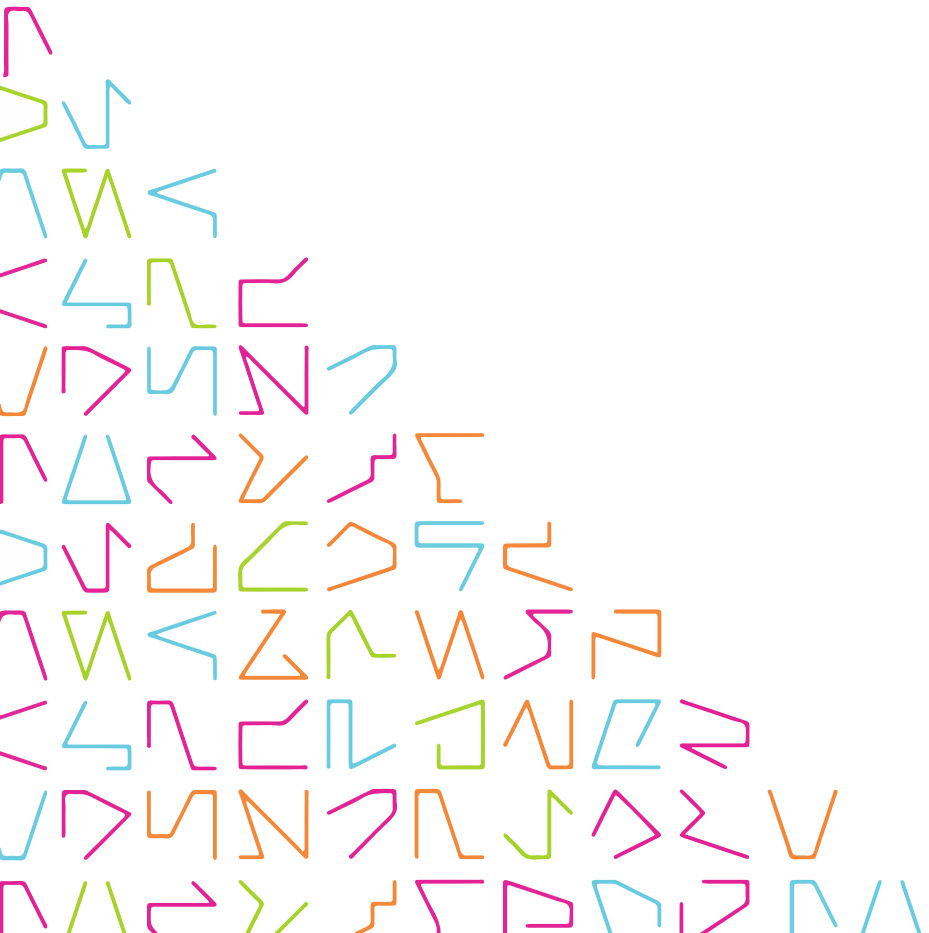
TAMPEREEN
AMMATTIKORKEAKOULU

Tietorakenteet ja algoritmit

Harjoitustyö 3 – Testipeti etsintä- ja/tai lajittelualgoritmeille

Valkoja Iiro



1 Yleistä

1.1 Tavoiteltu pistemäärä

Viittä pistettä oltaisiin hakemassa

1.2 Palauttajan yhteystiedot

Sekä sähköposti että puhelinnumerot ovat saatavissa TAMK:in järjetelmistä, en niitä ala julkaisemaan internetissä.

1.3 Tuntikirjanpito

2017-12-14	10:00-12:00	Perehdytty tehtävänantoon, linked listiin ja algoritmeihin yleisellä tasolla
2017-12-14	12:00-14:00	Koska tehtävänanto ei käskenyt käyttämään linked listejä, päätettiin noudattaa aineiston (tr7, lopun pointers & pitfalls) ohjetta olla keksimättä pyörää uudetsaan ja käyttä vektoreita. Ne myös sopivan esim. binäärihakuun paremmin koska keskikohta löytyy suoraan, ei tarvitse aina kelata koko listaa. Aloitettu mainin ja muutaman apuluokan hahhmoittelu.
2017-12-15	10:00-13:00	Metodit aineiston generoimiseen, paljon aikaa turhautui satunnaislukugeneraattorin kanssa. Selvisi että jos generaattorin alustaa funktiokutsun alussa tai luokan rakentajassa, toiminta on kovin epä varmaa. Lopussa siirretty luokan staattiseksi muuttujat ja alustetaan ennen mainia.
2017-11-15	13:00-14:00	Yhden pisteen toteutuksen kirjoittaminen ja testaus.
2017-11-15	15:00-17:00	Kahden pisteen toteutuksen kirjoittaminen ja testaus. Huomattava määrä aikaa meni cout muotoilujen kertaamiseen ja toteuttamiseen ja tilanteen hoitamiseen jossa käyttäjä haluaa liikaa alkioita tutkittavaan listaan.

2017-11-15	17:00-20:00	Ajastimen ihmettelyä. Tarkkuuden pitäisi riittää mikrosekunteihin asti mutta tulokset ovat pelkkää nollaa. Ongelmana oli nopea suoritus ja mikrosoftin tarjoama onneton minimitick, pidemmissä ajoissa kello tuottaa järkeviä tuloksia
2017-11-16	09:30-10:30	Toistojen määrä ja keskiarvot kakkostehtävään, tulostuksen uudelleenkirjotitaminen
2017-11-16	10:30-13:30	Toteutukset 3 ja 4 pisteelle, ainoa ero kun oli käytetyssä algoritmissa. Tulostus oli myös huomattavasti yksinkertaisempaa koska pyydettiin vain lista ennen ja jälkeen järjestelyn
2017-11-16	14:30-17:00	Viimeisien algoritmien kirjoittamista, statistiikan keräämisen lisäys kaikkiin (operaattoreiden ylikirjoittaminen kuulosti niin huonolta ajatukselta luettavuuden kannalta että päätin hoittaa homman omalla statistiikkaluokalla). Tehtävänannon läpikäyntiä uudelleen, mitä missäkin kohtaa nyt ihan oikeasti pyydettiin.
2017-11-16	17:00-20:00	Yleistä muotoilua, siistimistä, tulostinfunktioita ja utility luokalle metodit stringien pituuksien vertailuun ja lukujen muuttamiseksi strigeiksi (MinGW bugi, jostain syystä c++11 toString metodi on jäänyt jonnekkin)
2017-11-17	10:30-12:30	Testailua, viilaamista, tulostuksen uudelleenkirjoitus kohtiin 2 ja 5
2017-11-17	12:30-14:00	Korjataan visual studion warningit pois, pääasiassa tuputtamalla tyyppimuunnoksia joka väliin ja käyttämällä unsigned eikä int
2017-11-17	15:00-16:00	Tämä tiedosto.

2 Ohjelman kääntäminen & ajaminen

Ohjelmaa on käännetty MinGW mukana tulevalla g++ (GCC) 5.3.0 kääntäjällä. Asenna kääntäjä (windows-ympäristössä se kannattaa myös lisätä PATH:iin) ja suorita:

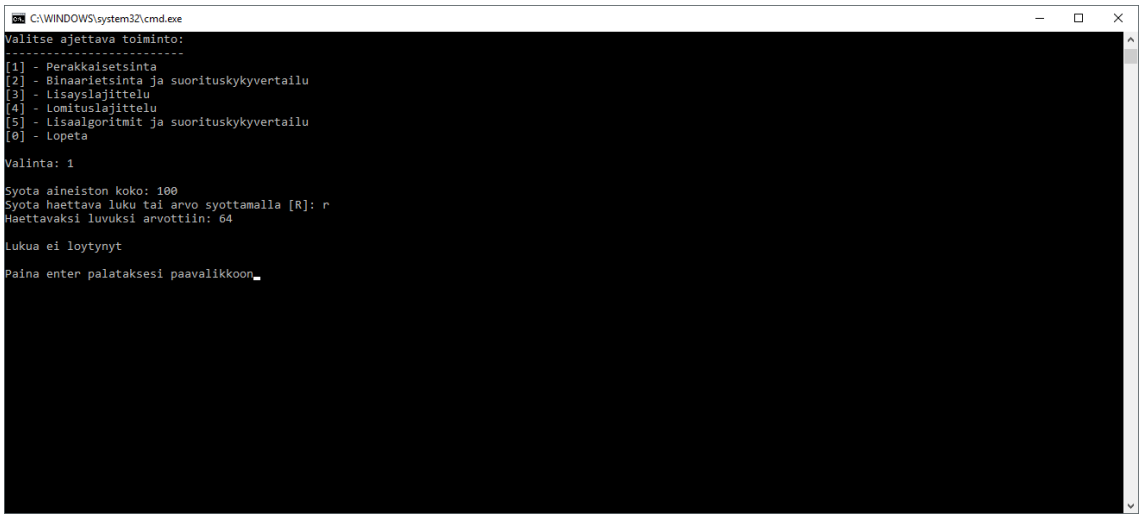
```
g++ -std=c++0x -o .\H3.exe *.cpp
```

Jossa:	
g++	kutsuu kääntäjää
-std=c++0x	kertoo että käytetään C++11 standardia (ISO/IEC 14882:2011)
-o .\H3.exe	käännetyn tiedoston nimi
*.cpp	sisällytetään kaikki cpp tiedostot työkansiosta

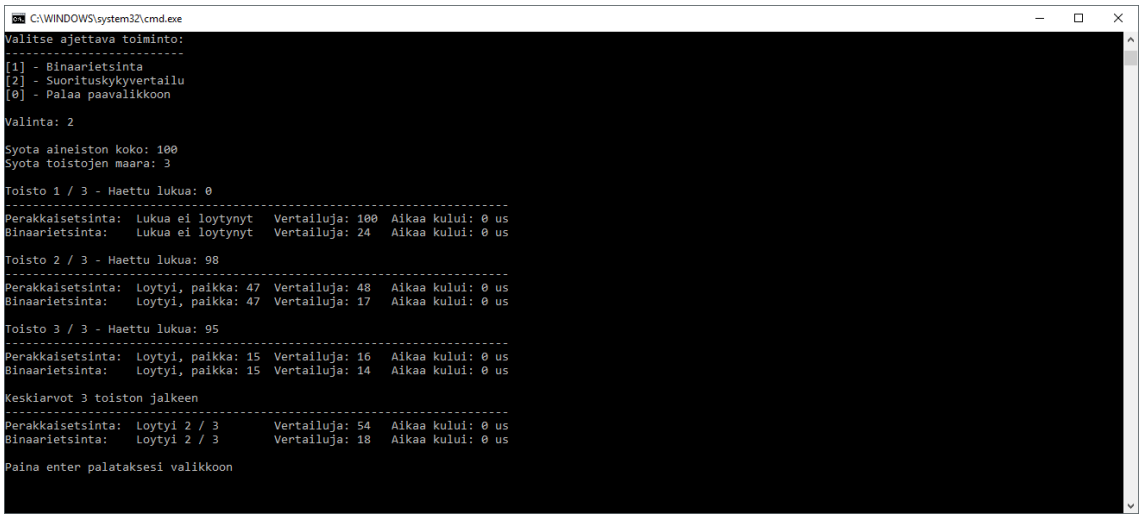
Tämän jälkeen kansiosta pitäisi löytyä H3.exe ajettavaksi.

3 Testitapaukset

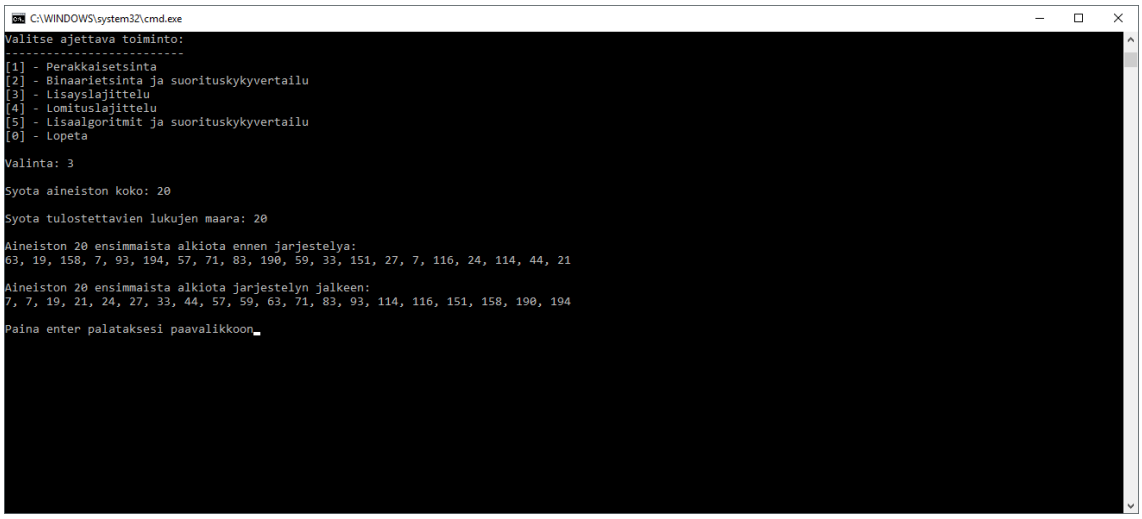
3.1 Yhden pisteen työ



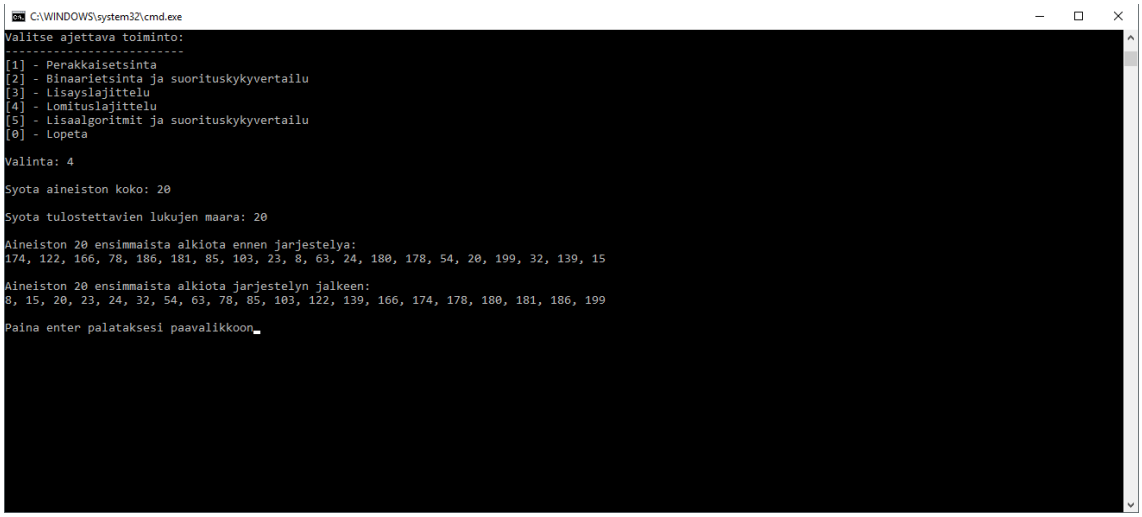
3.2 Kahden pisteen työ



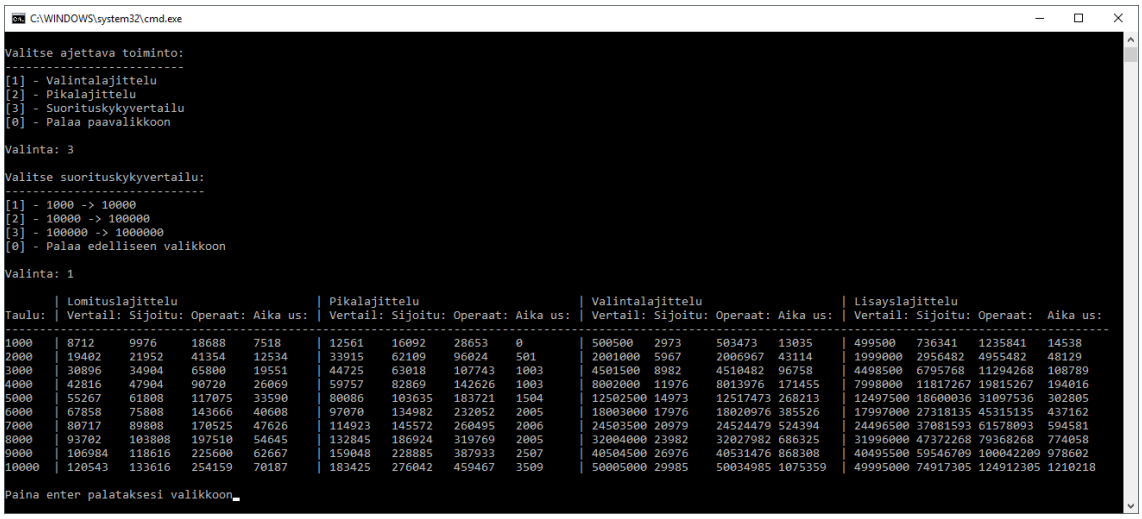
3.3 Kolmen pisteen työ



3.4 Neljän pisteen työ

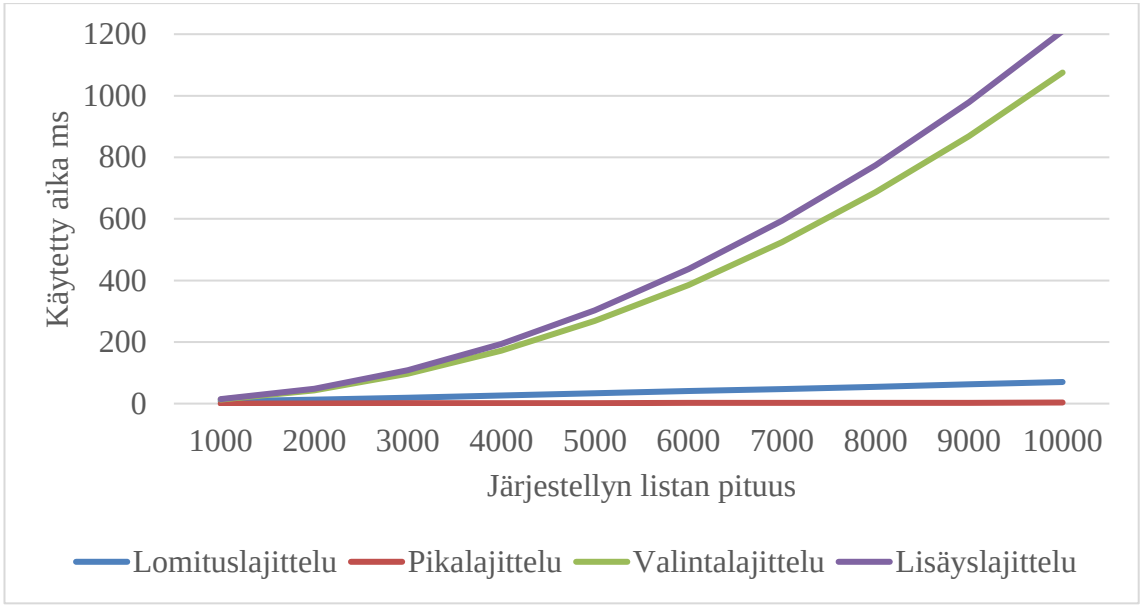


3.5 Viiden pisteen työ



Taulu:	Lomituslajittelu Vertail:	Sijoitu:	Operaat:	Aika us:	Pikalajittelu Vertail:	Sijoitu:	Operaat:	Aika us:	Valintalajittelu Vertail:	Sijoitu:	Operaat:	Aika us:	Lisayslajittelu Vertail:	Sijoitu:	Operaat:	Aika us:
1000	8712	9976	18688	7518	12561	16092	28653	0	500500	2973	503473	13035	499500	736341	1235841	14538
2000	19402	21952	41354	12534	33915	62109	96024	501	2001000	5967	2006967	43114	1999000	2956482	4955482	48129
3000	30896	34904	65800	19551	44725	63018	107743	1003	4501500	8982	4510482	96758	4498500	6795768	11294268	108789
4000	42816	47904	90720	26069	59757	82869	142626	1003	8002000	11976	8013976	171455	7998000	11817267	19815267	194016
5000	55267	61808	117075	33590	80086	103635	183721	1504	12502500	14973	12517473	268213	12497500	18600036	31097536	302805
6000	67858	75808	143666	40608	97070	134982	232052	2005	18003000	17976	18020976	385526	17997000	27318135	45315135	437162
7000	80717	89808	170525	47626	114923	145572	260495	2006	24503500	20979	24524479	524394	24496500	37081593	61578093	594581
8000	93702	103808	197510	54645	132845	186924	319769	2005	32004000	23982	32027982	686325	31996000	47372268	79368268	774058
9000	106984	118616	225600	62667	159048	228885	387933	2507	40504500	26976	40531476	868308	40495500	59546709	100042209	978602
10000	120543	133616	254159	70187	183425	276042	459467	3509	50005000	29985	50034985	1075359	49995000	74917305	124912305	1210218

	Lomituslajittelu	Pikalajittelu	Valintalajittelu	Lisäyslajittelu
1000	7.518	0.000	13.035	14.538
2000	12.534	0.501	43.114	48.129
3000	19.551	1.003	96.758	108.789
4000	26.069	1.003	171.455	194.016
5000	33.590	1.504	268.213	302.805
6000	40.608	2.005	385.526	437.162
7000	47.626	2.006	524.394	594.581
8000	54.645	2.005	686.325	774.058
9000	62.667	2.507	868.308	978.602
10000	70.187	3.509	1075.359	1210.218



4 Suunnitteluratkaisut lyhyesti

4.1 Yhden pisteen työ

Pyydetään käyttäjältä tiedot, luodaan lista, koitetaan muutta lista halutun pituiseksi. Tämä heittää bad_alloc virheen jos listan koon muuttaminen ei onnistu, jolloin pyydetään käyttäjältä lyhyempää pituutta. Hakutehtävissä lista täytettiin parittomilla luvuilla joiden luominen ei kovin monimutkaista ole. Kuten ei myöskään peräkkäisetsintä, loopataan lista lävitse ja palautetaan avain jos haettava arvo löytyy.

4.2 Kahden pisteen työ

Algoritmi hakee puolivälin, tarkistaa onko se haettu tulos, jos ei niin onko haettu suurempi vai pienempi, lähettää itselleen sen puolikkaan listasta jolla haettava luku näyttäisi olevan. Toistaa kunnes luku löytyy tai lista loppuu.

Syötteet muuten samalla tavalla kuin ensimmäisessä, mutta suorituskykyvertailussa kiinnostaa myös toistojen määrä. Sama tulostuksessa, pidetään mielessä sekä kierroksen tiedot että kaikkien kierroksien keskiarvot, jotka tulostetaan lopussa.

4.3 Kolmen pisteen työ

Järjestelyssä aineisto muuttuu satunnaiseksi. Käytetään 10x listan pituutta rajana, paitsi jos se menee INT_MAX yli, jolloin käytetään itse pituutta. Arvotaan luvut utility luokan staattista generaattoria hyväksi käyttäen.

Käyttäjälle näkyvä puoli on enemmän tai vähemmän vastaava kuin ennenkin. Haettavaa luku on vaihtunut tulostettavien alkioiden määrään.

Lisäyslajittelu ei kovasti selittämistä kaipaa, aloitetaan toisesta alkiosta ja tarkistetaan onko sen vasemmalla puolella oleva pienempi. Vaihdetaan päittäin jos on. Toistetaan kaikille alkioille.

4.4 Neljän pisteen työ

Muuten sama kuin kolmen pisteen, algoritmi vaihtui lomituslajitteluksi. Lista jaetaan aina puoliksi kunnes jokainen alkiot on oma, yhden pituinen lista. Sitten tullaan takaisinpäin ja yhdistetään aina puolikkaat yhteen, verraten alkioita niin että jokaisen osalistan sisällä alkiot ovat järjestyksessä. Lopulta päädytään takaisin ylös ja palautetaan viimeinen lista, jossa on kaikki alkiot.

4.5 Viiden pisteen työ

Kaksi uutta algoritmia ja valikko näiden ja suorituskykyvertailun valitsemiseksi. Vertailu ei sinänsä ole monimutkaista, kaikki algoritmit ja niiden tilastointi on tässä kohtaa jo valmiina, ajetaan vaan samalla aineistolla jokainen algoritmi läpi ja otetaan statistiikkaoliosta luvut talteen jokaisen ajon jälkeen. Listan tulostaminen oli hieman monimutkaisempaa.. Ensin käydään kaikkien rivien kaikki sarakkeet lävitse, selvitetään kunkin sarakkeen suurin leveysvaatimus ja käytetään tätä tulostuksessa jokaisen sarakkeen leveytenä.

5 Lähdetekstitiedostot

```
/*
* algorithm.h
* ===========
*/

#pragma once

int sequentialSearch(Statistic &stat, std::vector<int> &list, int target);
int binarySearch(Statistic &stat, std::vector<int> &list, int target, int start,
int end);

void insertionSort(Statistic &stat, std::vector<int> &list);
void selectionSort(Statistic &stat, std::vector<int> &list);

std::vector<int> mergeSort(Statistic &stat, std::vector<int> list);
std::vector<int>  mergeSortCombine(Statistic  &stat,  std::vector<int>  left,
std::vector<int> right);

void quickSort(Statistic &stat, std::vector<int> &list, int lo, int hi);
int quickSortPart(Statistic &stat, std::vector<int> &list, int lo, int hi);

/*
* main.h
* ======
*/

#pragma once

#include <chrono>
#include <iomanip>
#include <iostream>
#include <limits.h>
#include <random>
#include <sstream>
#include <string>
#include <vector>

#include "utilities.h"
#include "statistic.h"
#include "algorithm.h"

void feature_1();
void feature_2();
void feature_2_Find();
void feature_2_Compare();
void feature_2_Print(std::vector<std::vector<std::string>> output);
void feature_3();
void feature_4();
void feature_5();
void feature_5_Sort(int algorithm);
void feature_5_Compare();
void feature_5_Print(std::vector<std::vector<std::string>> output);
```

```
/*
* statistic.h
* ===========
*/

#pragma once

class Statistic
{
    public:
        Statistic();

        void reset();
        void addComparisons(int addition);
        void addAssignments(int addition);
        void timerStart();
        void timerStop();

        void swap(int &first, int &second);
        bool isEqual(int first, int second);
        bool isGreater(int first, int second);

        int getComparisons();
        int getAssignments();
        int getOperations();
        int getTimeElapsed();

    private:
        int comparisons;
        int assignments;
        std::chrono::high_resolution_clock::time_point startTime;
        std::chrono::high_resolution_clock::time_point stopTime;
};

/*
* utilities.h
* ===========
*/

#pragma once

class Utilities
{
    public:
        static std::default_random_engine generator;

        int askLength(std::vector<int> &list);
        int askTarget(int randMax);
        int askPrints(int printsMax);
        int randValue(int randMax);

        int accumulate(std::vector<int> &list, int start, int end);
        int widerStr(std::string str1, std::string str2);
        std::string toString(int number);

        void populateWithOdds(std::vector<int> &list, int size);
        void populateWithRand(std::vector<int> &list, int size);
        void printList(std::vector<int> &list, int size);
};
```

```
/*
* algorithm.cpp
* =============
*/

#include "main.h"

int sequentialSearch(Statistic &stat, std::vector<int> &list, int target)
{
    for (unsigned i = 0; i < list.size(); i++)
    {
        if (stat.isEqual(list.at(i), target))
        {
            return i;
        }
    }

    return -1;
}

int binarySearch(Statistic &stat, std::vector<int> &list, int target, int start,
int end)
{
    if (stat.isEqual(end - start, 1))
    {
        if (stat.isEqual(target, list.at(start)))
            return start;

        if (stat.isEqual(target, list.at(end)))
            return end;

        return -1;
    }

    int middle = start + (int) std::floor((end - start) / 2);

    if (stat.isEqual(target, list.at(middle)))
    {
        return middle;
    }

    if (stat.isGreater(target, list.at(middle)))
    {
        return binarySearch(stat, list, target, middle, end);
    }
    else
    {
        return binarySearch(stat, list, target, start, middle);
    }
}

void insertionSort(Statistic &stat, std::vector<int> &list)
{
    for (unsigned i = 0; i < list.size(); i++)
    {
        for (int j = i; j > 0; j--)
        {
            if (stat.isGreater(list.at(j - 1), list.at(j)))
            {
                stat.swap(list[j-1], list[j]);
            }
        }
    }
}
```

```
void selectionSort(Statistic &stat, std::vector<int> &list)
{
    int min;

    for (unsigned i = 0; i < list.size(); i++)
    {
        min = i;

        for (unsigned j = i + 1; j < list.size(); j++)
        {
            if (stat.isGreater(list.at(min), list.at(j)))
            {
                min = j;
            }
        }

        if (!stat.isEqual(min, i))
        {
            stat.swap(list[i], list[min]);
        }
    }
}

std::vector<int> mergeSort(Statistic &stat, std::vector<int> list)
{
    if (list.size() == 1)
    {
        return list;
    }

    int middle = (int) std::floor(list.size() / 2);
    std::vector<int> left(list.begin(), list.begin() + middle);
    std::vector<int> right(list.begin() + middle, list.end());

    left = mergeSort(stat, left);
    right = mergeSort(stat, right);

    return mergeSortCombine(stat, left, right);
}

std::vector<int>  mergeSortCombine(Statistic  &stat,  std::vector<int>  left,
std::vector<int> right)
{
    std::vector<int> result;

    while (!left.empty() && !right.empty())
    {
        if (stat.isGreater(right.front(), left.front()))
        {
            stat.addAssignments(1);
            result.push_back(left.front());
            left.erase(left.begin());
        }
        else
        {
            stat.addAssignments(1);
            result.push_back(right.front());
            right.erase(right.begin());
        }
    }

    while (!left.empty())
    {
        stat.addAssignments(1);
        result.push_back(left.front());
        left.erase(left.begin());
    }
```

```
    while (!right.empty())
    {
        stat.addAssignments(1);
        result.push_back(right.front());
        right.erase(right.begin());
    }

    return result;
}

void quickSort(Statistic &stat, std::vector<int> &list, int lo, int hi)
{
    if (stat.isGreater(hi, lo))
    {
        int part = quickSortPart(stat, list, lo, hi);

        quickSort(stat, list, lo, part - 1);
        quickSort(stat, list, part + 1, hi);
    }
}

int quickSortPart(Statistic &stat, std::vector<int> &list, int lo, int hi)
{
    int i = lo - 1;
    int pivot = list.at(hi);

    for (int j = lo; j < hi; j++)
    {
        if (stat.isGreater(pivot, list.at(j)))
        {
            i += 1;
            stat.swap(list[i], list[j]);
        }
    }

    if (stat.isGreater(list.at(i + 1), list.at(hi)))
    {
        stat.swap(list[hi], list[i + 1]);
    }

    return i + 1;
}

/*
* feature_1.cpp
* ======
*/

#include "main.h"

void feature_1()
{
    int listIndex = -1;
    int listLength = 0;
    int listTarget = 0;

    Utilities util;
    Statistic stat;
    std::vector<int> listVector;

    // Listan pituus ja haettava avain
    listLength = util.askLength(listVector);
    listTarget = util.askTarget(listLength);

    // Luodaan aineisto, käydään luvut läpi
    util.populateWithOdds(listVector, listLength);
    listIndex = sequentialSearch(stat, listVector, listTarget);
```

```
    // Löytyikö haettava luku
    if (listIndex != -1)
    {
        std::cout << std::endl;
        std::cout << "Luku loytyi listan paikalta " << listIndex << std::endl;
    }
    else
    {
        std::cout << std::endl;
        std::cout << "Lukua ei loytynyt" << std::endl;
    }

    std::cout << std::endl;
    std::cout << "Paina enter palataksesi paavalikkoon";
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

/*
* feature_2.cpp
* ======
*/

#include "main.h"

void feature_2()
{
    int selection = -1;
    std::string inputString;
    std::stringstream inputStream;

    while (selection != 0)
    {
        std::cout << std::endl;
        std::cout << "Valitse ajettava toiminto:" << std::endl;
        std::cout << "--------------------------" << std::endl;
        std::cout << "[1] - Binaarietsinta" << std::endl;
        std::cout << "[2] - Suorituskykyvertailu" << std::endl;
        std::cout << "[0] - Palaa paavalikkoon" << std::endl;

        while (true)
        {
            std::cout << std::endl << "Valinta: ";
            std::getline(std::cin, inputString);

            inputStream.clear();
            inputStream.str(inputString);

            if (!(inputStream >> selection) || selection < 0 || selection > 2)
            {
                std::cout << "Virheellinen valinta, ole hyva ja valitse
numeroista 0-2" << std::endl;
            }
            else
            {
                break;
            }
        }

        switch (selection)
        {
            case 1:
                feature_2_Find();
                break;
            case 2:
                feature_2_Compare();
                break;
        }
    }
}
```

```
void feature_2_Find()
{
    int listIndex  = 0;
    int listLength = 0;
    int listTarget = 0;

    Utilities util;
    Statistic stat;
    std::vector<int> listVector;

    // Listan pituus, haettava avain, aineiston luonti
    listLength = util.askLength(listVector);
    listTarget = util.askTarget(listLength);
    util.populateWithOdds(listVector, listLength);

    // Käydään luvut läpi
    stat.timerStart();
    listIndex = binarySearch(stat, listVector, listTarget, 0, listLength - 1);
    stat.timerStop();

    // Lopputuloseksen tulostus
    if (listIndex != -1)
    {
        std::cout << std::endl;
        std::cout << "Luku loytyi listan paikalta " << util.toString(listIndex)
<< std::endl;
    }
    else
    {
        std::cout << std::endl;
        std::cout << "Lukua ei loytynyt" << std::endl;
    }

    std::cout << "Vertailuja:   " << util.toString(stat.getComparisons()) <<
std::endl;
    std::cout << "Aikaa kului: " << util.toString(stat.getTimeElapsed()) << "
us" << std::endl;

    std::cout << std::endl;
    std::cout << "Paina enter palataksesi valikkoon";
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

void feature_2_Compare()
{
    int repeats    = 0;
    int listIndex  = 0;
    int listLength = 0;
    int listTarget = 0;
    int avgSeqFind = 0;
    int avgSeqComp = 0;
    int avgSeqTime = 0;
    int avgBinFind = 0;
    int avgBinComp = 0;
    int avgBinTime = 0;

    Utilities util;
    Statistic stat;
    std::string inputString;
    std::stringstream inputStream;
    std::vector<int> listVector;

    // Listan pituus, luodaan aineisto
    listLength = util.askLength(listVector);
    util.populateWithOdds(listVector, listLength);
```

```
    // Toistojen määrä
    while (true)
    {
        std::cout << "Syota toistojen maara: ";
        std::getline(std::cin, inputString);

        inputStream.clear();
        inputStream.str(inputString);

        if (!(inputStream >> repeats))
        {
            std::cout << "Virheellinen syote, ole hyva ja anna positiivinen
kokonaisluku" << std::endl << std::endl;
        }
        else if (repeats < 1)
        {
            std::cout << "Virheellinen syote, toistoja pitaa suorittaa vahintaan
1" << std::endl << std::endl;
        }
        else
        {
            break;
        }
    }

    // Tulostusvektori = toistot + keskiarvot
    std::vector<std::vector<std::string>> output(repeats + 1);

    for (int i = 0; i < repeats; i++)
    {
        // Otsikkorivi
        output[i].push_back("Toisto  "  +  util.toString(i  +  1)  +  "  /  "  +
util.toString(repeats) + " - Haettu lukua: " + util.toString(listTarget));

        // Arvotaan haettava luku
        listTarget = util.randValue(listLength);

        // Perakkaisetsinta
        stat.reset();
        stat.timerStart();
        listIndex = sequentialSearch(stat, listVector, listTarget);
        stat.timerStop();

        output[i].push_back("Perakkaisetsinta:  ");
        output[i].push_back(listIndex   !=   -1   ?   "Loytyi,   paikka:   "   +
util.toString(listIndex) + "  " : "Lukua ei loytynyt  ");
        output[i].push_back("Vertailuja:  "  +  util.toString(stat.getCompari-
sons()) + "  ");
        output[i].push_back("Aikaa   kului:   "   +   util.toString(stat.get-
TimeElapsed()) + " us");

        avgSeqFind += (listIndex != -1 ? 1 : 0);
        avgSeqComp += stat.getComparisons();
        avgSeqTime += stat.getTimeElapsed();

        // Binaarietsinta
        stat.reset();
        stat.timerStart();
        listIndex = binarySearch(stat, listVector, listTarget, 0, listLength -
1);
        stat.timerStop();

        output[i].push_back("Binaarietsinta:    ");
        output[i].push_back(listIndex   !=   -1   ?   "Loytyi,   paikka:   "   +
util.toString(listIndex) + "  " : "Lukua ei loytynyt  ");
        output[i].push_back("Vertailuja:  "  +  util.toString(stat.getCompari-
sons()) + "  ");
        output[i].push_back("Aikaa   kului:   "   +   util.toString(stat.get-
TimeElapsed()) + " us");
```

```
        avgBinFind += (listIndex != -1 ? 1 : 0);
        avgBinComp += stat.getComparisons();
        avgBinTime += stat.getTimeElapsed();
    }

    // Keskiarvot tulostusvektoriin
    output[repeats].push_back("Keskiarvot " + util.toString(repeats) + " toiston
jalkeen");
    output[repeats].push_back("Perakkaisetsinta:  ");
    output[repeats].push_back("Loytyi " + util.toString(avgSeqFind) + " / " +
util.toString(repeats) + "  ");
    output[repeats].push_back("Vertailuja:     "     +     util.toString((int)
std::floor(avgSeqComp / repeats)) + "  ");
    output[repeats].push_back("Aikaa    kului:    "    +    util.toString((int)
std::floor(avgSeqTime / repeats)) + " us");
    output[repeats].push_back("Binaarietsinta:    ");
    output[repeats].push_back("Loytyi " + util.toString(avgBinFind) + " / " +
util.toString(repeats) + "  ");
    output[repeats].push_back("Vertailuja:     "     +     util.toString((int)
std::floor(avgBinComp / repeats)) + "  ");
    output[repeats].push_back("Aikaa    kului:    "    +    util.toString((int)
std::floor(avgBinTime / repeats)) + " us");

    // Tulostus
    feature_2_Print(output);

    std::cout << std::endl;
    std::cout << "Paina enter palataksesi valikkoon";
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

void feature_2_Print(std::vector<std::vector<std::string>> output)
{
    unsigned colWidth1 = 0;
    unsigned colWidth2 = 0;
    unsigned colWidth3 = 0;
    unsigned colWidth4 = 0;

    // Sarakkeiden leveydet
    for (unsigned i = 0; i < output.size(); i++)
    {
        if   (output[i].at(1).length()   >   colWidth1)   colWidth1   =   out-
put[i].at(1).length();
        if   (output[i].at(5).length()   >   colWidth1)   colWidth1   =   out-
put[i].at(5).length();

        if   (output[i].at(2).length()   >   colWidth2)   colWidth2   =   out-
put[i].at(2).length();
        if   (output[i].at(6).length()   >   colWidth2)   colWidth2   =   out-
put[i].at(6).length();

        if   (output[i].at(3).length()   >   colWidth3)   colWidth3   =   out-
put[i].at(3).length();
        if   (output[i].at(7).length()   >   colWidth3)   colWidth3   =   out-
put[i].at(7).length();

        if   (output[i].at(4).length()   >   colWidth4)   colWidth4   =   out-
put[i].at(4).length();
        if   (output[i].at(8).length()   >   colWidth4)   colWidth4   =   out-
put[i].at(8).length();
    }

    // Tulostus
    for (unsigned i = 0; i < output.size(); i++)
    {
        std::cout << std::endl;
        std::cout << output[i].at(0) << std::endl;
        std::cout  <<  std::string((colWidth1  +  colWidth2  +  colWidth3  +  col-
Width4), '-') << std::endl;
```

```
        std::cout << std::left << std::setw(colWidth1) << output[i].at(1);
        std::cout << std::left << std::setw(colWidth2) << output[i].at(2);
        std::cout << std::left << std::setw(colWidth3) << output[i].at(3);
        std::cout << std::left << std::setw(colWidth4) << output[i].at(4) <<
std::endl;
        std::cout << std::left << std::setw(colWidth1) << output[i].at(5);
        std::cout << std::left << std::setw(colWidth2) << output[i].at(6);
        std::cout << std::left << std::setw(colWidth3) << output[i].at(7);
        std::cout << std::left << std::setw(colWidth4) << output[i].at(8) <<
std::endl;
    }
}

/*
* feature_3.cpp
* ======
*/

#include "main.h"

void feature_3()
{
    int listPrints = 0;
    int listLength = 0;

    Utilities util;
    Statistic stat;
    std::vector<int> listVector;

    // Listan pituus ja näytettävien alkioiden määrä
    listLength = util.askLength(listVector);
    listPrints = util.askPrints(listLength);

    // Luodaan aineisto
    util.populateWithRand(listVector, listLength);

    if (listPrints > 0)
    {
        std::cout << std::endl;
        std::cout << "Aineiston " << util.toString(listPrints) << " ensimmaista
alkiota ennen jarjestelya:" << std::endl;
        util.printList(listVector, listPrints);
    }

    // Järjestellään
    insertionSort(stat, listVector);

    if (listPrints > 0)
    {
        std::cout << std::endl;
        std::cout << "Aineiston " << util.toString(listPrints) << " ensimmaista
alkiota jarjestelyn jalkeen:" << std::endl;
        util.printList(listVector, listPrints);
    }

    // Paluu valikkoon
    std::cout << std::endl;
    std::cout << "Paina enter palataksesi paavalikkoon";
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
```

```
/*
* feature_4.cpp
* ======
*/

#include "main.h"

void feature_4()
{
    int listPrints = 0;
    int listLength = 0;

    Utilities util;
    Statistic stat;
    std::vector<int> listVector;

    // Listan pituus ja näytettävien alkioiden määrä
    listLength = util.askLength(listVector);
    listPrints = util.askPrints(listLength);

    // Luodaan aineisto
    util.populateWithRand(listVector, listLength);

    if (listPrints > 0)
    {
        std::cout << std::endl;
        std::cout << "Aineiston " << util.toString(listPrints) << " ensimmaista
alkiota ennen jarjestelya:" << std::endl;
        util.printList(listVector, listPrints);
    }

    // Järjestellään
    listVector = mergeSort(stat, listVector);

    if (listPrints > 0)
    {
        std::cout << std::endl;
        std::cout << "Aineiston " << util.toString(listPrints) << " ensimmaista
alkiota jarjestelyn jalkeen:" << std::endl;
        util.printList(listVector, listPrints);
    }

    // Paluu valikkoon
    std::cout << std::endl;
    std::cout << "Paina enter palataksesi paavalikkoon";
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

/*
* feature_5.cpp
* ======
*/

#include "main.h"

void feature_5()
{
    int selection = -1;
    std::string inputString;
    std::stringstream inputStream;

    while (selection != 0)
    {
        std::cout << std::endl;
        std::cout << "Valitse ajettava toiminto:" << std::endl;
        std::cout << "--------------------------" << std::endl;
        std::cout << "[1] - Valintalajittelu" << std::endl;
        std::cout << "[2] - Pikalajittelu" << std::endl;
        std::cout << "[3] - Suorituskykyvertailu" << std::endl;
        std::cout << "[0] - Palaa paavalikkoon" << std::endl;
```

```
        while (true)
        {
            std::cout << std::endl;
            std::cout << "Valinta: ";
            std::getline(std::cin, inputString);

            inputStream.clear();
            inputStream.str(inputString);

            if (!(inputStream >> selection) || selection < 0 || selection > 3)
            {
                std::cout << "Virheellinen valinta, ole hyva ja valitse
numeroista 0-3" << std::endl;
            }
            else
            {
                break;
            }
        }

        switch (selection)
        {
            case 1:
            case 2:
                feature_5_Sort(selection);
                break;
            case 3:
                feature_5_Compare();
                break;
        }
    }
}

void feature_5_Sort(int algorithm)
{
    int listPrints = 0;
    int listLength = 0;

    Utilities util;
    Statistic stat;
    std::vector<int> listVector;

    // Listan pituus ja näytettävien alkioiden määrä
    listLength = util.askLength(listVector);
    listPrints = util.askPrints(listLength);

    // Luodaan aineisto
    util.populateWithRand(listVector, listLength);

    if (listPrints > 0)
    {
        std::cout << std::endl;
        std::cout << "Aineiston " << util.toString(listPrints) << " ensimmaista
alkiota ennen jarjestelya:" << std::endl;
        util.printList(listVector, listPrints);
    }

    // Järjestellään
    if (algorithm == 1)
    {
        selectionSort(stat, listVector);
    }
    else
    {
        quickSort(stat, listVector, 0, listVector.size() - 1);
    }
```

```
    if (listPrints > 0)
    {
        std::cout << std::endl;
        std::cout << "Aineiston " << util.toString(listPrints) << " ensimmaista
alkiota jarjestelyn jalkeen:" << std::endl;
        util.printList(listVector, listPrints);
    }

    // Paluu valikkoon
    std::cout << std::endl;
    std::cout << "Paina enter palataksesi valikkoon";
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

void feature_5_Compare()
{
    int selection  = 0;
    int listLength = 0;

    Utilities util;
    Statistic stat;
    std::string inputString;
    std::stringstream inputStream;

    std::vector<int> baseVector;
    std::vector<int> tempVector;
    std::vector<std::vector<std::string>> output(11);

    std::cout << std::endl;
    std::cout << "Valitse suorituskykyvertailu:" << std::endl;
    std::cout << "-----------------------------" << std::endl;
    std::cout << "[1] - 1000 -> 10000" << std::endl;
    std::cout << "[2] - 10000 -> 100000" << std::endl;
    std::cout << "[3] - 100000 -> 1000000" << std::endl;
    std::cout << "[0] - Palaa edelliseen valikkoon" << std::endl;

    while (true)
    {
        std::cout << std::endl;
        std::cout << "Valinta: ";
        std::getline(std::cin, inputString);

        inputStream.clear();
        inputStream.str(inputString);

        if (!(inputStream >> selection) || selection < 0 || selection > 3)
        {
            std::cout << "Virheellinen valinta, ole hyva ja valitse numeroista
0-3" << std::endl;
        }
        else
        {
            switch (selection)
            {
                case 1:
                    listLength = 1000;
                    break;
                case 2:
                    listLength = 10000;
                    break;
                case 3:
                    listLength = 100000;
                    break;
            }

            // Pois inputista
            break;
        }
    }
```

```
    // Palattaessa ohitetaan suoritus
    if (selection != 0)
    {
        try
        {
            // Otsikkorivi
            output[0].push_back("Taulu: ");

            for (int i = 0; i < 4; i++)
            {
                output[0].push_back("| ");
                output[0].push_back("Vertail: ");
                output[0].push_back("Sijoitu: ");
                output[0].push_back("Operaat: ");
                output[0].push_back("Aika us: ");
            }

            // Suorituskykyvertailu
            for (int i = 1; i <= 10; i++)
            {
                output[i].push_back(util.toString(listLength * i));
                output[i].push_back("|");

                //  Varmistetaan  että  vektorit  mahtuvat  muistiin,  luodaan
aineisto
                baseVector.resize(listLength * i);
                tempVector.resize(listLength * i);
                util.populateWithRand(baseVector, listLength * i);

                // Lomituslajittelu
                tempVector = baseVector;
                stat.reset();
                stat.timerStart();
                tempVector = mergeSort(stat, tempVector);
                stat.timerStop();

                output[i].push_back(util.toString(stat.getComparisons())   +   "
");
                output[i].push_back(util.toString(stat.getAssignments())   +   "
");
                output[i].push_back(util.toString(stat.getOperations()) + " ");
                output[i].push_back(util.toString(stat.getTimeElapsed())   +   "
");
                output[i].push_back("|");

                // Pikalajittelu
                tempVector = baseVector;
                stat.reset();
                stat.timerStart();
                quickSort(stat, tempVector, 0, tempVector.size() - 1);
                stat.timerStop();

                output[i].push_back(util.toString(stat.getComparisons())   +   "
");
                output[i].push_back(util.toString(stat.getAssignments())   +   "
");
                output[i].push_back(util.toString(stat.getOperations()) + " ");
                output[i].push_back(util.toString(stat.getTimeElapsed())   +   "
");
                output[i].push_back("|");

                // Valintalajittelu
                tempVector = baseVector;
                stat.reset();
                stat.timerStart();
                selectionSort(stat, tempVector);
                stat.timerStop();
```

```
                output[i].push_back(util.toString(stat.getComparisons())   +   "
");
                output[i].push_back(util.toString(stat.getAssignments())   +   "
");
                output[i].push_back(util.toString(stat.getOperations()) + " ");
                output[i].push_back(util.toString(stat.getTimeElapsed())   +   "
");
                output[i].push_back("|");

                // Lisäyslajittelu
                tempVector = baseVector;
                stat.reset();
                stat.timerStart();
                insertionSort(stat, tempVector);
                stat.timerStop();

                output[i].push_back(util.toString(stat.getComparisons())   +   "
");
                output[i].push_back(util.toString(stat.getAssignments())   +   "
");
                output[i].push_back(util.toString(stat.getOperations()) + " ");
                output[i].push_back(util.toString(stat.getTimeElapsed())   +   "
");
            }

            feature_5_Print(output);
        }
        catch (const std::bad_alloc& e)
        {
            std::cout << "Testi pysahtyi muistivirheeseen, todennakoisesti liian
suuri lista (" << e.what() << ")" << std::endl;
        }

        // Paluu valikkoon
        std::cout << std::endl;
        std::cout << "Paina enter palataksesi valikkoon";
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
}

void feature_5_Print(std::vector<std::vector<std::string>> output)
{
    Utilities util;
    std::vector<int> colWidths(21, 0);

    // Sarakkeiden leveydet
    for (int i = 0; i < 11; i++)
    {
        for (int j = 0; j < 21; j++)
        {
            if (output[i].at(j).length() > (unsigned) colWidths.at(j))
            {
                colWidths.at(j) = output[i].at(j).length();
            }
        }
    }

    // Tulostetaan otsikkorivi
    std::cout << std::endl;
    std::cout << std::left << std::setw(colWidths.at(0)) << " ";
    std::cout << std::left << std::setw(util.accumulate(colWidths, 1, 6)) << "|
Lomituslajittelu";
    std::cout << std::left << std::setw(util.accumulate(colWidths, 6, 11)) <<
"| Pikalajittelu";
    std::cout << std::left << std::setw(util.accumulate(colWidths, 11, 16)) <<
"| Valintalajittelu";
    std::cout << std::left << std::setw(util.accumulate(colWidths, 16, 21)) <<
"| Lisayslajittelu" << std::endl;
```

```
    // Tulostetaan tulokset
    for (int i = 0; i < 11; i++)
    {
        for (int j = 0; j < 21; j++)
        {
            std::cout << std::left << std::setw(colWidths.at(j)) << out-
put[i].at(j);
        }

        std::cout << std::endl;

        // Viiva sarakeotsikoiden alla
        if (i == 0)
        {
            std::cout << std::string(util.accumulate(colWidths, 0, 21), '-') <<
std::endl;
        }
    }
}

/*
* main.cpp
* =======
*/

#include "main.h"

// Alustetaan staattinen random engine utilities luokkaan
std::default_random_engine Utilities::generator = std::default_random_en-
gine((int) std::time(nullptr));

int main()
{
    std::cout << "Testipeti etsinta- ja lajittelualgoritmeille - Valkoja Iiro"
<< std::endl;

    int selection = -1;
    std::string inputString;
    std::stringstream inputStream;

    while (selection != 0)
    {
        std::cout << std::endl;
        std::cout << "Valitse ajettava toiminto:" << std::endl;
        std::cout << "--------------------------" << std::endl;
        std::cout << "[1] - Perakkaisetsinta" << std::endl;
        std::cout << "[2] - Binaarietsinta ja suorituskykyvertailu" <<
std::endl;
        std::cout << "[3] - Lisayslajittelu" << std::endl;
        std::cout << "[4] - Lomituslajittelu" << std::endl;
        std::cout << "[5] - Lisaalgoritmit ja suorituskykyvertailu" <<
std::endl;
        std::cout << "[0] - Lopeta" << std::endl;

        while (true)
        {
            std::cout << std::endl;
            std::cout << "Valinta: ";
            std::getline(std::cin, inputString);

            inputStream.clear();
            inputStream.str(inputString);

            if (!(inputStream >> selection) || selection < 0 || selection > 5)
            {
                std::cout << "Virheellinen valinta, ole hyva ja valitse
numeroista 0-5" << std::endl;
            }
```

```
            else
            {
                break;
            }
        }

        switch (selection)
        {
            case 1:
                feature_1();
                break;
            case 2:
                feature_2();
                break;
            case 3:
                feature_3();
                break;
            case 4:
                feature_4();
                break;
            case 5:
                feature_5();
                break;
        }
    }

    return 0;
}

/*
* statistic.cpp
* ======
*/

#include "main.h"

Statistic::Statistic()
{
    comparisons = 0;
    assignments = 0;
}

void Statistic::timerStart()
{
    startTime = std::chrono::high_resolution_clock::now();
}

void Statistic::timerStop()
{
    stopTime = std::chrono::high_resolution_clock::now();
}

void Statistic::addComparisons(int addition)
{
    comparisons += addition;
}

void Statistic::addAssignments(int addition)
{
    assignments += addition;
}

int Statistic::getComparisons()
{
    return comparisons;
}

int Statistic::getAssignments()
{
    return assignments;
}
```

```
int Statistic::getOperations()
{
    return comparisons + assignments;
}

int Statistic::getTimeElapsed()
{
    return (int) std::chrono::duration_cast<std::chrono::microseconds>(stopTime
- startTime).count();
}

bool Statistic::isEqual(int first, int second)
{
    comparisons++;
    return (first == second);
}

bool Statistic::isGreater(int first, int second)
{
    comparisons++;
    return (first > second);
}

void Statistic::swap(int &first, int &second)
{
    assignments += 3;

    int temp = first;
    first = second;
    second = temp;
}

void Statistic::reset()
{
    comparisons = 0;
    assignments = 0;
}

/*
* utilities.cpp
* ======
*/

#include "main.h"

int Utilities::askLength(std::vector<int> &list)
{
    int inputInt;
    std::string inputString;
    std::stringstream inputStream;

    while (true)
    {
        try
        {
            std::cout << std::endl;
            std::cout << "Syota aineiston koko: ";
            std::getline(std::cin, inputString);

            inputStream.clear();
            inputStream.str(inputString);

            if (!(inputStream >> inputInt))
            {
                std::cout << "Virheellinen syote, ole hyva ja anna positiivinen
kokonaisluku" << std::endl;
            }
```

```
            else if (inputInt < 1)
            {
                std::cout << "Virheellinen syote, aineiston koon pitaa olla
vahintaan 1" << std::endl;
            }
            else if (inputInt >= (int) (INT_MAX / 2))
            {
                std::cout << "Virheellinen syote, luvun pitaa olla pienempi kuin
" << (int) (INT_MAX / 2) << std::endl;
            }
            else
            {
                list.resize(inputInt);
                break;
            }
        }
        catch (const std::bad_alloc& e)
        {
            std::cout << "Muistivirhe, todennakoisesti liian suuri lista (" <<
e.what() << ")" << std::endl;
        }
    }

    return inputInt;
}

int Utilities::askTarget(int randMax)
{
    int inputInt;
    std::string inputString;
    std::stringstream inputStream;

    while (true)
    {
        std::cout << "Syota haettava luku tai arvo syottamalla [R]: ";
        std::getline(std::cin, inputString);

        if (inputString.compare("R") == 0 || inputString.compare("r") == 0)
        {
            inputInt = randValue(randMax);

            std::cout << "Haettavaksi luvuksi arvottiin: " << inputInt <<
std::endl;
            break;
        }
        else
        {
            inputStream.clear();
            inputStream.str(inputString);

            if (!(inputStream >> inputInt))
            {
                std::cout << "Virheellinen syote, ole hyva ja anna kokonaisluku
valilta " << INT_MIN << " ja " << INT_MAX << std::endl << std::endl;
            }
            else
            {
                break;
            }
        }
    }

    return inputInt;
}
```

```
int Utilities::askPrints(int max)
{
    int inputInt;
    std::string inputString;
    std::stringstream inputStream;

    while (true)
    {
        std::cout << std::endl;
        std::cout << "Syota tulostettavien lukujen maara: ";
        std::getline(std::cin, inputString);

        inputStream.clear();
        inputStream.str(inputString);

        if (!(inputStream >> inputInt))
        {
            std::cout << "Virheellinen syote, ole hyva ja anna positiivinen
kokonaisluku" << std::endl;
        }
        else if (inputInt < 0)
        {
            std::cout << "Virheellinen syote, maara ei voi olla negatiivinen"
<< std::endl;
        }
        else
        {
            break;
        }
    }

    if (inputInt > max)
    {
        inputInt = max;
    }

    return inputInt;
}

int Utilities::randValue(int randMax)
{
    std::uniform_int_distribution<int> distribution(0, randMax);
    return distribution(generator);
}

int Utilities::accumulate(std::vector<int> &list, int start, int end)
{
    int result = 0;

    for (int i = start; i < end; i++)
    {
        result += list.at(i);
    }

    return result;
}

int Utilities::widerStr(std::string str1, std::string str2)
{
    int wider = str1.length() > str2.length() ? str1.length() : str2.length();

    return wider;
}

std::string Utilities::toString(int number)
{
    std::stringstream strStream;
    strStream << number;
    return strStream.str();
}
```

```
void Utilities::populateWithOdds(std::vector<int> &list, int size)
{
    int value = 1;

    for (int i = 0; i < size; i++)
    {
        list[i] = value;
        value += 2;
    }
}

void Utilities::populateWithRand(std::vector<int> &list, int size)
{
    std::uniform_int_distribution<int> distribution(0, (size > INT_MAX / 10 ?
size : size * 10));

    for (int i = 0; i < size; i++)
    {
        list[i] = distribution(generator);
    }
}

void Utilities::printList(std::vector<int> &list, int size)
{
    for (int i = 0; i < size; i++)
    {
        std::cout << list.at(i);

        if (i < size - 1)
        {
            std::cout << ", ";
        }
        else
        {
            std::cout << std::endl;
        }
    }
}
```