

Ohjelmistokehityksen teknologioita - Seminaarityö

Quicksort algoritmi

3 Tietorakenteet ja Algoritmit

Victor Cherkasov

Sisältö

Tiivistelmä	1
1 Johdanto/Ylätason esittely	1
2 Käytetyt tekniikat	1
2.1 Valmistautuminen	1
2.1.1 Kansion ja tiedoston luominen.	1
2.2 Tiedon hakeminen ja tutustuminen algoritmiin.....	2
2.2.1 Mikä on algoritmi?	2
2.2.2 Yksinkertaisia algoritmeja.....	2
2.2.3 Mitä on Quicksort algoritmi?.....	3
2.3 Quicksort algoritmin luominen VScode:een	3
2.3.1 Tehtävä.....	3
2.3.2 Suunnittelu.....	4
2.3.3 Toteutus ja selitys.....	4
2.3.4 Debug	5
2.4 Testaaminen	6
2.4.1 1000 alkion listalla	6
2.4.2 10000 alkion listalla	6
2.4.3 Yhteenveto.....	7
2.5 Vastaus kysymyksiin.....	7
2.5.1 Missä tilanteissa Quicksort-algoritmia käytetään?.....	7
2.5.2 Missä tilanteissa Quicksort-algoritmia vältetään?.....	7
2.5.3 Quicksort-Algoritmin hyödyt ja haitat	7
3 Algoritmin vuokaavio	8
4 Yhteenveto / Tulokset / Johtopäätökset / Arviointia / Pohdinta tai muun niminen loppuluku	8
4.1 Yhteenveto.....	8
4.2 Tulokset.....	8
Lähdeluettelo.....	9

Tiivistelmä

Tavoitteena on opiskella Quicksort algoritmia, rakentaa omaa Quicksort algoritmia Pythonin avulla ja lopuksi analysoida algoritmia Big O notaation avulla.

1 Johdanto/Ylätason esittely

Huhtikuun alussa sain luettua Grokking Algorithms kirjoittaja Aditya Bhargava, missä käsitellään perusalgoritmia, jotka voi hyödyntää työympäristössä mm: Quicksort, Dijkstran, Greedy:n, KNN algorimia. Kaikki algoritmit ovat esitelty kuvien kautta, mikä aiheuttaa vaikeuksien ymmärryksessä sekä hahmotuksessa tietyn algoritmin toiminnallisuutta.

Mielestäni tietorakenteet ja algoritmit ovat ohjelmoinnin perusasioita, jotka täytyy ymmärtää joka ikisessä ohjelmoinnin kielessä ts. tietorakenteet ja algoritmit 20 % perusasiasta ohjelmoinnin maailmassa. Mainitsin ylhäällä, että kirjan sain luettua huhtikuun alussa ja nyt on erinomainen hetki tutkia yhdestä olevasta algoritmista.

Tutkimuksen tarkoitus on tutustua Quicksort algoritmiin ja osata soveltaa oikeissa tilanteissa sekä samalla vastata seuraaviin tutkimuskysymyksiin:

1. Mitä on Quicksort algoritmi?
2. Missä tilanteissa Quicksort-algoritmia käytetään?
3. Missä tilanteissa Quicksort-algoritmia vältetään?
4. Quicksort-Algoritmin hyödyt ja haitat

2 Käytetyt tekniikat

2.1 Valmistautuminen

2.1.1 Kansion ja tiedoston luominen.

```
admin@Victors-MBP softwareTechnologies % mkdir seminar_quicksort
admin@Victors-MBP softwareTechnologies % code seminar_quicksort
admin@Victors-MBP softwareTechnologies % touch seminar_quicksort/quicksort.py
admin@Victors-MBP softwareTechnologies % touch seminar_quicksort/examples.py
admin@Victors-MBP softwareTechnologies %
```

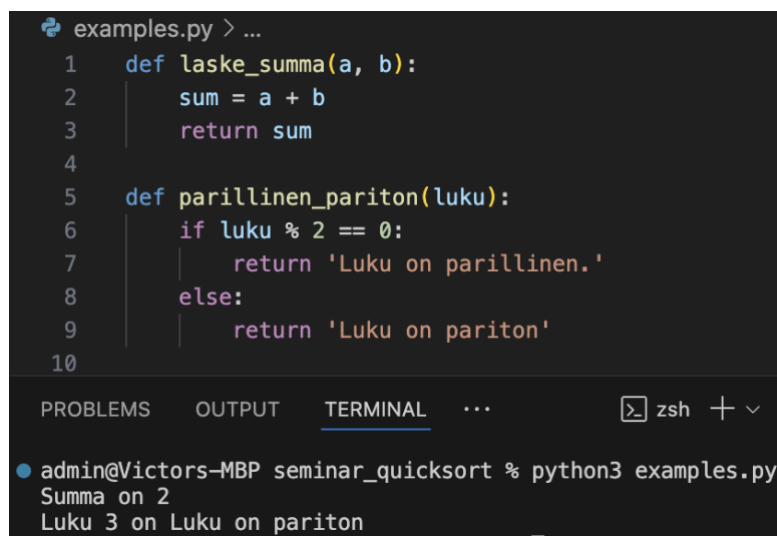
2.2 Tiedon hakeminen ja tutustuminen algoritmiin

2.2.1 Mikä on algoritmi?

Algoritmi on joukko tietyllä kielellä kirjoitettuja sääntöjä, jotka auttavat suorittamaan tietyn tehtävän tai ratkaisemaan ongelman.

2.2.2 Yksinkertaisia algoritmeja

Liitteenä on yksinkertaisia algoritmeja, missä annetaan x luku/luvut ja palautetaan saatu arvo.



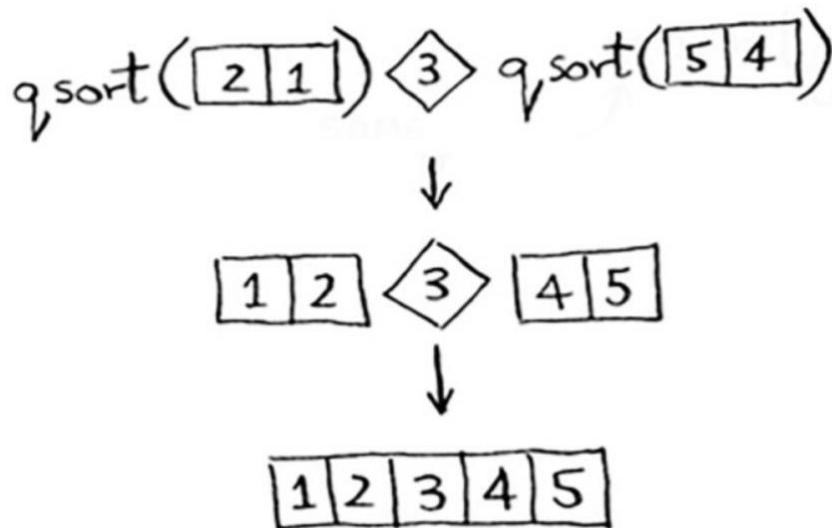
```
examples.py > ...
1  def laske_summa(a, b):
2      sum = a + b
3      return sum
4
5  def parillinen_pariton(luku):
6      if luku % 2 == 0:
7          return 'Luku on parillinen.'
8      else:
9          return 'Luku on pariton'
10

PROBLEMS  OUTPUT  TERMINAL  ...  zsh + v
● admin@Victors-MBP seminar_quicksort % python3 examples.py
Summa on 2
Luku 3 on Luku on pariton
```

2.2.3 Mitä on Quicksort algoritmi?

Quicksort on järjestysalgoritmi, joka valitsee pivot-alkion ja lajittelee osataulun rekursiivisesti, kunnes koko taulu on järjestetty suurjärjestyksessä

Seuraavassa esimerkissä valitaan luku 3 pivot-alkioksi.



Kuva 1

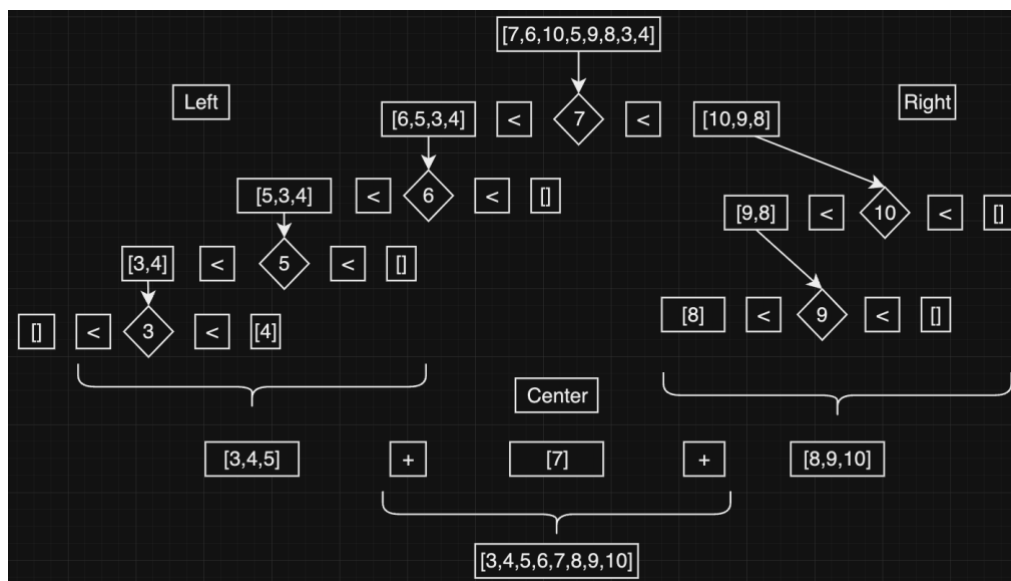
2.3 Quicksort algoritmin luominen VScode:een

2.3.1 Tehtävä

Seuraava lista täytyy lajitella Quicksort algoritmin avulla:

[7, 6, 10, 5, 9, 8, 3, 4]

2.3.2 Suunnittelu



Kuva 2

2.3.3 Toteutus ja selitys

```
quicksort.py X
quicksort.py > quicksort
1 def quicksort(s):
2     #jos listan pituus on yksi tai pienempi palautetaan lista suoraan
3     if len(s) <= 1:
4         return s
5
6     #asetetaan pivottia (listan ensimmäinen alkio)
7     pivot = s[0]
8
9     #filteröidään vasen puolen alkiot, mikäli ne ovat pivottia pienempi
10    left = list(filter(lambda x: x < pivot, s))
11
12    #filteröidään keskipuolen alkiot, mikäli ne ovat pivottia yhtäsuuret
13    center = [ i for i in s if i == pivot]
14
15    #filteröidään oikean puolen alkiot, mikäli ne ovat pivottia suurempia
16    right = list(filter(lambda x: x > pivot, s))
17
18    #palautetaan järjestelty lista
19    return quicksort(left) + center + quicksort(right)
```

Kuva 3

2.3.4 Debug

Käyn läpi ensimmäisen tason suunnittelua varmistaakseni, että algoritmi toimii oikein.

```
✓ VARIABLES
  ✓ Locals
    (return) <lambda>: False
    > (return) <listcomp>: [7]
    > center: [7]
    > left: [6, 5, 3, 4]
    pivot: 7
    > right: [10, 9, 8]
    > s: [7, 6, 10, 5, 9, 8, 3, 4]
  > Globals
```

Pivot on 7, sillä se on listan ensimmäinen alkio

Left on [6, 5, 3, 4], sillä lista alkiot ovat pivottia pienempia

Right on [10, 9, 8], sillä lista alkiot ovat pivottia suurempia

Center on 7, koska listan i on yhtä suuri valitun pivotin kanssa

2.4 Testaaminen

2.4.1 1000 alkion listalla

```
28 def test_quicksort():
29     #luodaan satunnainen lista
30     s = [random.randint(0, 1000) for _ in range(1000)]
31
32     s_copy = s[:]
33
34     start_time = time.time()
35     #lajitellaan kopioitua listaa Quicksort-algoritilla
36     sorted_list = quicksort(s_copy)
37     end_time = time.time()
38
39     #pika tarkistus, että lista on järjestetty
40     assert sorted_list == sorted(s), "Quicksort-algoritmi ei lajittele listaa oikein!"
41
42     #lopuksi tulostetaan suoritusaika
43     print('Quicksort-algoritmin suoritusaika: ', end_time - start_time, "sekuntia")
44
45     test_quicksort()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● admin@91-154-66-82 seminar_quicksort % python3 quicksort.py
  Quicksort-algoritmin suoritusaika:  0.0013489723205566406 sekuntia
○ admin@91-154-66-82 seminar_quicksort %
```

suoritusaika: 0.0013489723205566406 sekuntia

2.4.2 10000 alkion listalla

```
28 def test_quicksort():
29     #luodaan satunnainen lista
30     s = [random.randint(0, 10000) for _ in range(1000)]
31
32     s_copy = s[:]
33
34     start_time = time.time()
35     #lajitellaan kopioitua listaa Quicksort-algoritilla
36     sorted_list = quicksort(s_copy)
37     end_time = time.time()
38
39     #pika tarkistus, että lista on järjestetty
40     assert sorted_list == sorted(s), "Quicksort-algoritmi ei lajittele listaa oikein!"
41
42     #lopuksi tulostetaan suoritusaika
43     print('Quicksort-algoritmin suoritusaika: ', end_time - start_time, "sekuntia")
44
45     test_quicksort()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● admin@91-154-66-82 seminar_quicksort % python3 quicksort.py
  Quicksort-algoritmin suoritusaika:  0.0013489723205566406 sekuntia
● admin@91-154-66-82 seminar_quicksort % python3 quicksort.py
  Quicksort-algoritmin suoritusaika:  0.001461029052734375 sekuntia
○ admin@91-154-66-82 seminar_quicksort %
```

suoritusaika: 0.001461029052734375 sekuntia

2.4.3 Yhteenveto

Quicksort-algoritmi aikavaatimus on $O(n \log n)$

2.5 Vastaus kysymyksiin

2.5.1 Missä tilanteissa Quicksort-algoritmia käytetään?

Quicksort-algoritmia käytetään tilanteissa, jossa on suuri datamäärä. Algoritmin avulla voi tehokkaasti ja nopeasti lajitella dataa $O(n \log n)$ aikavaatimuksella.

2.5.2 Missä tilanteissa Quicksort-algoritmia vältetään?

Quicksort-algoritmia pyritään välttämään tilanteissa, jossa tietorakenne on valmiiksi lähes järjestetty, sillä algoritmi voi ajaa ikuisen silmukkaan.

2.5.3 Quicksort-Algoritmin hyödyt ja haitat

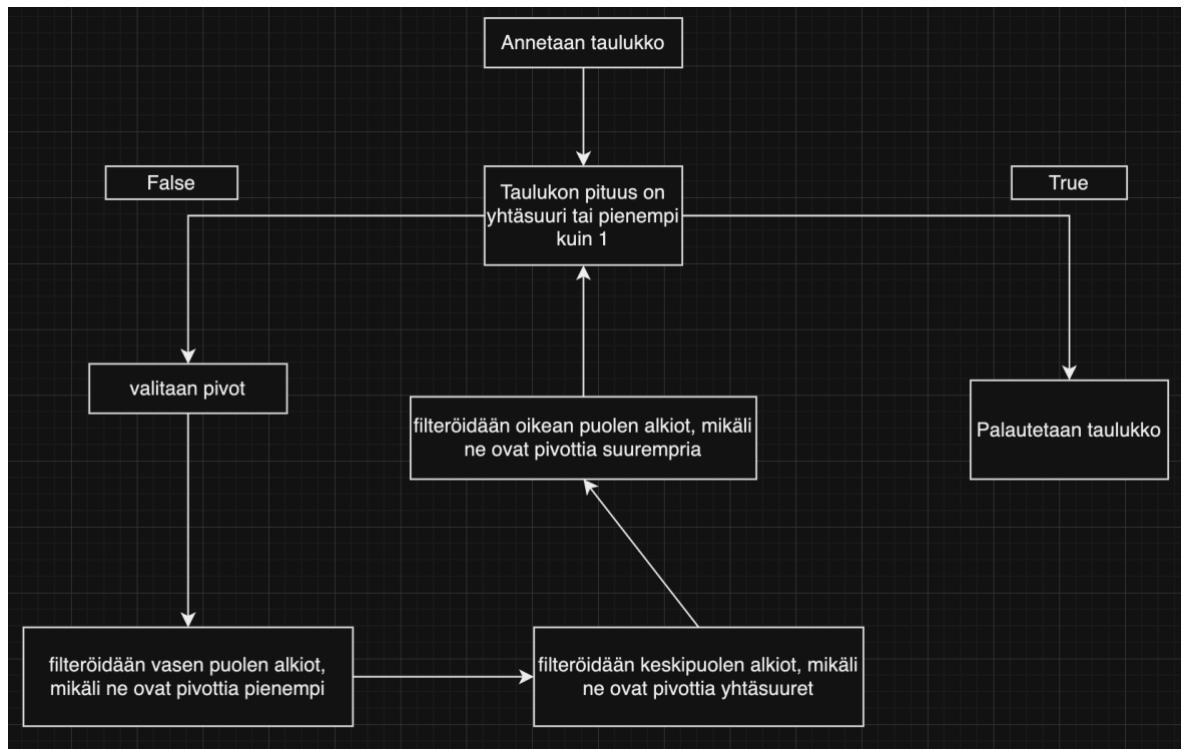
Hyödyt:

1. *Nopeus*
2. *Tehokkuus*
3. *Helppo toteuttaa*

Haitat:

1. *Huono suoritus pahimmassa tapauksessa*
2. *Huono pienille tietomääriille*
3. *Huono melko järjestetyille tietorakenteelle*

3 Algoritmin vuokaavio



Kuva 4 Quicksort vuokaavio

4 Yhteenveto / Tulokset

4.1 Yhteenveto

Oli erinomainen kokemus kirjoittaa tämä seminaarityötä. Tämän työn aikana tutustuin siihen aihepiiriin, joka minua kiinnostaa sekä sen kanssa haluaisin työskennellä tulevaisuudessa.

Tämän seminaarityön avulla saavutin kaikki asetetut tavoitteet ja samalla vastaisin omiin kysymyksiin.

4.2 Tulokset

Quicksort-algoritmi on $O(n \log n)$ Big O notaation mukaan, mikä tarkoittaa että sen suoritusaika kasvaa logaritmisesti syötteen koon kasvaessa.

Kiitos!

Lähdeluettelo

Grokking Algorithms By Aditya Bhargava, Chapter 4. Quicksort: Kuva 1, Big O notation.
Luettu 26.4.2024

Luotu Draw.io avulla: Kuva 2

Koodi on toteutettu VScode avulla: Kuva 3