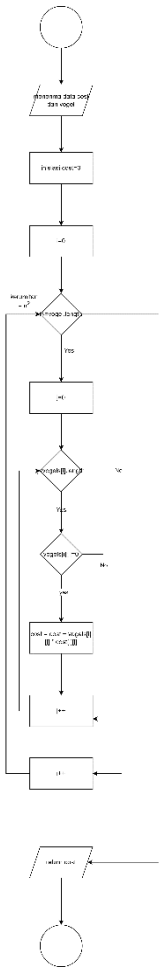
Gambar 5. findFirstLowCost



Gambar 6. checkRowColLeft



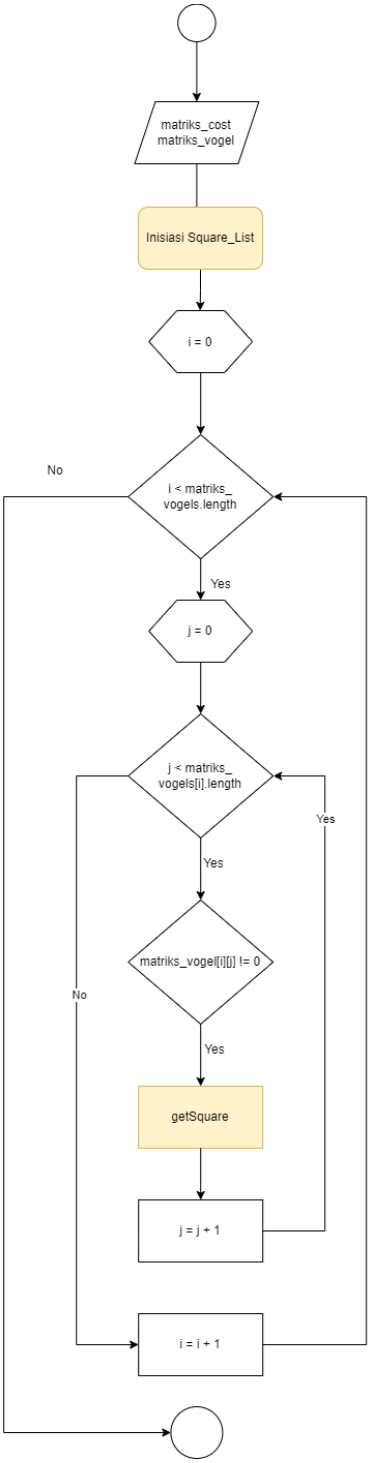
Gambar 7. satisfyTheSupplyandDemand



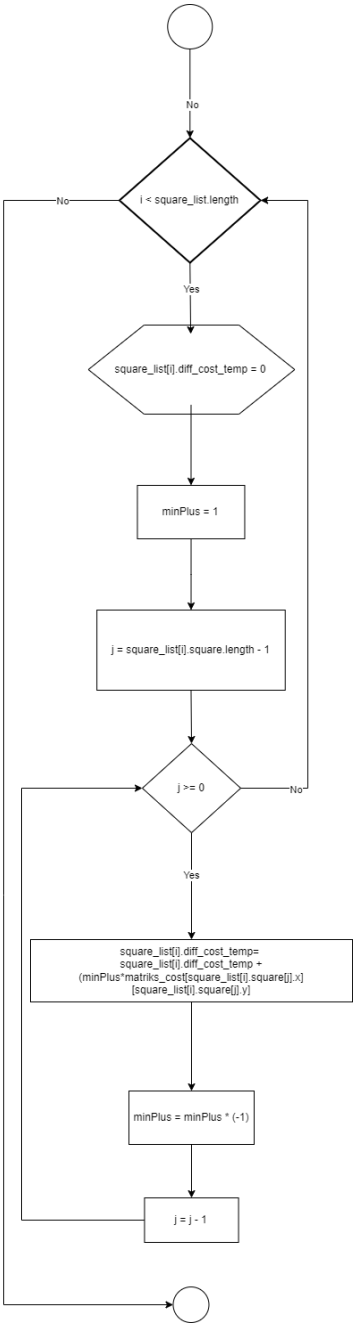
Gambar 8. countTheVogelCost

II. Brute Force

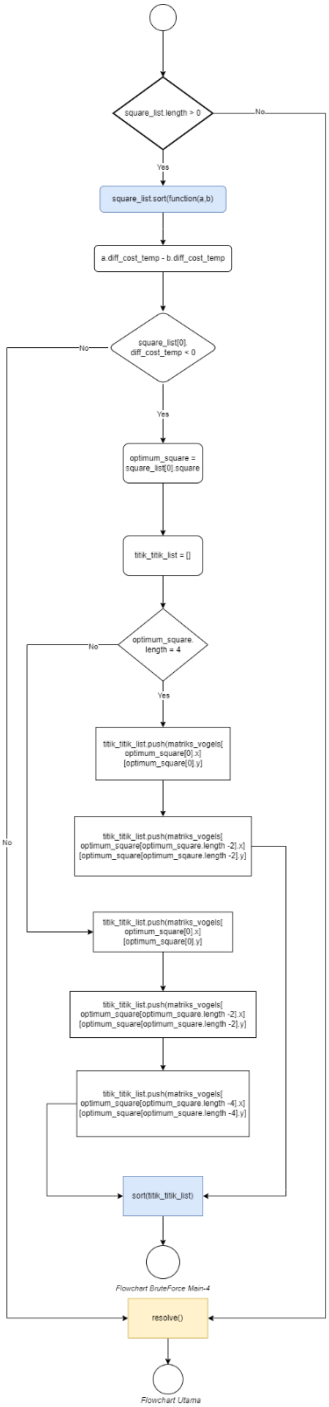
Diagram alir algoritma BruteForce yang diterapkan dalam program dapat digambarkan sebagai berikut,



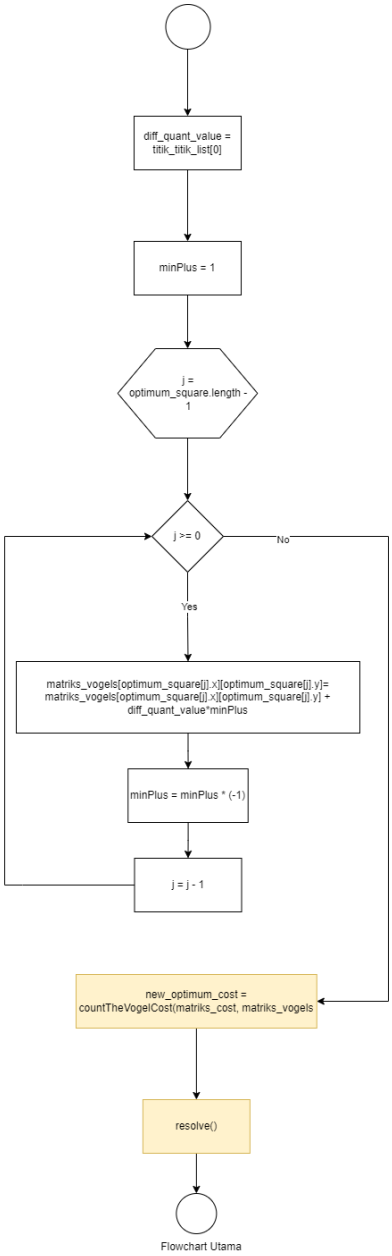
Gambar 2.1 BruteForce Main – 1



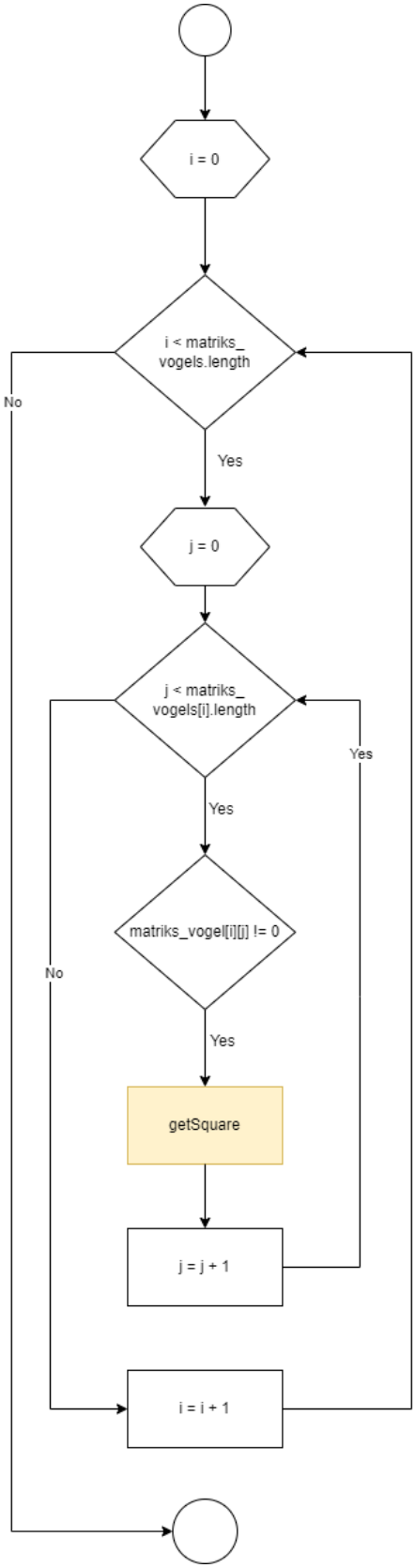
Gambar 2.2 BruteForce Main - 2



Gambar 2.3 BruteForce Main – 3

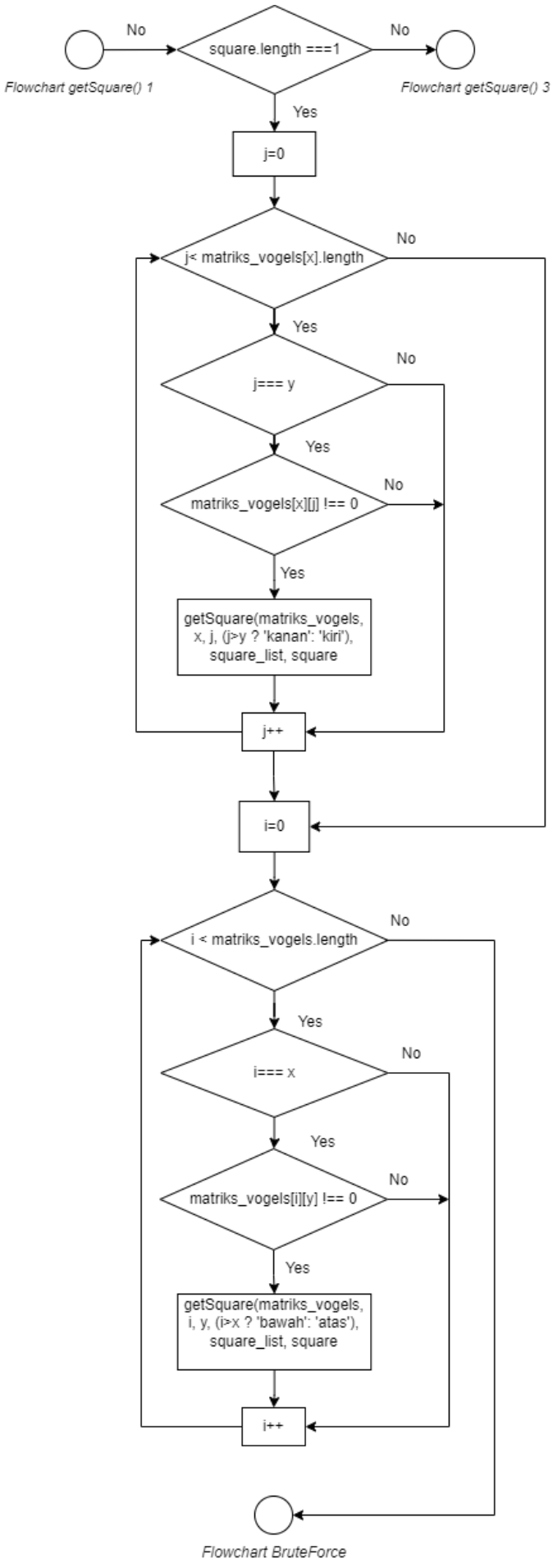


Gambar 2.4 BruteForce Main – 4

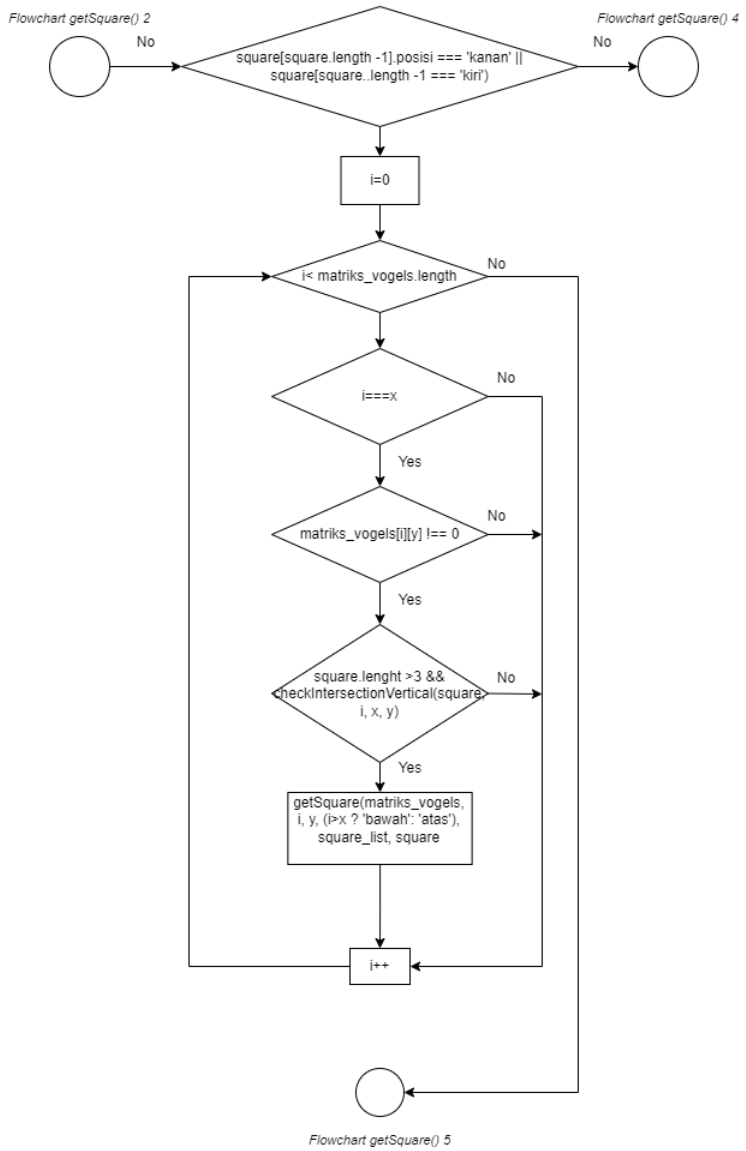


Gambar 2.5. square_list

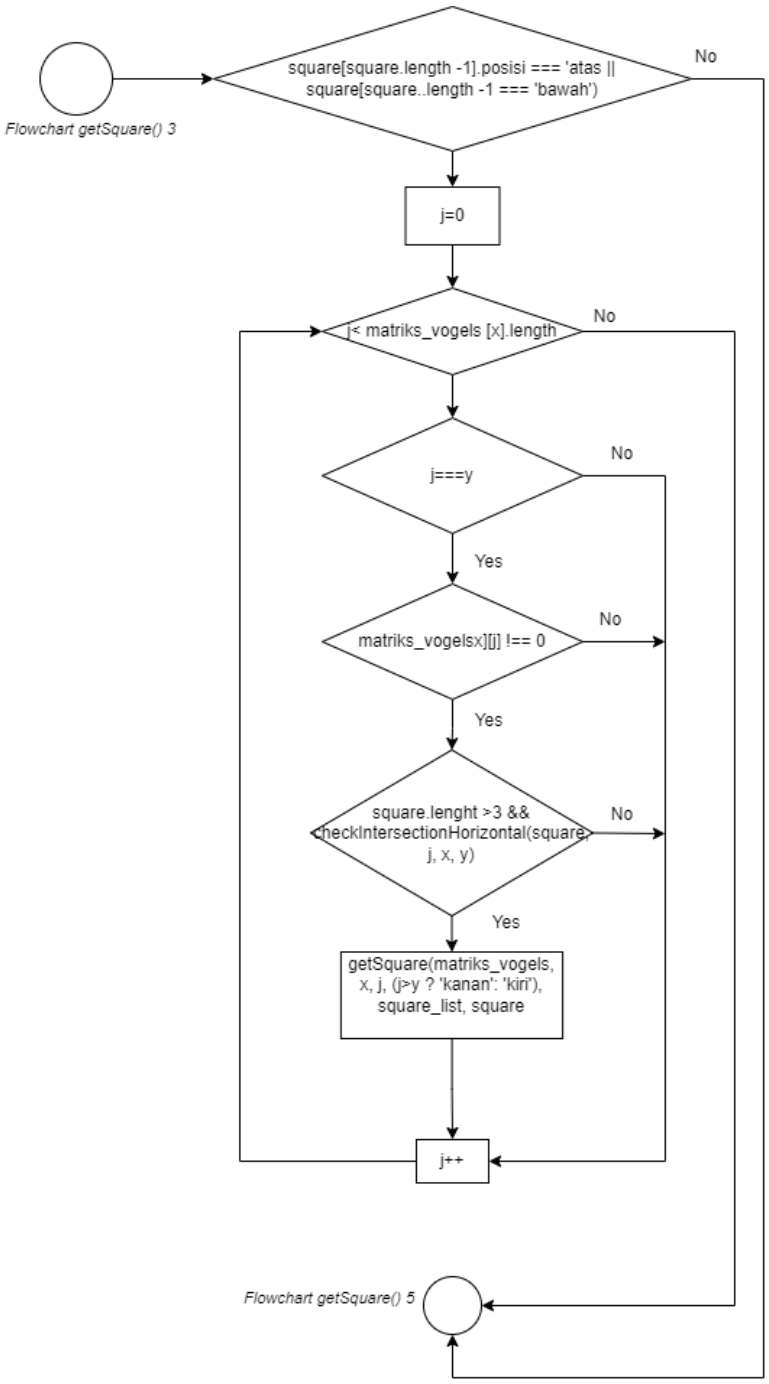
Gambar 2.6 getSquare() - 1


No
square.length ===1
No
Flowchart getSquare() 1
Flowchart getSquare() 3
Yes
j=0
j< matriks_vogels[x].length
No
Yes
j=== y
No
Yes
matriks_vogels[x][j] !== 0
No
Yes
getSquare(matriks_vogels, x, j, (j>y ? 'kanan': 'kiri'), square_list, square
j++
i=0
i < matriks_vogels.length
No
Yes
i=== x
No
Yes
matriks_vogels[i][y] !== 0
No
Yes
getSquare(matriks_vogels, i, y, (i>x ? 'bawah': 'atas'), square_list, square
i++
Flowchart BruteForce

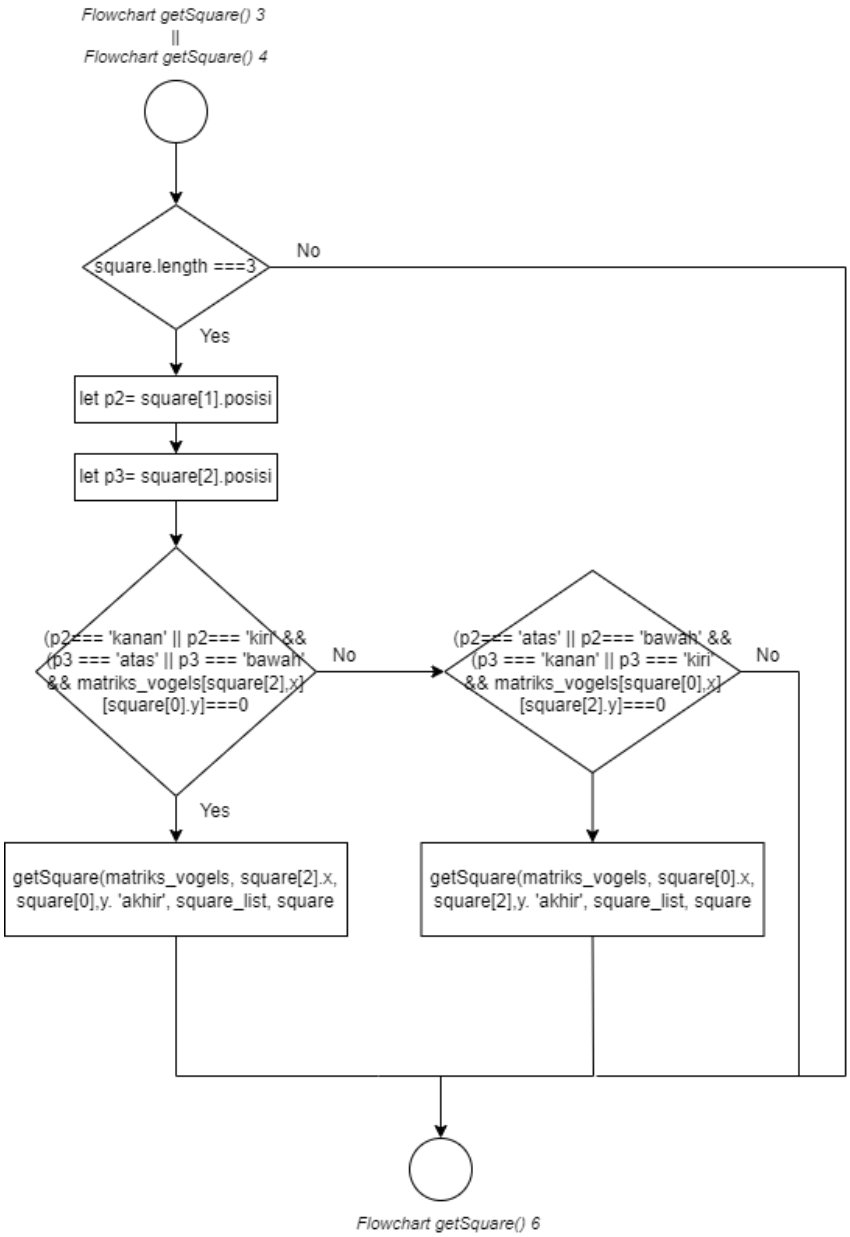
Gambar 2.7 getSquare() - 2



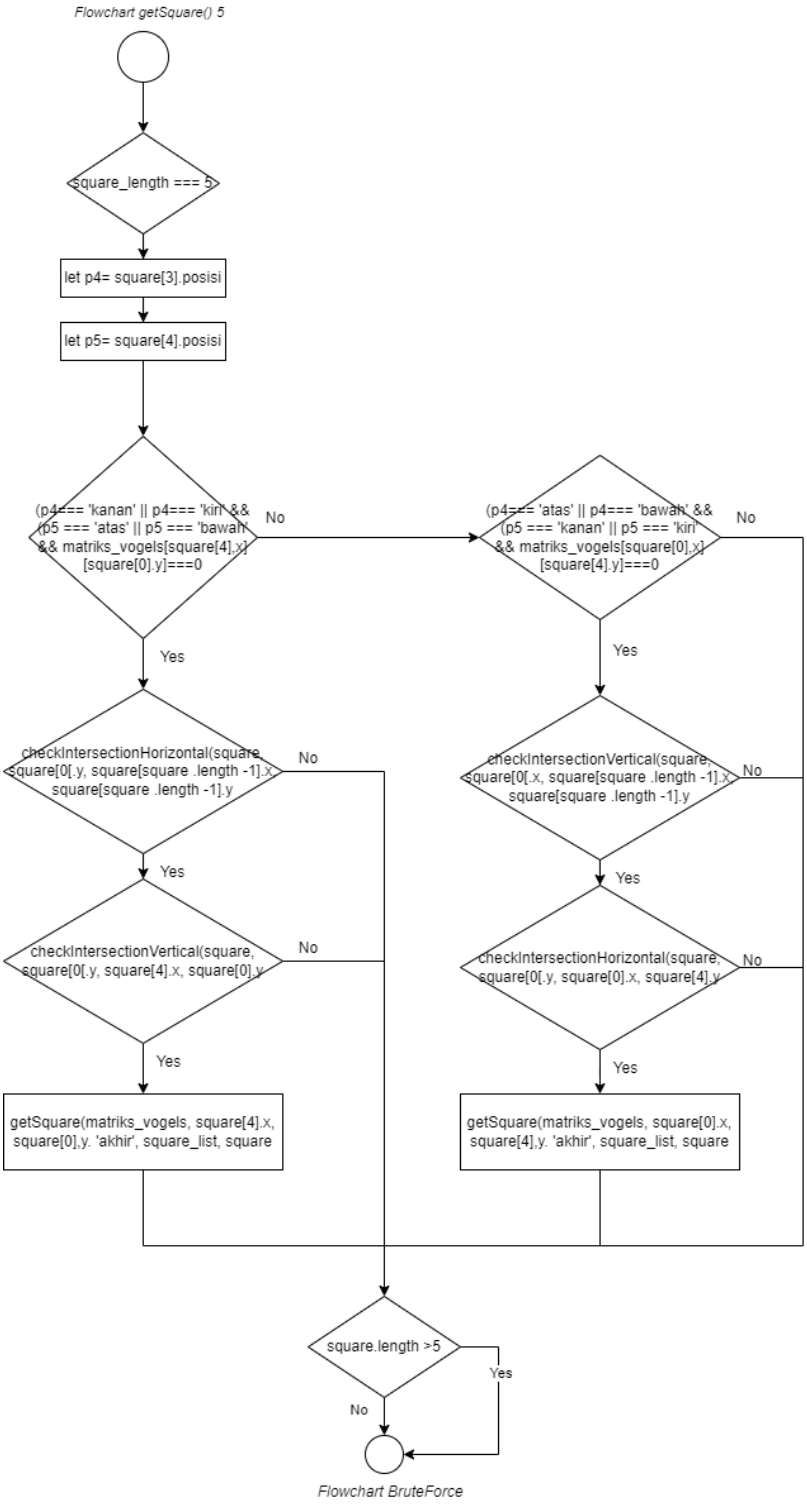
Gambar 2.8 getSquare() - 3



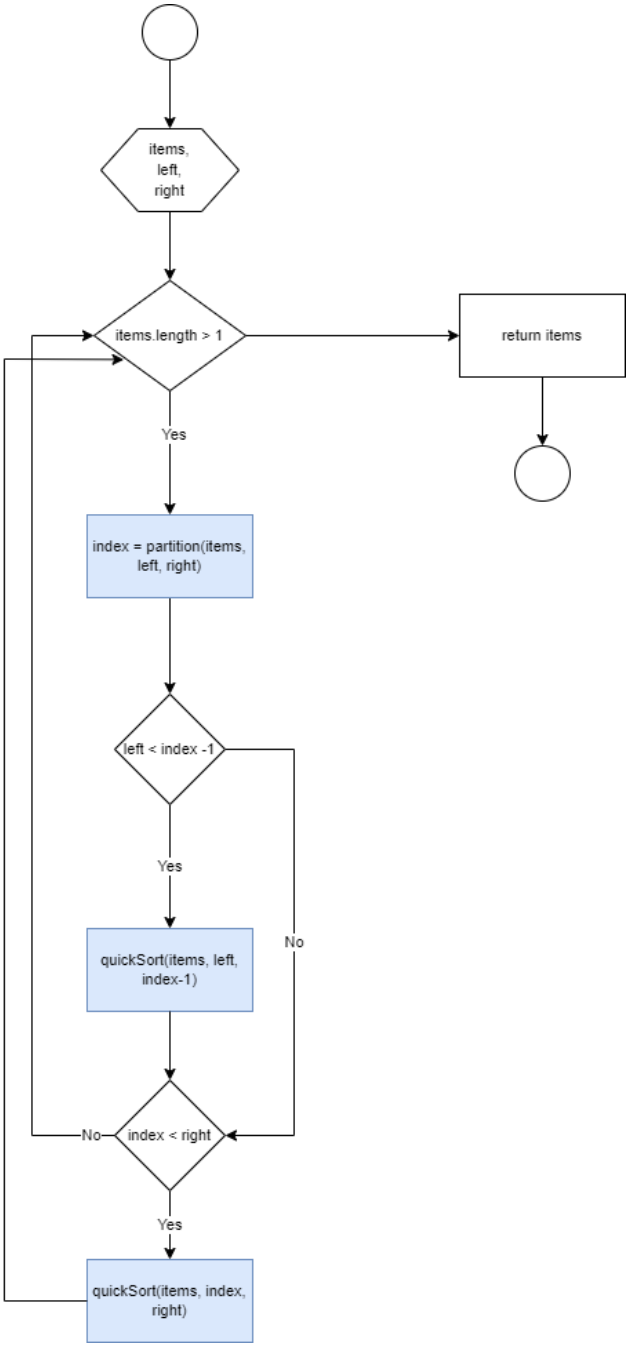
Gambar 2.9 getSquare() - 4



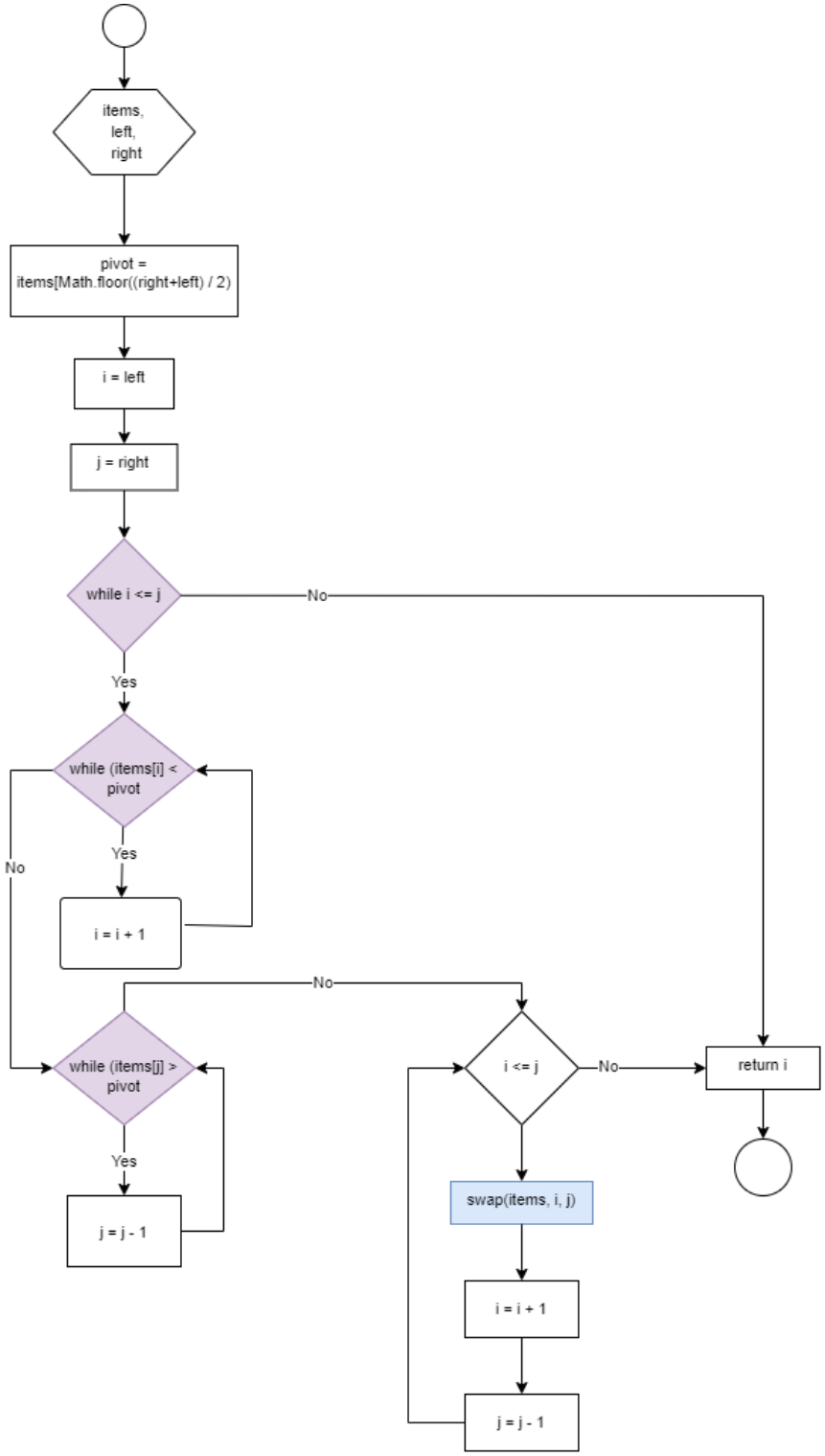
Gambar 2.10 getSquare() - 5



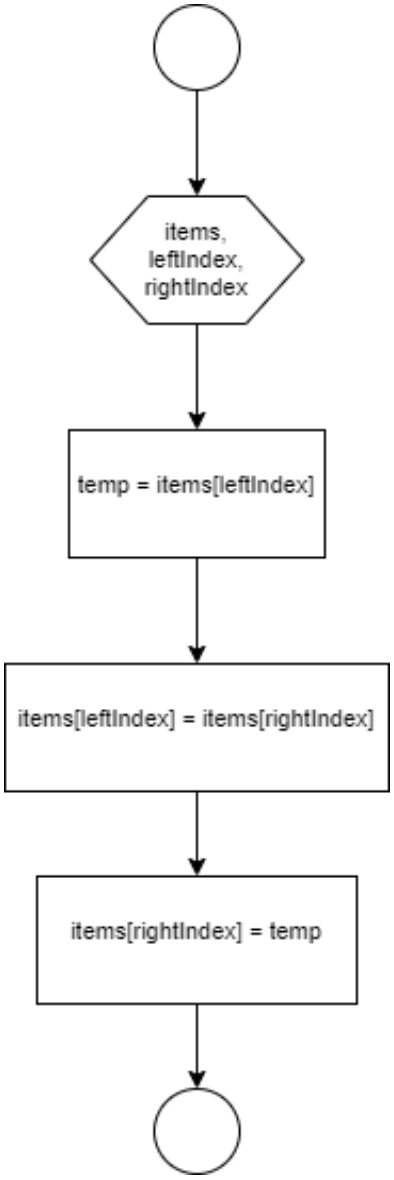
Gambar 2.11 getSquare() - 6



Gambar 2.11 quickSort()



Gambar 2.12 partition()



Gambar 2.13 swap()

Diagram alir secara lengkap dapat dilihat melalui link berikut,
https://drive.google.com/file/d/13zoNsfO60l9iZ2XtRatE0Z0XBxnpxaMZ/view?usp=sharing

PANDUAN CARA MENGGUNAKAN PROGRAM

Jika ingin menggunakan aplikasi web ini secara online, Anda dapat membukanya di https://greedy-olaf.herokuapp.com/ . Akan tetapi jika ingin dipasang pada localhost komputer Anda, berikut adalah langkah-langkahnya:

1. Install NodeJS pada komputer Anda (https://nodejs.org/en/)
2. Clone Repository Github ini ke komputer Anda
3. Buka Command Prmpot/Terminal pada folder tempat repo ini Anda clone, kemudian jalankan npm install
4. Setelah itu, jalankan npm run dev untuk menjalankan secara lokal