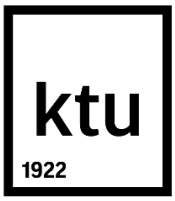


KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

1 laboratorinis darbas

ataskaita

Atliko:

IFF-8/8 gr. studentas

Povilas Dirsė

2020-04-05

Priėmė:

Doc. Pilkauskas Vytautas

KAUNAS 2020

TURINYS

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).

Užduoties variantas: **Radix Sort**

2. DUOMENŲ STRUKTŪROS REALIZUOTOS IŠORINĖJE ATMINTYJE STRUKTŪRINĖ DIAGRAMA

Elemento indeksas	Numeris	Elemento indeksas	Numeris
4 bitai	4 bitai	4 bitai	4 bitai

```
public override double Next()
        {
            Byte[] data = new Byte[8];
            fs.Seek(nextNode, SeekOrigin.Begin);
            fs.Read(data, 0, 8);
            prevNode = currentNode;
            currentNode = nextNode;
            double result = BitConverter.ToDouble(data, 0);
            nextNode = BitConverter.ToInt32(data, 4);
            return result;

        }
```

3. RADIX SORT RIKIAVIMO ALGORITMO ANALIZĖS REZULTATAI

3.1. Algoritmo sudėtingumas

3.1.1 Algoritmo sudėtingumo teorinis įvertinimas

Pirmiausia praeinama pro visus elementus ir iš jų sudaroma lentelė su skaičių pasikartojimais. Po to sudaroma dažnių lentelė ir tuomet elementai dedami, pradedant nuo paskutiniojo elemento, tol kol visi elementai sudedami į jiems reikiamas vietas. Tuomet elementai rušiuojami pagal sekančia skaičiaus poziciją.

Teoriškas algoritmo įvertinimas:$O(d * (n + b)$

Šaltinis: “Radix Sort” geeksforgeeks, www.geeksforgeeks.org/radix-sort/

Apskaičiuotas algoritmo sudėtingumo įvertinimas

	Kaina	Kiekis

`public static void Radix_sort(PlatesContainer items)`
`{`
`  int[] arr = new int[items.Count];`
`  AddIntToArray(items, arr);`
`  For (int exp = 1; exp < Math.Pow(10,9); exp *= 10)`
`  {`
`  Counting_sort(arr, exp);`
`  }`
`}`
	$c1$
	$c2$
	$c3$
	$T(n+m)$
		1
		d
		d
		d

`public static void Counting_sort(int[] A, int exp)`
`{`
`  int[] C = new int[10];`
`  int[] D = new int[A.Length];`
 	
`  int number;`
`  for (int i = 0; i < A.Length; i++)`
`  {`
`  number = (A[i] / exp % 10);`
`  C[number] = C[number] + 1;`
`  }`
 	
`  for (int i = 1; i < 10; i++)`
`  {`
`  C[i] = C[i - 1] + C[i];`
`  }`
 	
`  int index;`
`  for (int i = A.Length-1; i >= 0; i--)`
`  {`
`  number = (A[i] / exp % 10);`
`  index = C[number];`
`  C[number] = index - 1;`
`  D[index-1] = A[i];`
`  }`
 	
`  for (int i = 0; i < A.Length; i++)`
`  {`
`  A[i] = D[i];`
`  }`
`}`
	$c5$
	$c6$
	$c7$
	$c8$
	
	$c9$
	$c10$
	
	$c11$
	
	$c12$
	
	$c13$
	$c14$
	$c15$
	$c16$
	$c17$
	$c18$
	
	$c19$
	
	$c20$
		1
		1
		
		1
		A.Length
		
		A.Length
		
		A.Length+1
		
		M
		
		M-1
		
		1
		A.Length+1
		
		A.Length
		A.Length
		A.Length
		A.Length
		
		A.Length+1
		
		A.Length

Radix Sort:

C1 + C2*D + C3*D + T(N + M)*D = C1 + D(C2 + C3 + T(N + M)) = O(D * (N + M))

Counting Sort:

C5 + C6 + C7 + C8*(A.Length) + C9*(A.Length) + C10*(A.Length+1) + C11*M + C12*(M-1) + C13 + C14*(A.Length+1) + C15*(A.Length) + C16*(A.Length) + C17*(A.Length) + C18*(A.Length) + C19*(A.Length+1) + C20*(A.Length)= C5 + C6+ C7 + A.Length(C8 + C9 + C10 + C14 + C15 + C16 + C17 + C18 + C19 + C20) + C10 + C14 + C19 + M(C11 + C12) - C12 = O(A.Length + M)

Sudėtingumas: $O(d * (m + n))$

A.Length = n = elementu kiekis

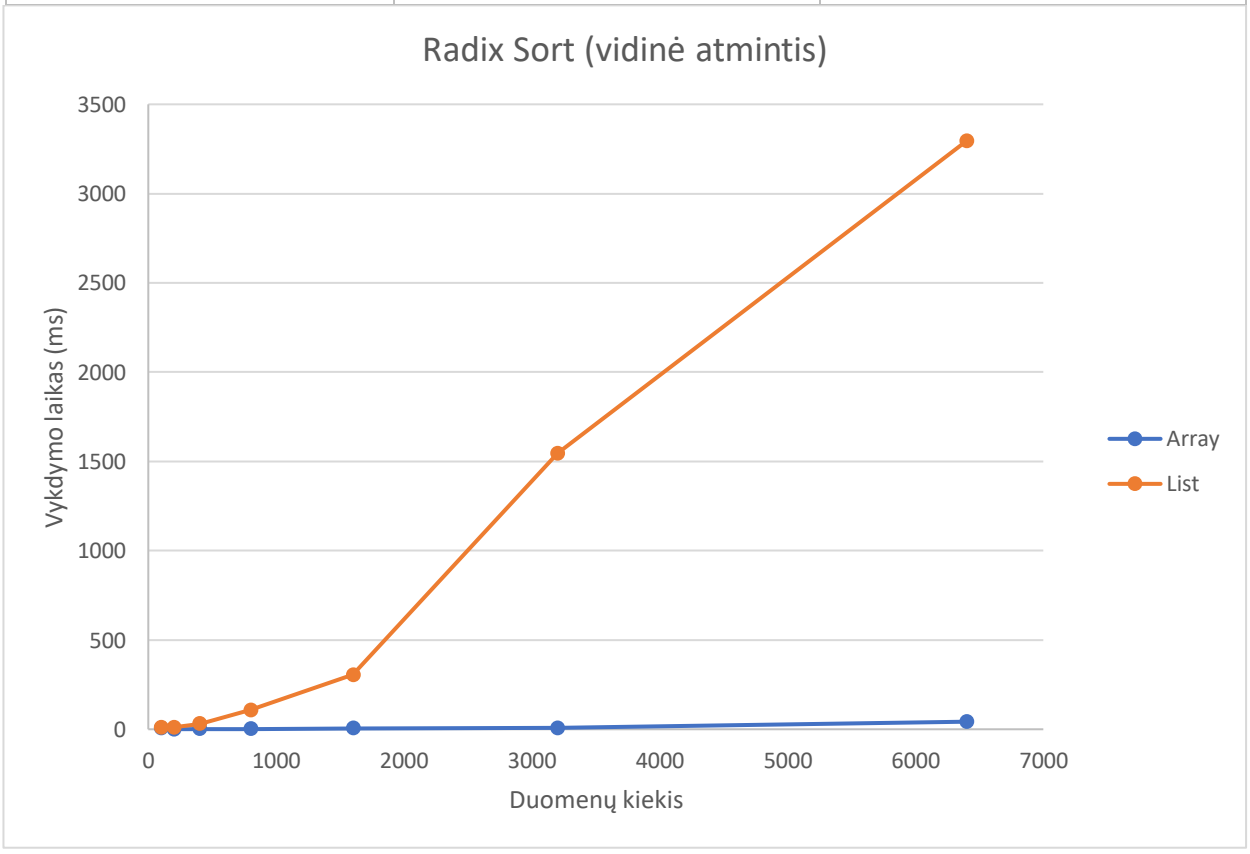
d – dešimčių dalių kiekis

m - skirtingu kombinaciju skaicius

3.2. Algoritmo eksperimentinis tyrimas

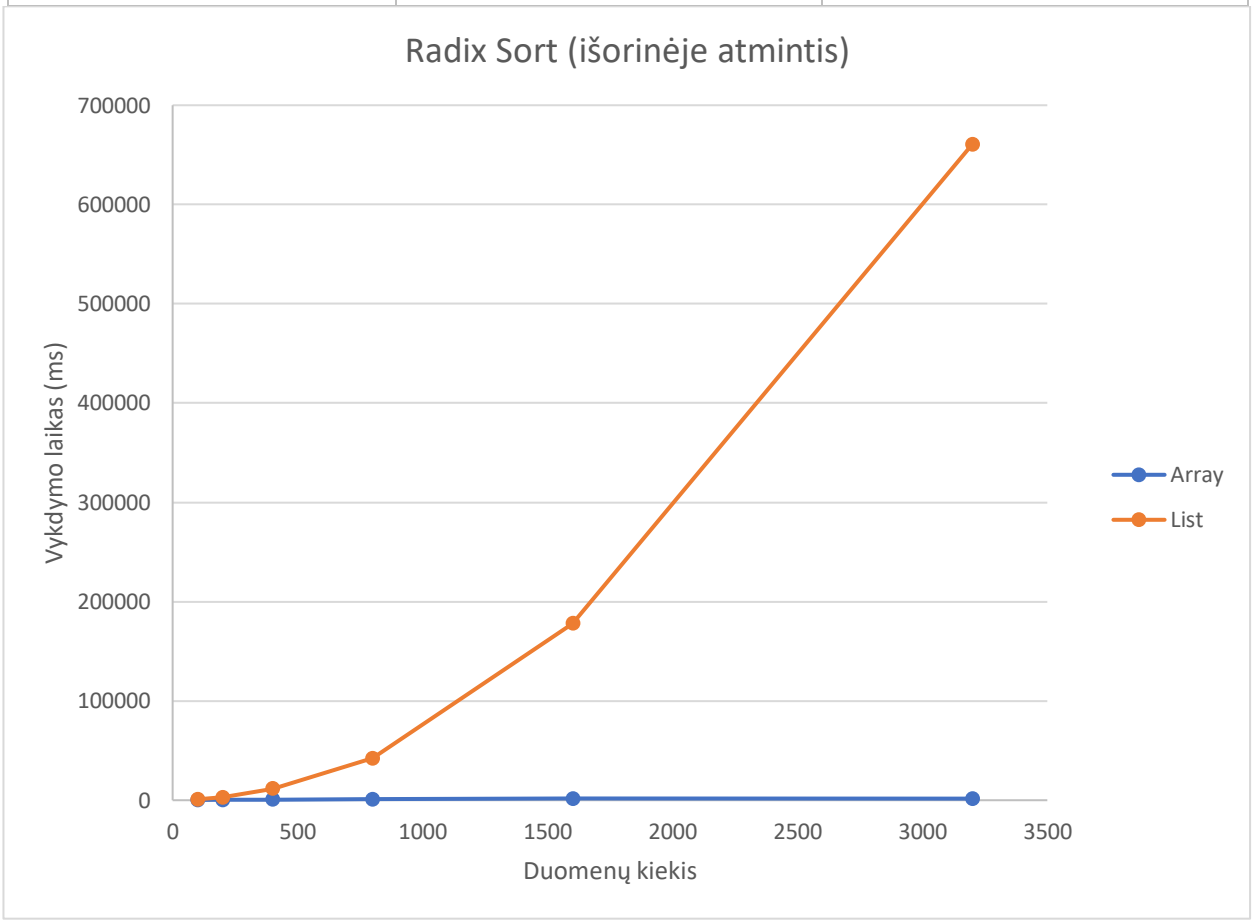
3.2.1 Vidinė atmintis

Duomenys ir rezultatai		
Duomenų kiekis	Vykdymo laikas (ms)	
	Array	List
100	6.3075	8.6321
200	0.5458	10.6291
400	1.0646	29.8899
800	1.5681	107.89
1600	5.3555	304.184
3200	7.7368	1546.06
6400	42.4012	3295.08



3.2.2 Išorinė atmintis

Duomenys ir rezultatai		
Duomenų kiekis	Vykdymo laikas (ms)	
	Array	List
100	236.1754	911.1935
200	339.8461	2952.3533
400	779.7065	11473.6329
800	1048.1825	42334.2499
1600	1518.2541	177971.2561
3200	1593.8198	660705.0344
6400	3037.6448	-



4. IŠVADOS

Šio laboratorinio darbo metu buvo realizuotas Radix sort algoritmas. Šis algoritmas buvo realizuotas tiek vidinėje, tiek išorinėje atmintyje. Radix sort yra vidutinio sudėtingumo algoritmas, nesudėtingai aprašomas, į jo vidų įeina counting sort algoritmas, kuris yra paprastas. Radix sort sudėtingumas yra $O(d(n+m))$, vadinasi jog šis algoritmas yra ganėtinai greitas. Algoritmą pavyko išpildyti tik su masyvu, su sąrašu laiko sudėtingumas gavosi ne toks, koks turėtų būti teoriškai.

5. PRIEDAI

Radix Sort vidinėje atmintyje:

RadixSort.cs:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace RadixSort
{
    class Radix_Sort
    {
        static void Main(string[] args)
        {
            Test_Array_List();
        }

        public static void Test_Array_List()
        {
            int[] amounts = { 100, 200, 400, 800, 1600, 3200, 6400 };
            Console.WriteLine("\n----------------------------Sorting Tests in Internal
Memory-----------------------------");
            Console.WriteLine("-------------------------------------ARRAY RADIX SORT-------
------------------------------");
            foreach (int amount in amounts)
            {
                Test_Array_Radix_Sort(amount);
            }

            Console.WriteLine("-------------------------------------LIST RADIX SORT--------
------------------------------");
            foreach (int amount in amounts)
            {
                Test_List_Radix_Sort(amount);
            }
        }

        public static void Test_Array_Radix_Sort(int amount)
        {
            PlatesContainer container = new PlatesContainer(amount);
            DoContainer(container);

            var watch = System.Diagnostics.Stopwatch.StartNew();
            Radix_sort(container);
            watch.Stop();
            Console.WriteLine(string.Format("{0, -10} {1, -10}", amount,
watch.Elapsed.TotalMilliseconds));
        }

        public static void Test_List_Radix_Sort(int amount)
        {
            LinkedList<NumberPlate> plateList = new LinkedList<NumberPlate>();
            plateList = DoList(plateList, amount);

            var watch = System.Diagnostics.Stopwatch.StartNew();
            Radix_sort(plateList, amount);
            watch.Stop();
            Console.WriteLine(string.Format("{0, -10} {1, -10}", amount,
watch.Elapsed.TotalMilliseconds));
        }

        // --------------- ARRAY METHODS ---------------
```

```
public static void Counting_sort(int[] A, int exp)
{
    int[] C = new int[10];               // Skaiciu poziciju pasikartojimu (masyvas)
    int[] D = new int[A.Length];         // Surusiuotas masyvas

    // Skaiciu daznumas
    int number;
    for (int i = 0; i < A.Length; i++)
    {
        number = (A[i] / exp % 10);
        C[number] = C[number] + 1;
    }

    // Skaiciu kiekis "<=" nurodytam skaiciui
    for (int i = 1; i < 10; i++)
    {
        C[i] = C[i - 1] + C[i];
    }

    // Rusiavimo algoritmas
    int index;
    for (int i = A.Length-1; i >= 0; i--)
    {
        number = (A[i] / exp % 10);
        index = C[number];
        C[number] = index - 1;
        D[index-1] = A[i];
    }

    // Masyvo priskyrimas tolimesniam rusiavimui
    for (int i = 0; i < A.Length; i++)
    {
        A[i] = D[i];
    }
}

public static void Radix_sort(PlatesContainer items)
{
    int[] arr = new int[items.Count];
    AddIntToArray(items, arr);
    for (int exp = 1; exp < Math.Pow(10,9); exp*=10)
    {
        Counting_sort(arr, exp);
    }

    string plates;
    NumberPlate plate;
    for (int i = 0; i < items.Size; i++)
    {
        plate = new NumberPlate();
        plates = BackToPlate(plate, arr[i]);
        items.Set(plate, i);
    }
}

public static void Print(PlatesContainer plates)
{
    for (int i = 0; i < plates.Size; i++)
    {
        Console.WriteLine(plates.Take(i).Letters + " " + plates.Take(i).Number);
    }
}

public static PlatesContainer DoContainer(PlatesContainer plates)
{
```

```
        NumberPlate plate;
        for (int i = 0; i < plates.Size; i++)
        {
            plate = new NumberPlate();
            plate.Generator();
            plates.AddPlate(plate);
        }
        return plates;
    }

    public static void AddIntToArray(PlatesContainer items, int[] array)
    {
        NumberPlate plate;
        for (int i = 0; i < items.Size; i++)
        {
            plate = items.Take(i);
            array[i] = plate.GetPlateCode();
        }
    }

    // --------------- LIST METHODS ---------------

    public static void Counting_sort(LinkedList<int> list, int exp)
    {
        LinkedList<int> D = new LinkedList<int>();
        LinkedList<int>[] lists = new LinkedList<int>[10];
        for (int i = 0; i < lists.Length; i++)
        {
            lists[i] = new LinkedList<int>();
        }

        // Skaiciu daznumas
        int number;
        var currentNode = list.Head;
        while ((currentNode != null) && (currentNode.Data != 0))
        {
            number = (currentNode.Data / exp % 10);
            lists[number].InsertNodeAtEnd(currentNode.Data);
            list.DeleteFirstNode();
            currentNode = currentNode.Next;
        }

        // Sudeda isrikiuotas reiksmes atgal i originalu list'a
        for (int i = 0; i < lists.Length; i++)
        {
            currentNode = lists[i].Head;
            while ((currentNode != null) && (currentNode.Data != 0))
            {
                list.InsertNodeAtEnd(currentNode.Data);
                currentNode = currentNode.Next;
            }
        }
    }

    public static void Radix_sort(LinkedList<NumberPlate> list, int n)
    {
        LinkedList<int> intList = new LinkedList<int>();
        LinkedList<string> sortedList = new LinkedList<string>();
        AddIntToLinkedList(list, intList);
        for (int exp = 1; exp < Math.Pow(10, 9); exp *= 10)
        {
            Counting_sort(intList, exp);
        }

        string plates;
```

```
            NumberPlate plate;

            var currentNode = intList.Head;
            for (int i = 0; i < n; i++)
            {
                plate = new NumberPlate();
                plates = BackToPlate(plate, currentNode.Data);
                sortedList.InsertNodeAtEnd(plates);
                currentNode = currentNode.Next;
            }
        }

        public static LinkedList<NumberPlate> DoList(LinkedList<NumberPlate> plateList, int
n)
        {
            NumberPlate plate;
            for (int i = 0; i < n; i++)
            {
                plate = new NumberPlate();
                plate.Generator();
                plateList.InsertNodeAtEnd(plate);
            }
            return plateList;
        }

        public static void Print(LinkedList<string> sortedList)
        {
            var currentNode = sortedList.Head;
            while (currentNode != null)
            {
                Console.WriteLine("{0}", currentNode.Data);
                currentNode = currentNode.Next;
            }
        }

        public static void AddIntToLinkedList(LinkedList<NumberPlate> plateList,
LinkedList<int> intList)
        {
            NumberPlate number;
            int plateCode;

            var currentNode = plateList.Head;
            while ((currentNode != null) && (currentNode.Data != null))
            {
                number = currentNode.Data;
                plateCode = number.GetPlateCode();
                intList.InsertNodeAtEnd(plateCode);
                currentNode = currentNode.Next;
            }
        }

        // ----------- GENERAL METHODS -------------
        public static string BackToPlate(NumberPlate plate, int number)
        {
            string plateString = number.ToString();
            string firstLetter = plateString.Substring(0, 2);
            string secondLetter = plateString.Substring(2, 2);
            string thirdLetter = plateString.Substring(4, 2);
            string numbers = plateString.Substring(6, 3);

            char one = (char)int.Parse(firstLetter);
            char two = (char)int.Parse(secondLetter);
            char three = (char)int.Parse(thirdLetter);

            string plateLetters = one.ToString() + two.ToString() + three.ToString();
```

```
            string plateNumber = numbers;

            plate.Letters = plateLetters;
            plate.Number = plateNumber;

            string result = plateLetters + " " + numbers;

            return result;
        }
    }
}
```

PlatesContainer.cs:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace RadixSort
{
    class PlatesContainer
    {
        private NumberPlate[] Plates;
        public int Count { get; private set; }
        public int Size { get; private set; }

        public PlatesContainer(int size)
        {
            Plates = new NumberPlate[size];
            Count = 0;
            Size = size;
        }

        public PlatesContainer()
        {

        }

        public void AddPlate(NumberPlate numberPlate)
        {
            Plates[Count++] = numberPlate;
        }

        public NumberPlate Take(int index)
        {
            return Plates[index];
        }

        public void Set(NumberPlate plate, int index)
        {
            if (index < Plates.Length)
            {
                Plates[index] = plate;
            }
        }
    }
}
```

NumberPlate.cs:

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
using System.Text;
```

```
namespace RadixSort
{
    class NumberPlate : IComparable<NumberPlate>, IEquatable<NumberPlate>
    {
        public string Letters { get; set; }
        public string Number { get; set; }
        public int Char { get; set; }

        public NumberPlate(string letters, string number)
        {
            Letters = letters;
            Number = number;
        }

        public NumberPlate()
        {

        }

        public string GetLettersCode(string result)
        {
            result = null;
            for (int i = 0; i < Letters.Length; i++)
            {
                Char = Letters[i];
                result += Char;
            }
            return result;
        }

        public int GetPlateCode()
        {
            string plateString = null;
            int plate = 0;
            string codeString = null;
            codeString = GetLettersCode(codeString);
            plateString = codeString + Number;
            plate = int.Parse(plateString);

            return plate;
        }

        public void Generator()
        {
            string plate = null;
            char random;

            Random rnd = new Random();
            for (int i = 0; i < 6; i++)
            {
                if (i < 2)
                {
                    random = (char)rnd.Next('A', '[');
                    plate += random;
                }
                else if (i == 2)
                {
                    random = (char)rnd.Next('A', '[');
                    plate += random + " ";
                }
                else
                {
                    random = (char)rnd.Next('0', ':');
                    plate += random;
                }
```

```
            }
            string[] values = plate.Split(" ");
            Letters = values[0];
            Number = values[1];
        }

        public int CompareTo([AllowNull] NumberPlate other)
        {
            throw new NotImplementedException();
        }

        public bool Equals([AllowNull] NumberPlate other)
        {
            throw new NotImplementedException();
        }
    }
}
```

LinkedList.cs:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
using System.Diagnostics;
using System.Text;


namespace RadixSort
{
    public class LinkedList<T> : IEnumerable<T> where T : IComparable<T>, IEquatable<T>
    {
        public class Node<T>
        {
            public T Data { get; }
            public Node<T> Next { get; set; }
            public Node(T data)
            {
                Data = data;
            }
        }

        public Node<T> Head { get; set; }

        public void InsertNodeAtEnd(T element)
        {
            Node<T> temp = Head;
            var node = new Node<T>(element);

            if (Head == null)
            {
                Head = node;
            }
            else
            {
                while (temp.Next != null)
                {
                    temp = temp.Next;
                }
                temp.Next = node;
            }
        }

        public void InsertNodeAtFirst(Node<T> node)
        {
```

```
        node.Next = Head;
        Head = node;
    }

    public void InsertNodeAtAGivenPosition(Node<T> node, int indexPosition)
    {
        int counter = 0;
        Node<T> temp = Head;

        while (temp.Next != null)
        {
            if (indexPosition == counter)
            {
                node.Next = temp.Next;
                temp.Next = node;

                return;
            }
            temp = temp.Next;
            counter++;
        }
    }

    public void DeleteFirstNode()
    {
        Head = Head.Next;
    }

    public void DeleteLastNode()
    {
        Node<T> temp = Head;
        Node<T> previousNode = Head;

        while (temp.Next != null)
        {
            previousNode = temp;
            temp = temp.Next;
        }
        previousNode.Next = null;
    }

    public void DeleteNodeFromMiddle(int nodeIndexPosition)
    {
        int counter = 0;
        Node<T> temp = Head;
        Node<T> previousNode = Head;

        while (temp.Next != null)
        {
            if (nodeIndexPosition == counter)
            {
                previousNode.Next = temp.Next;
                return;
            }
            previousNode = temp;
            temp = temp.Next;
            counter++;
        }
    }

    public void Traverse()
    {
        while (Head.Next != null)
        {
            Head = Head.Next;
```

```
                Debug.WriteLine(Head.Data);
            }
        }

        public IEnumerator<T> GetEnumerator()
        {
            throw new NotImplementedException();
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            throw new NotImplementedException();
        }
    }
}
```

Radix Sort išorinėje atmintyje:

Radix_sort.cs:

```
using System;
using System.IO;
using System.Text;

namespace Radix_Sort
{
    class Radix_sort
    {
        static void Main(string[] args)
        {
            int seed = (int)DateTime.Now.Ticks & 0x0000FFFF;
            // Parodo nesurusiuotus ir surusiuotus duomenys array ir list faile
            //Test_File_Array_List(seed);
            Test_Array_List_File();
        }

        public static void Test_Array_List_File()
        {
            int[] amounts = { 100, 200, 400, 800, 1600, 3200, 6400 };
            Console.WriteLine("\n----------------------------Sorting Tests in External
Memory-----------------------------");
            Console.WriteLine("--------------------------------FILE ARRAY RADIX SORT-------
-------------------------");
            Console.WriteLine(string.Format("{0, -10} {1, -10}", "Amount", "Time (ms)"));
            foreach (int amount in amounts)
            {
                Test_Array_Radix_Sort_File(amount);
            }

            Console.WriteLine("--------------------------------FILE LIST RADIX SORT--------
-------------------------");
            Console.WriteLine(string.Format("{0, -10} {1, -10}", "Amount", "Time (ms)"));
            foreach (int amount in amounts)
            {
                Test_List_Radix_Sort_File(amount);
            }
        }

        public static void Test_Array_Radix_Sort_File(int n)
        {
            string filename;
            filename = @"mydataarray.dat";
            MyFileArray myfilearray = new MyFileArray(filename, n);
            using (myfilearray.fs = new FileStream(filename, FileMode.Open,
FileAccess.ReadWrite))
```

```
            {
                var watch = System.Diagnostics.Stopwatch.StartNew();
                Radix_Sort(myfilearray);
                watch.Stop();
                Console.WriteLine(string.Format("{0, -10} {1, -10}", n,
watch.Elapsed.TotalMilliseconds));
            }
        }

        public static void Test_List_Radix_Sort_File(int n)
        {
            string filename;
            filename = @"mydatalist.dat";
            MyFileList myfilelist = new MyFileList(filename, n);
            using (myfilelist.fs = new FileStream(filename, FileMode.Open,
            FileAccess.ReadWrite))
            {
                var watch = System.Diagnostics.Stopwatch.StartNew();
                Radix_Sort(myfilelist, n);
                watch.Stop();
                Console.WriteLine(string.Format("{0, -10} {1, -10}", n,
watch.Elapsed.TotalMilliseconds));
            }
        }


        //public static void Radix_Sort(DataList items)
        //{
        //    for (int exp = 1; exp < Math.Pow(10, 9); exp *= 10)
        //    {
        //        CountingSort(items, exp);
        //    }
        //}

        public static void Radix_Sort(DataArray items)
        {
            for (int exp = 1; exp < Math.Pow(10, 9); exp *= 10)
            {
                Counting_Sort(items, exp);
            }
        }

        public static void Counting_Sort(DataArray items, int exp)
        {
            string resultsFile = "results.txt";
            string countFile = "count.txt";
            int n = items.Length;

            MyFileArray results = new MyFileArray(n, resultsFile);
            MyFileArray count = new MyFileArray(10, countFile);

            using (count.fs = new FileStream(countFile, FileMode.Open,
FileAccess.ReadWrite))
            {
                using (results.fs = new FileStream(resultsFile, FileMode.Open,
FileAccess.ReadWrite))
                {
                    int value;
                    for (int i = 0; i < n; i++)
                    {
                        value = items[i] / exp % 10;
                        count[value] = count[value] + 1;
                    }

                    for (int i = 1; i < 10; i++)
```

```
                    {
                        count[i] = count[i] + count[i - 1];
                    }

                    for (int i = n-1; i >= 0 ; i--)
                    {
                        value = items[i] / exp % 10;
                        results[count[value] - 1] = items[i];
                        count[value] = count[value] - 1;
                    }

                    for (int i = 0; i < n; i++)
                    {
                        items[i] = results[i];
                    }
                }
            }
        }

        public static void Radix_Sort(DataList items, int seed)
        {
            for (int exp = 1; exp < Math.Pow(10, 9); exp *= 10)
            {
                Counting_Sort(items, exp, seed);
            }
        }

        public static void Counting_Sort(DataList items, int exp, int seed)
        {
            string resultsFile = "results.dat";
            string countFile = "count.dat";
            int n = items.Length;

            MyFileList results = new MyFileList(n, resultsFile);
            MyFileList count = new MyFileList(n, countFile);

            using (count.fs = new FileStream(countFile, FileMode.Open,
FileAccess.ReadWrite))
            {
                using (results.fs = new FileStream(resultsFile, FileMode.Open,
FileAccess.ReadWrite))
                {
                    int temp;
                    var current = items.Head();
                    while (current != 0)
                    {
                        current = current / exp % 10;
                        count.Set(current, count.Value(current) + 1);
                        current = items.Next();
                    }

                    for (int i = 1; i < 10; i++)
                    {
                        count.Set(i, count.Value(i) + count.Value(i - 1));
                    }

                    for (int i = n - 1; i >= 0; i--)
                    {
                        temp = items.Value(i) / exp % 10;
                        results.Set(count.Value(temp) - 1, items.Value(i));
                        count.Set(temp, count.Value(temp) - 1);
                    }
                    items.OverWrite(results);
                }
            }
```

```
        }

        public static void Test_File_Array_List(int seed)
        {
            int n = 10;
            string filename;
            filename = @"mydataarraytest.txt";
            //filename = @"mydataarray.dat";
            MyFileArray myfilearray = new MyFileArray(filename, n);
            using (myfilearray.fs = new FileStream(filename, FileMode.Open,
FileAccess.ReadWrite))
            {
                Console.WriteLine("\n FILE ARRAY \n");
                myfilearray.Print(n);
                Radix_Sort(myfilearray);
                myfilearray.Print(n);
            }
            filename = @"mydatalisttest.txt";
            MyFileList myfilelist = new MyFileList(filename, n);
            using (myfilelist.fs = new FileStream(filename, FileMode.Open,
            FileAccess.ReadWrite))
            {
                Console.WriteLine("\n FILE LIST \n");
                myfilelist.Print(n);
                Radix_Sort(myfilelist, seed);
                myfilelist.Print(n);
            }
        }
    }
    abstract class DataArray
    {
        protected int length;
        public int Length { get { return length; } }
        public abstract int this[int index] { get; set; }
        public abstract void Swap(int j, double a, double b);
        public void Print(int n)
        {
            for (int i = 0; i < n; i++)
            {
                string plate = ConvertToString(this[i]);
                Console.Write(" {0:F5} ", plate);
            }
            Console.WriteLine();
        }

        public string ConvertToString(int number)
        {
            string plateString = number.ToString();
            string firstLetter = plateString.Substring(0, 2);
            string secondLetter = plateString.Substring(2, 2);
            string thirdLetter = plateString.Substring(4, 2);
            string numbers = plateString.Substring(6, 3);

            char one = (char)int.Parse(firstLetter);
            char two = (char)int.Parse(secondLetter);
            char three = (char)int.Parse(thirdLetter);

            string plateLetters = one.ToString() + two.ToString() + three.ToString();

            string result = plateLetters + " " + numbers;

            return result;
        }
    }
```

```
    abstract class DataList
    {
        protected int length;
        public int Length { get { return length; } }
        public abstract int Head();
        public abstract int Next();
        public abstract void Swap(double a, double b);
        public abstract void OverWrite(DataList items);
        public abstract void Set(int index, int value);
        public abstract int Value(int index);
        public void Print(int n)
        {
            Console.Write(" {0:F5} ", ConvertToString(Head()));
            for (int i = 1; i < n; i++)
            {
                string plate = ConvertToString(Next());
                Console.Write(" {0:F5} ", plate);
            }
            Console.WriteLine();
        }

        public string ConvertToString(int number)
        {
            string plateString = number.ToString();
            string firstLetter = plateString.Substring(0, 2);
            string secondLetter = plateString.Substring(2, 2);
            string thirdLetter = plateString.Substring(4, 2);
            string numbers = plateString.Substring(6, 3);

            char one = (char)int.Parse(firstLetter);
            char two = (char)int.Parse(secondLetter);
            char three = (char)int.Parse(thirdLetter);

            string plateLetters = one.ToString() + two.ToString() + three.ToString();

            string result = plateLetters + " " + numbers;

            return result;
        }
    }
}
```

Array.cs:

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;

namespace Radix_Sort
{
    class MyFileArray : DataArray
    {
        public MyFileArray(int n, string filename)
        {
            int[] data = new int[n];
            length = n;

            for (int i = 0; i < length; i++)
                data[i] = 0;

            if (File.Exists(filename)) File.Delete(filename);
            try
            {
                using (BinaryWriter writer = new BinaryWriter(File.Open(filename,
FileMode.Create)))
```

```
            {
                for (int j = 0; j < length; j++)
                    writer.Write(data[j]);
            }
        }
        catch (IOException ex)
        {
            Console.WriteLine(ex.ToString());
        }
    }

    public MyFileArray(string filename, int n)
    {
        int[] plates = new int[n];
        length = n;
        for (int i = 0; i < length; i++)
        {
            plates[i] = Generator();
        }
        if (File.Exists(filename)) File.Delete(filename);
        try
        {
            using (BinaryWriter writer = new BinaryWriter(File.Open(filename,
            FileMode.Create)))
            {
                for (int j = 0; j < length; j++)
                    writer.Write(plates[j]);
            }
        }
        catch (IOException ex)
        {
            Console.WriteLine(ex.ToString());
        }
    }

    public int Generator()
    {
        string lettersPlate = null;
        string numberPlate = null;
        char random;
        int result;

        Random rnd = new Random();
        for (int i = 0; i < 6; i++)
        {
            if (i <= 2)
            {
                random = (char)rnd.Next('A', '[');
                lettersPlate += random;
            }
            else
            {
                random = (char)rnd.Next('0', ':');
                numberPlate += random;
            }
        }

        result = ConvertToInt(lettersPlate, numberPlate);

        return result;
    }

    public int ConvertToInt(string plate, string number)
    {
        int numberPlate = 0;
```

```
            string result = null;
            int first = (int)char.Parse(plate.Substring(0, 1));
            int second = (int)char.Parse(plate.Substring(1, 1));
            int third = (int)char.Parse(plate.Substring(2, 1));

            result = first.ToString() + second.ToString() + third.ToString() +
number.ToString();
            numberPlate = int.Parse(result);

            return numberPlate;
        }

        public FileStream fs { get; set; }
        public override int this[int index]
        {
            get
            {
                Byte[] data = new Byte[8];
                fs.Seek(4 * index, SeekOrigin.Begin);
                fs.Read(data, 0, 4);
                int result = BitConverter.ToInt32(data, 0);
                return result;
            }
            set
            {
                Byte[] data = new byte[4];
                BitConverter.GetBytes(value).CopyTo(data, 0);
                BitConverter.ToInt32(data, 0);
                fs.Seek(4 * index, SeekOrigin.Begin);
                fs.Write(data, 0, 4);
            }
        }
        public override void Swap(int j, double a, double b)
        {
            Byte[] data = new Byte[16];
            BitConverter.GetBytes(b).CopyTo(data, 0);
            BitConverter.GetBytes(a).CopyTo(data, 8);
            fs.Seek(8 * (j - 1), SeekOrigin.Begin);
            fs.Write(data, 0, 16);
        }
    }
}
```

List.cs:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace Radix_Sort
{
    class MyFileList : DataList
    {
        int prevNode;
        int currentNode;
        int nextNode;
        public MyFileList(string filename, int n)
        {
            int[] plates = new int[n];
            length = n;
            for (int i = 0; i < length; i++)
            {
                plates[i] = Generator();
            }
```

```
            if (File.Exists(filename)) File.Delete(filename);
            try
            {
                using (BinaryWriter writer = new BinaryWriter(File.Open(filename,
FileMode.Create)))
                {
                    writer.Write(4);
                    for (int j = 0; j < length; j++)
                    {
                        writer.Write(plates[j]);
                        writer.Write((j + 1) * 8 + 4);
                    }
                }
            }
            catch (IOException ex)
            {
                Console.WriteLine(ex.ToString());
            }
        }

        public MyFileList(int n, string filename)
        {
            length = n;
            if (File.Exists(filename))
            {
                File.Delete(filename);
            }
            try
            {
                using (BinaryWriter writer = new BinaryWriter(File.Open(filename,
FileMode.Create)))
                {
                    writer.Write(4);
                    for (int j = 0; j < length; j++)
                    {
                        writer.Write(0);
                        writer.Write((j + 1) * 8 + 4);
                    }
                }
            }
            catch (IOException ex)
            {
                Console.WriteLine(ex.ToString());
            }
        }

        public int Generator()
        {
            string lettersPlate = null;
            string numberPlate = null;
            char random;
            int result;

            Random rnd = new Random();
            for (int i = 0; i < 6; i++)
            {
                if (i <= 2)
                {
                    random = (char)rnd.Next('A', '[');
                    lettersPlate += random;
                }
                else
                {
                    random = (char)rnd.Next('0', ':');
                    numberPlate += random;
```

```
                }
            }

            result = ConvertToInt(lettersPlate, numberPlate);

            return result;
        }

        public int ConvertToInt(string plate, string number)
        {
            int numberPlate = 0;
            string result = null;
            int first = (int)char.Parse(plate.Substring(0, 1));
            int second = (int)char.Parse(plate.Substring(1, 1));
            int third = (int)char.Parse(plate.Substring(2, 1));

            result = first.ToString() + second.ToString() + third.ToString() +
number.ToString();
            numberPlate = int.Parse(result);

            return numberPlate;
        }

        public FileStream fs { get; set; }
        public override int Head()
        {
            Byte[] data = new Byte[8];
            fs.Seek(0, SeekOrigin.Begin);
            fs.Read(data, 0, 4);
            currentNode = BitConverter.ToInt32(data, 0);
            prevNode = -1;
            fs.Seek(currentNode, SeekOrigin.Begin);
            fs.Read(data, 0, 8);
            int result = BitConverter.ToInt32(data, 0);
            nextNode = BitConverter.ToInt32(data, 4);
            return result;
        }
        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);
            nextNode = BitConverter.ToInt32(data, 4);
            return result;

        }
        public override void Swap(double a, double b)
        {
            Byte[] data;
            fs.Seek(prevNode, SeekOrigin.Begin);
            data = BitConverter.GetBytes(a);
            fs.Write(data, 0, 8);
            fs.Seek(currentNode, SeekOrigin.Begin);
            data = BitConverter.GetBytes(b);
            fs.Write(data, 0, 8);

        }
        public override void Set(int index, int value)
        {
            Byte[] data = new Byte[4];
            fs.Seek((8 * index) + 4, SeekOrigin.Begin);
            BitConverter.GetBytes(value).CopyTo(data, 0);
```

```
            fs.Write(data, 0, 4);
        }
        public override int Value(int index)
        {
            int value = Head();

            for (int i = 1; i < Length; i++)
            {
                int next = Next();
                if (i == index)
                {
                    value = next;
                    return value;
                }
            }
            return value;
        }

        public override void OverWrite(DataList items)
        {
            for (int i = 0; i < items.Length; i++)
            {
                Byte[] data = new Byte[4];
                fs.Seek(8 * i + 4, SeekOrigin.Begin);
                BitConverter.GetBytes(items.Value(i)).CopyTo(data, 0);
                fs.Write(data, 0, 4);
            }
        }
    }
}
```