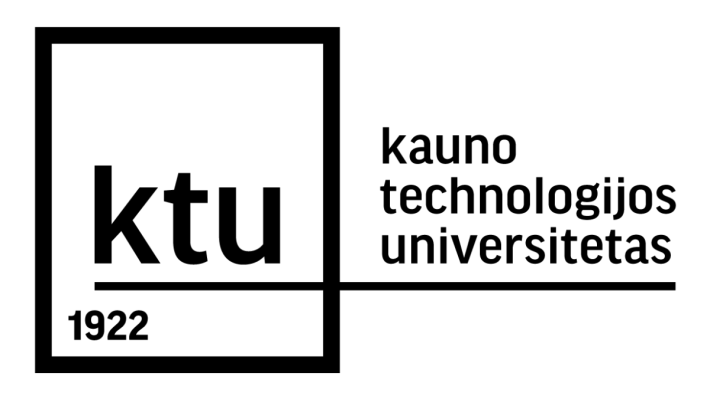
# KAUNO TECHNOLOGIJOS UNIVERSITETAS INFORMATIKOS FAKULTETAS



**ALGORITMŲ SUDARYMAS IR ANALIZĖ (P170B400)**

***Laboratorinis darbas 1***

Atliko:

IFZ-7/1 grupės studentė Liveta Griguolaitė Priėmė:

Lekt. Tadas Kraujalis

KAUNAS 2019

# Užduotis

* 1. Užduoties variantas

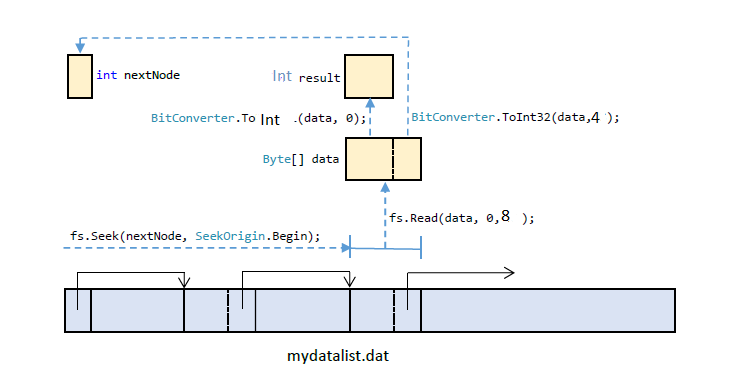


A grupė : Heap sort

B grupė : Quick sort

C grupė : Maišos lentelės su tiesioginiu adresavimu, kai naudojama tiesinė maišos funkcija.

* 1. Failo fragmentas kai dirbama išorinėje aplinkoje



public override int Next()

{

Byte[] data = new Byte[8];

fs.Seek(nextNode, SeekOrigin.Begin);

fs.Read(data, 0,8);

prevNode = currentNode;

currentNode = nextNode;

int result = BitConverter.ToInt32(data, 0);

number = result;

nextNode = BitConverter.ToInt32(data, 4);

cc1 = cc1 + 9;

return result;

}

# Heap sort

## Heap sort yra lyginamoji rūšiavimo technika, pagrįsta dvejetainio krūvos duomenų struktūra. Tai panaši į atrankos rūšį, kur mes pirmą kartą surandame maksimalų elementą ir įterpiame maksimalų elementą gale. Mes pakartojame tą patį procesą likusiam elementui.

Teorinis algoritmo sudėtingumas: O(nLogn).

Šaltiniai: <https://www.geeksforgeeks.org/heap-sort/>

## <https://en.wikipedia.org/wiki/Heapsort>

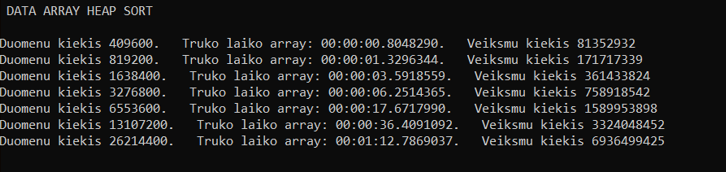
## Heap sort Array vidinėje atmintyje:

|  |  |  |
| --- | --- | --- |
| public static void HeapSortRAMarray(MyDataArray arr, int n)  {  for (int i = n / 2 - 1; i >= 0; i--)  {  heapifyarray(arr, n, i);  c++;  }    for (int i = n - 1; i >= 0; i--)  {    arr.Swap(0, i);c = c + 4;  c++;    heapifyarray(arr, i, 0);  c++;  }  }  public static void heapifyarray(MyDataArray arr, int n, int i)  {    int l = 2 \* i + 1;  c++;  int r = 2 \* i + 2;  c++;  int largest = i;  c++;  if (l < n && arr[l] > arr[largest])  largest = l;  c++;    if (r < n && arr[r] > arr[largest])  largest = r;  c++;    if (largest != i)  {  arr.Swap(i, largest);c = c + 4;  c++;    heapifyarray(arr, n, largest);  c++;  }  } | Kaina | Kartai |

MyDataArray klasės metodai, naudojami rikiavimo metu:

|  |  |  |
| --- | --- | --- |
| public override void Swap(int from, int to)  {  int t;  t = data[from];  data[from] = data[to];  data[to] = t;  } |  |  |

Rezultatai



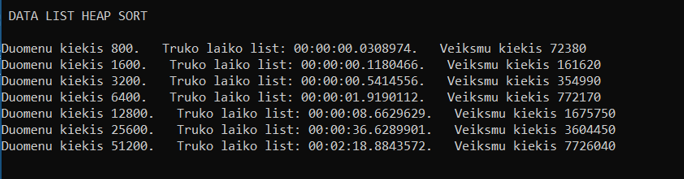
## Heap sort LinkedList vidinėje atmintyje

|  |  |  |
| --- | --- | --- |
| public static void HeapSortRAMlist(MyDataList arr, int n)  {    for (int i = n / 2 - 1; i >= 0; i--)  {  heapifylist(arr, n, i);  c1++;  }    for (int i = n - 1; i >= 0; i--)  {    arr.Swap(arr.GetNode(0), arr.GetNode(i));c1 = c1 + 3;  c1++;    heapifylist(arr, i, 0);  c1++;  }  }  public static void heapifylist(MyDataList arr, int n, int i)  {    int l = 2 \* i + 1;  c1++;  int r = 2 \* i + 2;  c1++;  int largest = i;  c1++;    if (l < n && arr.GetNode(l).data > arr.GetNode(largest).data)  largest = l;  c1++;    if (r < n && arr.GetNode(r).data > arr.GetNode(largest).data)  largest = r;  c1++;    if (largest != i)  {  arr.Swap(arr.GetNode(i), arr.GetNode(largest));c1 = c1+ 3;  c1++;  heapifylist(arr, n, largest);  c1++;  }  } | Kaina | Kartai |
|  |

MyDataList klasės metodai, naudojami rikiavimo metu:

|  |  |  |
| --- | --- | --- |
| public override void Swap(MyLinkedListNode a, MyLinkedListNode b)  {  int c = a.data;  a.data = b.data;  b.data = c;  }  public override MyLinkedListNode GetNode(int n)  {  currentNode = headNode;  for (int i = 0; i < n; i++)  {  currentNode = currentNode.nextNode;  }  return currentNode;  } | Kaina | Kartai |
|  |

Rezultatai



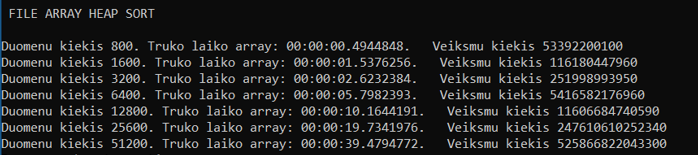
**Heap sort Array išorinėje atmintyje**

|  |  |  |
| --- | --- | --- |
| public static void HeapSortHDDArray(MyFileArray arr, int n)  {    for (int i = n / 2 - 1; i >= 0; i--)  {  heapify(arr, n, i);  c++;  }    for (int i = n - 1; i >= 0; i--)  {    arr.Swap(0, i);c = c + 10;  c++;    heapify(arr, i, 0);  c++;  }  }  public static void heapify(MyFileArray arr, int n, int i)  {  int largest = i;  c++;  int l = 2 \* i + 1;  c++;  int r = 2 \* i + 2;  c++;    if (l < n && arr[l] > arr[largest])  largest = l;  c++;    if (r < n && arr[r] > arr[largest])  largest = r;  c++;    if (largest != i)  {  arr.Swap(i, largest);  c++;    heapify(arr, n, largest);  }  } | Kaina | Kartai |

MyDataArray klasės metodai, naudojami rikiavimo metu:

|  |  |  |
| --- | --- | --- |
| public override void Swap(int a, int b)  {  Byte[] dataa = new Byte[4];  Byte[] datab = new Byte[4];  fs.Seek(4 \* (a), SeekOrigin.Begin);  fs.Read(dataa, 0, 4);  fs.Seek(4 \* (b), SeekOrigin.Begin);  fs.Read(datab, 0, 4);  fs.Seek(4 \* (b), SeekOrigin.Begin);  fs.Write(dataa, 0, 4);  fs.Seek(4 \* (a), SeekOrigin.Begin);  fs.Write(datab, 0, 4);  cc = cc + 10;    } | Kaina | Kartai |
|  |

Rezultatai



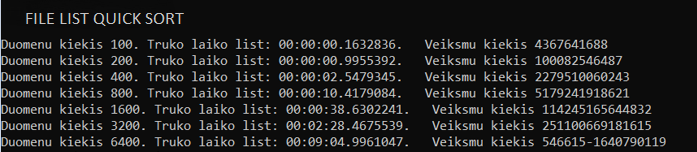
**Heap sort LinkedList išorinėje atmintyje**

|  |  |  |
| --- | --- | --- |
| public static void HeapSortHDDList(MyFileList arr, int n)  {  for (int i = n / 2 - 1; i >= 0; i--)  {  heapifyL(arr, n, i);  c1++;  }    for (int i = n - 1; i >= 0; i--)  {    arr.Swap(arr.GetNode(0), arr.GetNode(i));  c1++;  heapifyL(arr, i, 0);  c1++;  }  }  public static void heapifyL(MyFileList arr, int n, int i)  {  int largest = i;  c1++;  int l = 2 \* i + 1;  c1++;  int r = 2 \* i + 2;  c1++;    if (l < n && arr.GetData(l) > arr.GetData(largest))  largest = l;  c1++;    if (r < n && arr.GetData(r) > arr.GetData(largest))  largest = r;  c1++;    if (largest != i)  {  arr.Swap(arr.GetNode(i), arr.GetNode(largest));  c1++;    heapifyL(arr, n, largest);  c1++;  }  } | Kaina | Kartai |

MyDataList klasės metodai, naudojami rikiavimo metu:

|  |  |  |
| --- | --- | --- |
| public override void Swap(int a, int b)  {  Byte[] dataa = new Byte[4];  Byte[] datab = new Byte[4];  fs.Seek(a, SeekOrigin.Begin);  fs.Read(dataa, 0, 4);  fs.Seek(b, SeekOrigin.Begin);  fs.Read(datab, 0, 4);  fs.Seek(b, SeekOrigin.Begin);  fs.Write(dataa, 0, 4);  fs.Seek(a, SeekOrigin.Begin);  fs.Write(datab, 0, 4);  cc1 = cc1 + 10;  }  public override int GetNode(int i)  {  Head();  cc1++;  for (int j = 0; j < i; j++)  {  Next();  cc1++;  }  cc1++;  return currentNode;  }  public override int GetData(int i)  {  Head();  cc1++;  for (int j = 0; j < i; j++)  {  Next();  cc1++;  }  cc1++;  return number;  }  } | Kaina | Kartai |
|  |

**Rezultatas:**



# Quick sort

Algoritmas veikia tokiu principu:

* Pasirenkamas tam tikras masyvo elementas (angl. pivot); geriausiai kai jis yra mediana, tačiau praktikoje tai neefektyvu
* Perkeliant elementus, masyvas suskirstomas į dvi dalis: pirmoje dalyje yra tik elementai mažesni už pivot, kitame tik didesni
* Naudojantis algoritmu, rekursyviai rikiuojami šie du masyvai
* Kai nebelieka ką rikiuoti, pagrindinis masyvas yra surikiuotas

Teorinis algoritmo sudėtingumas: 

<https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort>

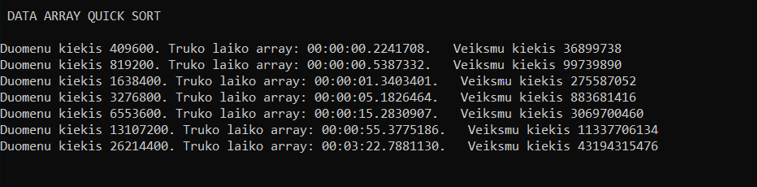
## Quick sort array vidinėje atmintyje

|  |  |  |
| --- | --- | --- |
| public static int partitionarray(MyDataArray arr, int low, int high)  {  int pivot = arr[high];c++;  int i = (low -1);  c++;  for (int j = low; j < high; j++)  {  if (arr[j] <= pivot)  {  i++;  c++;    arr.Swap(i, j);c = c + 4;  c++;  }  }  arr.Swap(i + 1, high);c = c + 4;  c++;  return i+1;  }  public static void sortarray(MyDataArray arr, int low, int high)  {  if (low < high)  {  int pi = partitionarray(arr, low, high);  c++;    sortarray(arr, low, pi-1 );c++;  sortarray(arr, pi+1 , high);c++;  }  } | Kaina | Kartai |

MyDataArray klasės metodai, naudojami rikiavimo metu:

|  |  |  |
| --- | --- | --- |
| public override void Swap(int from, int to)  {  int t;  t = data[from];  data[from] = data[to];  data[to] = t;  } |  |  |

**Rezultatas:**



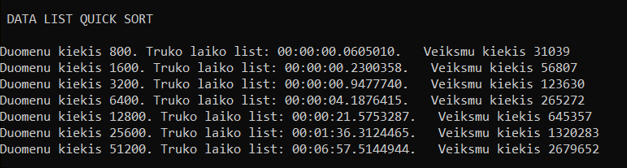
**Quick sort LinkedList vidinėje atmintyje**

|  |  |  |
| --- | --- | --- |
| public static int partitionlist(MyDataList arr, int low, int high)  {  int pivot = arr.GetValue(high);c1++;  int i = (low - 1);  c1++;  for (int j = low; j < high; j++)  {    if (arr.GetValue(j) <= pivot)  {  i++;  c1++;    arr.Swap(arr.GetNode(i ), arr.GetNode(j));c1 = c1 + 3;  c1++;  }  }  arr.Swap(arr.GetNode(i + 1), arr.GetNode(high));c1 = c1 + 3;  c1++;  return i + 1;  }  public static void sortlist(MyDataList arr, int low, int high)  {  if (low < high)  {      int pi = partitionlist(arr, low, high);  c1++;      sortlist(arr, low, pi - 1);c1++;  sortlist(arr, pi + 1, high);c1++;  }  } | Kaina | Kartai |

MyDataList klasės metodai, naudojami rikiavimo metu:

|  |  |  |
| --- | --- | --- |
| public override void Swap(MyLinkedListNode a, MyLinkedListNode b)  {  int c = a.data;  a.data = b.data;  b.data = c;  }  public override MyLinkedListNode GetNode(int n)  {  currentNode = headNode;  for (int i = 0; i < n; i++)  {  currentNode = currentNode.nextNode;  }  return currentNode;  }  public override int GetValue(int ind)  {  int iterator=0;  for (MyLinkedListNode iNode = headNode; iNode != null; iNode = iNode.nextNode)  {  if (iterator == ind)  return iNode.data;  iterator++;  }  return -1;  }  } | Kaina | Kartai |
|  |

**Rezultatas:**



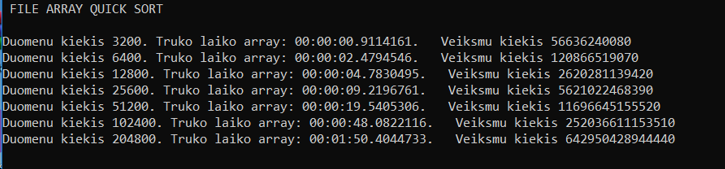
**Quick sort array išorinėje atmintyje**

|  |  |  |
| --- | --- | --- |
| public static int partitionarray(MyFileArray arr, int low, int high)  {  int pivot = arr[high];  c++;  int i = (low - 1);  c++;  for (int j = low; j < high; j++)  {  if (arr[j] <= pivot)  {  i++;  c++;  arr.Swap(i, j);  c++;  }  }  arr.Swap(i + 1, high);  c++;  return i + 1;  }  public static void sortarray(MyFileArray arr, int low, int high)  {  if (low < high)  {  int pi = partitionarray(arr, low, high);  c++;    sortarray(arr, low, pi - 1);c++;  sortarray(arr, pi + 1, high);c++;  }  } | Kaina | Kartai |

MyDataArray klasės metodai, naudojami rikiavimo metu:

|  |  |  |
| --- | --- | --- |
| public override void Swap(int a, int b)  {  Byte[] dataa = new Byte[4];  Byte[] datab = new Byte[4];  fs.Seek(4 \* (a), SeekOrigin.Begin);  fs.Read(dataa, 0, 4);  fs.Seek(4 \* (b), SeekOrigin.Begin);  fs.Read(datab, 0, 4);  fs.Seek(4 \* (b), SeekOrigin.Begin);  fs.Write(dataa, 0, 4);  fs.Seek(4 \* (a), SeekOrigin.Begin);  fs.Write(datab, 0, 4);  cc = cc + 10;  } |  |  |

**Rezultatas**



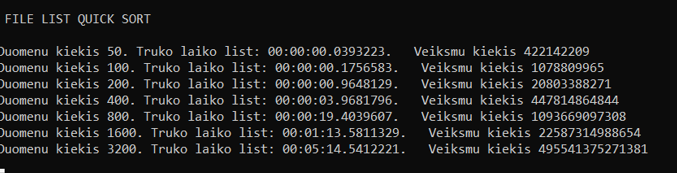
**Quick sort LinkedList išorinėje atmintyje**

|  |  |  |
| --- | --- | --- |
| public static int partitionlist(MyFileList arr, int low, int high)  {  int pivot = arr.GetData(high);c1++;  int i = (low - 1);  for (int j = low; j < high; j++)  {    if (arr.GetData(j) <= pivot)  {  i++;  c1++;    arr.Swap(arr.GetNode(i), arr.GetNode(j));  c1++;  }  }  arr.Swap(arr.GetNode(i+1), arr.GetNode(high));  c1++;  return i + 1;  }  public static void sortlist(MyFileList arr, int low, int high)  {  if (low < high)  {    int pi = partitionlist(arr, low, high);  c1++;    sortlist(arr, low, pi - 1);c1++;  sortlist(arr, pi + 1, high);c1++;  }  } | Kaina | Kartai |

MyDataList klasės metodai, naudojami rikiavimo metu:

|  |  |  |
| --- | --- | --- |
| public override void Swap(int a, int b)  {  Byte[] dataa = new Byte[4];  Byte[] datab = new Byte[4];  fs.Seek(a, SeekOrigin.Begin);  fs.Read(dataa, 0, 4);  fs.Seek(b, SeekOrigin.Begin);  fs.Read(datab, 0, 4);  fs.Seek(b, SeekOrigin.Begin);  fs.Write(dataa, 0, 4);  fs.Seek(a, SeekOrigin.Begin);  fs.Write(datab, 0, 4);  cc1 = cc1 + 10;  }  public override int GetNode(int i)  {  Head();  cc1++;  for (int j = 0; j < i; j++)  {  Next();  cc1++;  }  cc1++;  return currentNode;  }  public override int GetData(int i)  {  Head();  cc1++;  for (int j = 0; j < i; j++)  {  Next();  cc1++;  }  cc1++;  return number;  } | Kaina | Kartai |

**Rezultatas:**



# Išvados

Atlikdami šį labaratorinį darbą išmokome tirti algoritmų sudėtingumą, suskaičiuoti operacijų skaičių, bei atsirinkti, kuris algoritmas mums yra geriausia tinkamas. Rikiavimo algoritmai buvo atliekami dvejose struktūrose: Masyve(array) ir vienos krypties sąraše(LinkedList), bei išorinėje ir vidinėje atmintyje. Iš gautų rezultatų galime pastebėti, gauti algoritmai nėra pritaikyti LinkedList struktūrai, nes algoritmo testavimo laikas yra gerokai didesnis. Tai atspindi ir Heap sort ir Quick sort rikiavimo algoritmai. Taip pat pastebėjau ir įistikinua, kad vidinėje atmintyje duomenys rikiuojami greičiau nei išorinėje atmintyje. Taigi, rikiavimas yra efektyviausias jei rikiuojami duomenys yra masyve vidinėje atmintyje.

# Literatūros sąrašas

# Moodle medžiaga

<https://www.geeksforgeeks.org/quick-sort/>

<https://www.geeksforgeeks.org/heap-sort/>

## <https://en.wikipedia.org/wiki/Heapsort>