

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

Algoritmų sudarymas ir analizė

1 Laboratorinis darbas

Atliko:

IF 8/1 grupės stud.

Tomas Odinas

Priėmė

doc. Dalius Makackas

KAUNAS, 2020

TURINYS

[1. Užduotis 2](#_Toc35290788)

[2. Pasirinktos duomenų struktūros realizuotos išorinėje atmintyje struktūrinė diagrama 2](#_Toc35290789)

[Masyvo duomenų struktūros realizavimo išorinėje atmintyje diagrama: 2](#_Toc35290790)

[Sąrašo duomenų struktūros realizavimo išorinėje atmintyje diagrama: 3](#_Toc35290791)

[3. Algoritmo analizė 4](#_Toc35290792)

[Algoritmo įvertinimai literatūroje: 4](#_Toc35290793)

[Quicksort Partition metodo analizė: 4](#_Toc35290794)

[QuickSort metodo analizė 5](#_Toc35290795)

[Blogiausias atvejis: 6](#_Toc35290796)

[Geriausias atvejis: 7](#_Toc35290797)

[4. Atlikti eksperimentai 8](#_Toc35290798)

[Kai elementai saugomi operatyvioje atmintyje: 8](#_Toc35290799)

[Kai elementai saugomi diskinėje atmintyje: 9](#_Toc35290800)

[5. Išvados 10](#_Toc35290801)

[6. Priedas 10](#_Toc35290802)

# 1. Užduotis

Palyginkite rikiavimo algoritmus, kai rikiavimas atliekamas masyve ir sąraše („Linked List“), t. y. galimos tik tai struktūrai būdingos operacijos. Abu šiuos algoritmus realizuokite ir palyginkite dviem atvejais: 1) rikiuojami duomenų elementai saugomi operatyviojoje atmintyje; 2) rikiuojami duomenų elementai visą laiką saugomi išorinėje (diskinėje) atmintyje, o operatyviojoje atmintyje gali būti saugojami tik neindeksuotuose kintamuosiuose (neturi būti masyvo, sąrašo ar jų analogų operatyvinėje atmintyje).Programose turi būti numatyta galimybė išvesti į ekraną nesurikiuoto ar surikiuoto masyvo ar sąrašo fragmentą nuo dėstytoju nurodyto elemento.

Variantas: 5. Rikiavimas „Quick Sort“

# 2. Pasirinktos duomenų struktūros realizuotos išorinėje atmintyje struktūrinė diagrama

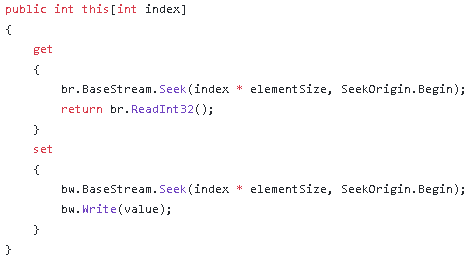
## Masyvo duomenų struktūros realizavimo išorinėje atmintyje diagrama:

Paveikslėlis, kuriame yra ekrano nuotrauka

Automatiškai sugeneruotas aprašymas

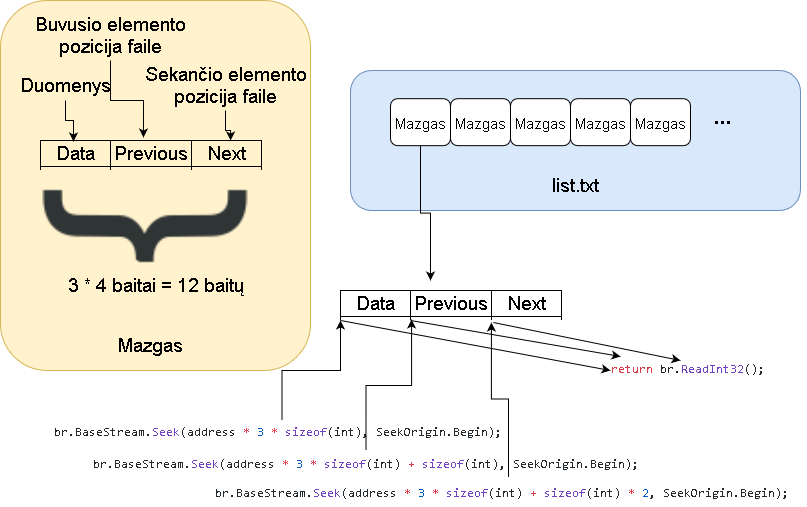
Pav. 1

Programos kodo fragmentas realizuojantis elemento nuskaitymą iš išorinės atminties:



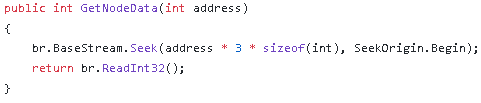
Pav. 2

## Sąrašo duomenų struktūros realizavimo išorinėje atmintyje diagrama:



Pav. 3

Programos kodo fragmentas realizuojantis elemento nuskaitymą iš išorinės atminties:



Pav. 4

# 3. Algoritmo analizė

## Algoritmo įvertinimai literatūroje:

Blogiausiu atveju . Geriausiu ir vidutiniu atveju

Šaltinis: <https://www.bigocheatsheet.com/>

## Quicksort Partition metodo analizė:

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | **private** **static** **int** **Partition**(**int**[] arr, **int** start, **int** end) {  **int** temp;  **int** p = arr[end];  **int** i = start - **1**;  **for** (**int** j = start; j <= end - **1**; j++)  **if** (arr[j] <= p) {  i++;  temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  temp = arr[i + **1**];  arr[i + **1**] = arr[end];  arr[end] = temp;  **return** i + **1**;  } | **Kaina**  c1  c1  c2  c3  c4  c2  c1  c1  c1  c2  c2  c1  c2 | **Kiekis**  1  1  1  end-start+1  end-start  k  k  k  k  1  1  1  1 |

Paaiškinimas: k – kiek kartų patenkinama if salyga (0 < k < end-start).

Partition metodo sudėtingumas:

Atmetus konstantas bendru atveju Partition metodo sudėtingumą galima laikyti **Θ(n)**

## QuickSort metodo analizė

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **public** **static** **void** **QuickSort**(**int**[] arr, **int** start, **int** end)  {  **if** (start < end)  {  **int** pivot = Partition(arr, start, end);  QuickSort(arr, start, pivot - **1**);  QuickSort(arr, pivot + **1**, end);  }  } | **Kaina**  c1  Θ(n)  QuickSort(k)  QuickSort(n-k+1) | **Kiekis**  1  1  1  1 |

Atmetus konstantas bendru atveju QuickSort metodo sudėtingumą galima laikyti:

## Blogiausias atvejis:

Blogiausiu algoritmo veikimo atveju Partition metodas uždavinį suskaldo į n-1 ir 0 elementų turinčius uždavinius.

Paveikslėlis, kuriame yra žinutė, žemėlapis

Automatiškai sugeneruotas aprašymas

Pav. 5

Tokiu atveju QuickSort metodo sudėtingumas yra:

Susumavus kiekvieno rekursinio iškvietimo sudėtingumą (sprendžiant medžio metodu) gauname:

Vertinant asimptotiškai gauname, kad QuickSort metodo sudėtingumas blogiausiu atveju yra

## Geriausias atvejis:

Geriausiu algoritmo veikimo atveju Partition metodas uždavinį suskaldo į ir elementų turinčius uždavinius.

Tokiu atveju QuickSort sudėtingumas yra:

Metodo rekursinius iškvietimus pavaizduojame medžiu:

Paveikslėlis, kuriame yra žinutė, žemėlapis

Automatiškai sugeneruotas aprašymas

Pav. 6

Susumavus kiekvieno rekursinio iškvietimo sudėtingumą (sprendžiant medžio metodu) gauname:

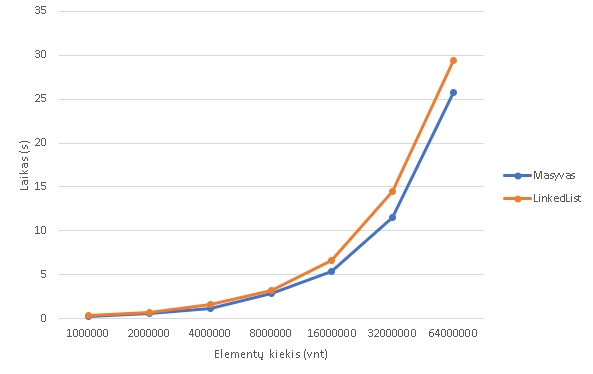
Atmetus konstantas gauname, kad QuickSort metodo sudėtingumas geriausiu atveju yra

# 4. Atlikti eksperimentai

## Kai elementai saugomi operatyvioje atmintyje:

Lentelėje pateikti masyvo ir „„Linked List““ duomenų struktūrų rikiavimo laikai sekundėmis priklausomai nuo duomenų kiekio.

|  |  |  |
| --- | --- | --- |
| Kiekis | Masyvas (sekundės) | „Linked List“ (sekundės) |
| 1000000 | 0,273 | 0,356 |
| 2000000 | 0,572 | 0,753 |
| 4000000 | 1,189 | 1,574 |
| 8000000 | 2,889 | 3,248 |
| 16000000 | 5,418 | 6,65 |
| 32000000 | 11,552 | 14,44 |
| 64000000 | 25,729 | 29,435 |

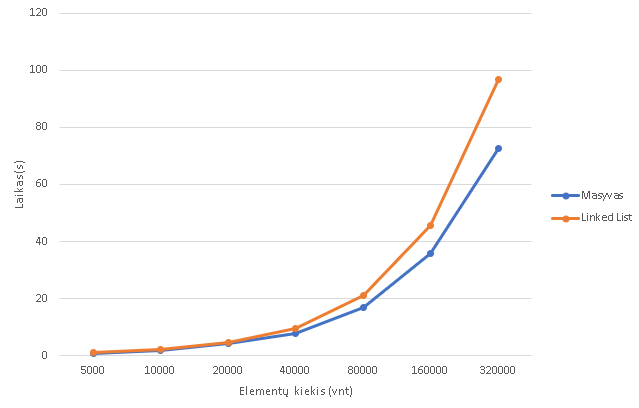


Pav. 7

## Kai elementai saugomi diskinėje atmintyje:

Lentelėje pateikti masyvo ir „„Linked List““ duomenų struktūrų rikiavimo laikai sekundėmis priklausomai nuo duomenų kiekio.

|  |  |  |
| --- | --- | --- |
| Kiekis | Masyvas (sekundės) | „Linked List“ (sekundės) |
| 5000 | 0,821 | 1,13 |
| 10000 | 1,947 | 2,116 |
| 20000 | 4,249 | 4,636 |
| 40000 | 7,94 | 9,579 |
| 80000 | 16,93 | 21,11 |
| 160000 | 35,71 | 45,5 |
| 320000 | 72,457 | 96,631 |



Pav. 8

# 5. Išvados

Elementų saugojimas masyve are „Linked List“ struktūroje rikiavimo greitaveikos iš esmės nekeičia. Saugant elementus „Linked List“ struktūroje padidėja tik konstantų reikšmės norint elementą nuskaityti ar įrašyti. Nors abiejų duomenų struktūrų realizavimas diskinėje atmintyje kreipties laikus žymiai padidina, tačiau bendros algoritmo tendencijos irgi nekeičia.

„Quick Sort“ rikiavimo algoritmas geriausiu ir vidutiniu atvejais asimptotiškai įvertinamas , o blogiausiu atveju .

„Quick Sort“ algoritmas priešingai nei „Merge Sort“ nereikalauja papildomos atminties elementams saugoti. Tai vienas iš jo privalumų.

Nors „Heap Sort“ blogiausiu atveju dirba greičiau nei „Quick Sort“, tačiau santykinai mažoms duomenų imtims naudojamas „Quick Sort“, nes konstantos jo įvertinime yra mažesnės.

Atraminio elemento (angl. „pivot“) pasirinkimui egzistuoja daug strategijų (pirmas elementas, paskutinis elementas, mediana, 3-ijų mediana, atsitiktinis ...), tačiau nei viena iš jų negarantuoja, jog bus absoliučiai išvengta blogiausio algoritmo veikimo atvejo.

# 6. Priedas

Nuoroda į programinio kodo saugyklą: <https://github.com/algoritmu-sudarymas-ir-analize/L1>