Извештај

# Аутор

Вук Радмиловић

Индекс: СВ73/2020

Студент ФТН-а на смеру „Софтверско инжењерство и информационе технологије“

# Опис рада У/И подсистема

Пројекат има две главне функције које се тичу улазног подсистема.

1. int handleCommandLine(int argc, char \*argv[]); који обрађује аргументе које корисник уноси путем командне линије
2. void handleInputGUI(); који формира графички интерфејс преко којег корисник може унети све потребне аргументе

У вези излазног подсистема имамо 3 различите врсте излаза:

1. визуелизација сортирања путем графичког интерфејса који је обављен класом VisualizerWindow
2. испис прикупљених података и сортираних података у конзолу путем функције void printAllData(ostream& stream, vector<Flight> flights, Arguments args, vector<Info> infos) ;
3. испис прикупљених података и сортираних података у излазни фајл помоћу исте функције као и 2)

Обједињење решења за 2) и 3) постигунуто је коришћењем ostream& stream који нам омогућава да уносимо податке у било који стрим, без обзира да ли је то фајл или конзола.

# Списак класа, изузетака и слободних функција

## Класе:

* Arguments: 5 различитих изузетака: 1) „Невалидан улазни фајл“ 2) „Невалидан излазни фајл“ 3) „Невалидна вредност за сорт“ 4) „Невалидна вредност за критеријум“ 5) „Невалидна вредност за поредак“
* Flight
* Info
* MyWindow
* Sort
* VisualizerWindow

## Енум класе:

* SortAlgorithm
* Criterium
* Order

## Слободне функције:

Helper.cpp:

* vector<string> split(const string &line, char seperator)
* int compareTime(string first, string second)

Sort.cpp:

* void swap(Flight& f1, Flight& f2)
* int min(int x1, int x2)
* void resetOriginalIndexes(vector<Flight>& data)

VisualizerWindow.cpp:

* int calculateWidth(vector<Info>& infos)
* string visualizationHeader()

main.cpp:

* int main(int argc, char \*argv[])
* void handleInputGUI()
* int handleCommandLine(int argc, char \*argv[])
* void printFlights(ostream& stream, vector<Flight> flights)
* void printGatheredData(ostream& stream, Arguments args, vector<Info> infos)
* void printAllData(ostream& stream, vector<Flight> flights, Arguments args, vector<Info> infos)
* const string SortAlgorithmToString(SortAlgorithm sortAlgorithm)
* const string CriteriumToString(Criterium criterium)
* const string OrderToString(Order order)
* void sortFlights(Arguments args)
* void performSort(Arguments args, vector<Flight> data, vector<Flight> originalData, Sort\* sorter)
* vector<Flight> loadFlights(const string path)

# Објашњење најбитнијих атрибута класа, функција чланица, слободних функција и изузетака

## Атрибути класа:

* Arguments::string \_inputFilePath – путања до изворног фајла
* Arguments::string \_outputFilePath – путања до излазног фајла
* Arguments::SortAlgorithm \_sortAlgorithm – алгоритам који се користи за сортирање
* Arguments::Criterium \_criterium – критеријум по којем се врши сортирање
* Arguments::Order \_order – растући или опадајући поредак елемената сортираног вектора
* Flight::string \_destination – одредиште лета
* Flight::string \_departureTime – време полетања
* Flight::string \_flightNumber – број лета
* Flight::string \_gateNumber – број излаза
* Info::unsigned long numCmps – број извршених поређења приликом једне итерације сорта
* Info::unsigned long numMoves – број померених елемената приликом једне итерације сорта
* Info::unordered\_map<int, int> changes – мапа са учињеним променама током једне итерације сорта
* VisualizerWindow::vector<Flight> \_data – подаци након сортирања
* VisualizerWindow::vector<Flight> \_originalData – подаци пре сортирања
* VisualizerWindow::vector<Info> \_infos – информације прикупљене током сортирања
* VisualizerWindow::vector<vector<Circle\*>> crcNodes – кругови који означавају позицију елемената током итерације сортирања
* VisualizerWindow::vector<vector<Line\*>> lnConnections – линије које повезују кругове и означавају померање елемената током итерација сортирања
* VisualizerWindow::vector<Out\_box\*> endBoxes – „кутије“ у које постављамо текст који означава елементе резултантног низа сортираних летова

## Функције чланице:

* Arguments::Arguments(string inputFilePath, string outputFilePath,

string sortAlgorithm, string criterium, string order) – конструктор који аутоматски проверава валидност унесених аргумената и баца изузетке у случају грешке

* int Flight::compare(const Flight& other, Criterium criterium) – функција која се користи за поређење два Flight објекта
* MyWindow::MyWindow(Point xy, int w, int h, const string& title) – конструктор који иницијализује прозор који се користи за унос података за сортирање
* vector<Info> SelectionSort::sort(vector<Flight>& data, Criterium criterium, Order order) – функција која сортира вектор летова по специфицираном критеријуму и поретку помоћу селекшн сорта
* vector<Info> MergeSort::sort(vector<Flight>& data, Criterium criterium, Order order) – функција која сортира вектор летова по специфицираном критеријуму и поретку помоћу мрџ сорта
* void MergeSort::merge(vector<Flight>& data, int left, int mid, int right, Criterium criterium, Order order, Info& info) – функција коју користи мрџ сорт приликом сортирања. Спаја два дела вектора по сортираном редоследу
* VisualizerWindow::VisualizerWindow(Point xy, int w, int h, const string& title, vector<Info>& infos, vector<Flight>& data, vector<Flight>& originalData) – конструктор који формира прозор који ће вршити визуелизацију сортирања летова по корисниковим критеријумима
* void VisualizerWindow::previous() – врши прелаз на прошлу итерацију током визуелизације сортирања
* void VisualizerWindow::next() – врши прелаз на следећу итерацију током визуелизације сортирања
* void VisualizerWindow::reset() – брише све линије које повезују кругове (елементе низа) током визуелизације сортирања
* void VisualizerWindow::simulateToEnd() – приказује све линије које повезују кругове (елементе низа) током визуелизације сортирања

## Слободне функције:

Helper.cpp:

* vector<string> split(const string &line, char seperator) – функција која дели стринг на низ стрингова који су били подељени сеператором
* int compareTime(string first, string second) – функција која пореди два 24-часовна времена

VisualizerWindow.cpp:

* int calculateWidth(vector<Info>& infos) – динамички рачуна потребну ширину прозора у зависности од броја итерација алгоритма током сортирања

main.cpp:

* int main(int argc, char \*argv[]) – главна покретачка функција програма, позива функције за улаз података командном линијом или преко графичког интерфејса
* void handleInputGUI() – креира графички интерфејс за унос података за сортирање и обрађује унос
* int handleCommandLine(int argc, char \*argv[]) – обрађује случај коришћења командне линије за унос података уместо графичког интерфејса
* void printAllData(ostream& stream, vector<Flight> flights, Arguments args, vector<Info> infos) – исписује све потребне податке након сортирања – сортиран низ и податке прикупљене током сортирања
* void sortFlights(Arguments args) – покреће механизам за сортирање одабраних података одабраним алгоритмом по одабраним критеријумима
* vector<Flight> loadFlights(const string path) – учитава летове из фајла у вектор летова

## Изузеци:

Arguments.cpp:

* „Невалидан улазни фајл“
* „Невалидан излазни фајл“
* „Невалидна вредност за сорт“
* „Невалидна вредност за критеријум“
* „Невалидна вредност за поредак“

# Структура аргумената командне линије и пример коришћења

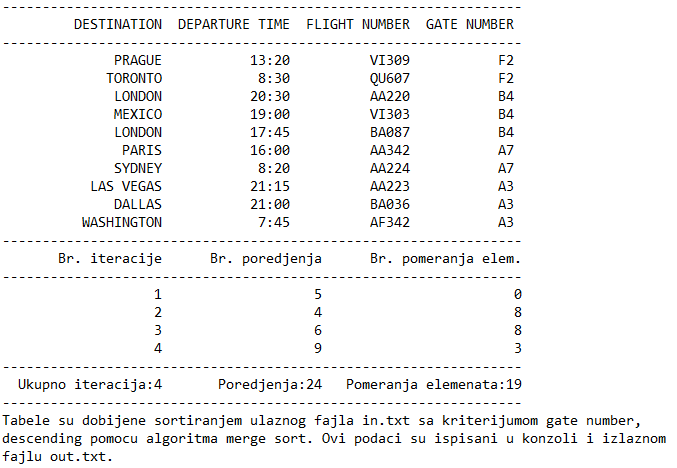
Аргументи:

1. Путања до улазног фајла
2. Путања до излазног фајла
3. Алгоритам за сортирање (1 – селекшн, 2 – мрџ)
4. Критеријум сортирања (1 – број лета, 2 – одредиште, 3 – време полетања, 4 – број излаза)
5. Поредак (1 – растући, 2 – опадајући)

Пример коришћења: in.txt out.txt 1 1 2

То значи сортирање података из in.txt у out.txt помоћу селекшн сорта по критеријуму број лета у опадајућем поретку.

# Структура излазне датотеке



На почетку излазне датотеке се налази заглавље са исписаним називима поља којима су представљени летови. Затим имамо приказ сортираног низа по корисниковом критеријуму. После тога видимо заглавље које садржи називе поља прикупљених приликом сортирања, а затим и та поља приказана кроз итерације (бр. Поређења и бр. Померања елемената). Након тога имамо збир где видимо укупни број итерација, поређења и померања елемената. На крају видимо сумарни опис који описује шта се налази у овој излазној датотеци тј. који алгоритам је коришћен, који су критеријум и поредак као и који су излазни и улазни фајлови.

# Опис merge sort-a

Алгоритам одабран за овај пројекат јесте итеративни merge sort. То је divide and conquer алгоритам чија рекурзивна верзија дели низ у 2 половине, позива поново себе за те 2 половине а затим спаја те 2 сада сортиране половине. Временска комплексност алгоритма је O(nlogn), што је веома добро за сорт алгоритам (selection sort има O(n2) временску комплексност). У овом пројекту коришћена је итеративна верзија овог алгоритма, што смањује overhead који може постати прилично велик у нашем случају где имамо неке додатне параметре због критеријума и чувања додатних информација приликом сортирања како би подаци били касније визуелизовани и приказани кориснику, тако да је због те практичности одабран овај алгоритам. Пошто радимо итеративно, уместо да позивамо рекурзивне функције за половине низа ми крећемо од тога да делимо низ на низове од по 2 елемената, затим 4, 8 итд... то јест на степене двојке. Ово значи да ће у нашем случају где имамо 10 елемената остати низ од 2 елемента који није сортиран након што је првих 8 елемената сортирано, што само значи да ће он бити сортиран у последњој итерацији.

# Напредни ООП концепти

## Наслеђивање

Наслеђивање је коришћено за сорт алгоритме. Имамо базну апстрактну class Sort која је онда наслеђена од стране class SelectionSort и class MergeSort.

Наслеђивање се такође користи код графичког интерфејса, где class MyWindow наслеђује класу Graph\_lib::Window, а исто то ради и class VisualizerWindow. Овим поступком добијамо приступ разним функцијама и атрибутима класе Window који нам омогућавају да лакше имплементирамо графички интерфејс без поновног дефинисања тих функција и атрибута.

## Преклапање оператора

Преклапање оператора врши се у класи Flight где је преклопљен оператор << (friend ostream& operator <<(ostream& stream, const Flight& flight)) ради лакшег и практичнијег исписивања лета у конзолу и фајлове по формату који је потребан за пројекат.

## Полиморфизам

Полиморфизам је такође коришћен за сорт алгоритме, јер class SelectionSort и class MergeSort обе имплементирају своју верзију функције vector<Info> sort(vector<Flight>& data, Criterium criterium, Order order). У main.cpp је ово искоришћено у функцији void performSort(Arguments args, vector<Flight> data, vector<Flight> originalData, Sort\* sorter) са vector<Info> infos = (\*sorter).sort(data, args.criterium(), args.order()); Овим је омогућено прослеђивање било којег сорт алгоритма овој функцији и њено извршавање.

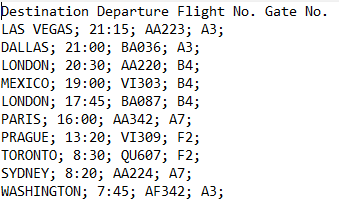
# Опис графичке спреге

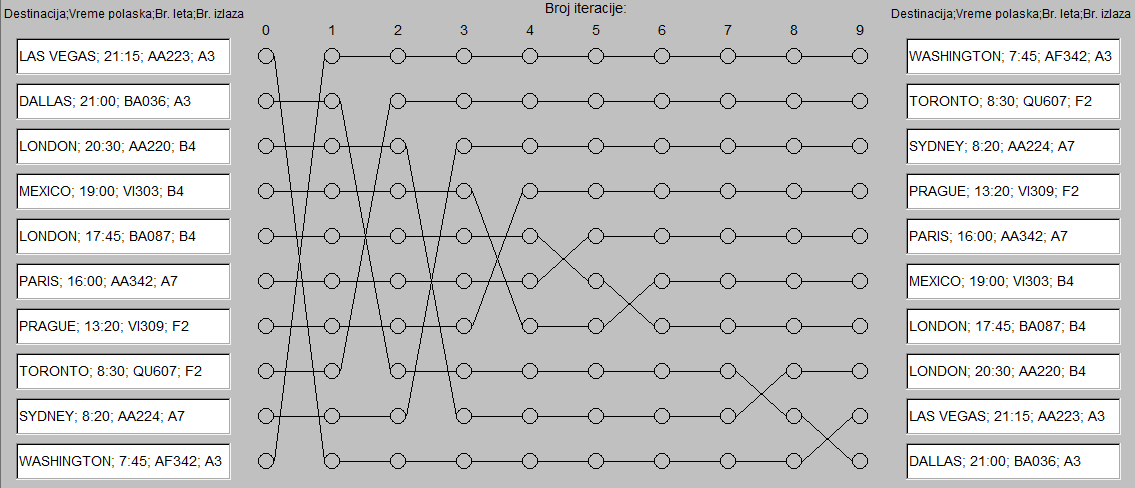
У овом пројекту постоје 2 графичка интерфејса.

1. class MyWindow : public Window – ово је графички интерфејс који се користи у случају да корисник не жели да унесе аргументе за сортирање преко командне линије. Тада се отвара графички интерфејс који му пружа могућност одабира улазне, излазне датотеке као и критеријума, поретка и алгоритма за сортирање. Као што је дискутовано у прошлом поглављу, наслеђује се класа Window из Graph\_lib. In\_box је класа која се користи за прављење поља у којих се може унети текст који ће означити које опције је корисник одабрао. Постоји и једно дугме класе Button – sortButton који покреће сортирање низа и води до графичког интерфејса за визуелизацију сортирања. Овде постоји и један Text објекат који се користи за приказивање грешке у случају да корисник унесе неке невалидне податке.
2. class VisualizerWindow : public Window – графички интерфејс који се користи за визуелизацију сортирања итерацију по итерацију после самог сортирања. Такође наслеђује Window из Graph\_lib. За приказ позиције елемената током сортирања користи се Circle који нам омогућава цртање кругова. За приказ померања елемената из позиције у позицију кроз итерације користи се Line који нам омогућава цртање линија које повезују кругове. Out\_box се користи за приказ „Кутија“ у које постављамо текст за резултантни низ сортираних летова. Интерфејс има 4 дугмета направљених помоћу класе Button: дугме за симулирање прошле итерације, следеће итерације, за ресетовање симулације и повратак на почетак, и за симулирање до краја. Ширина овог интерфејса се може аутоматски израчунавати у зависности од броја итерација које је потребно симулирати, док су остали атрибути фиксно постављени или дати од стране корисника.

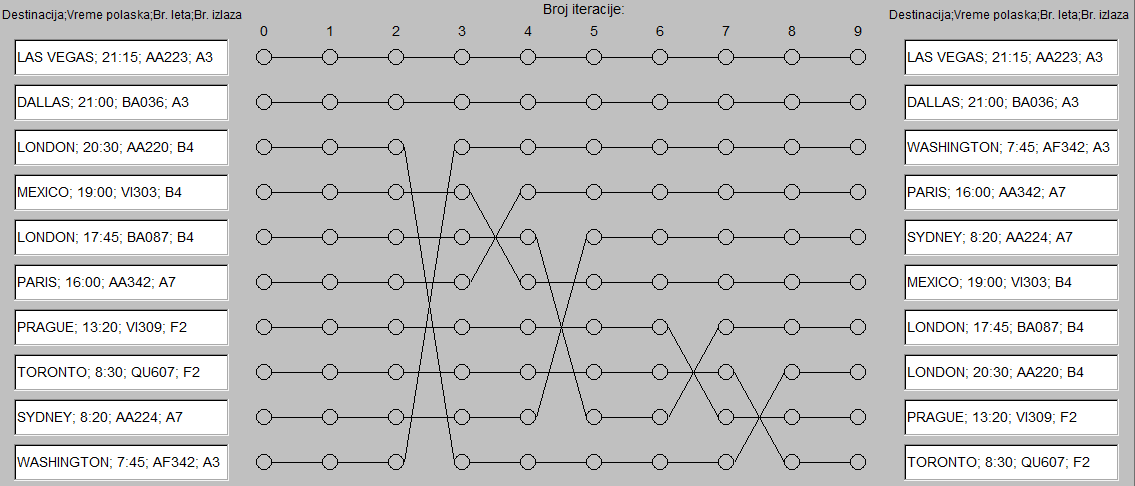
# Тестни случајеви

Фајл in.txt:

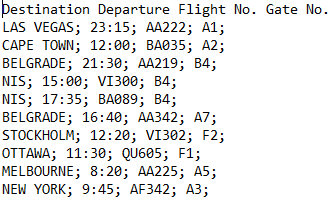


Уколико сортирамо по дестинацији, са selection sort-ом и опадајућим поретком: 

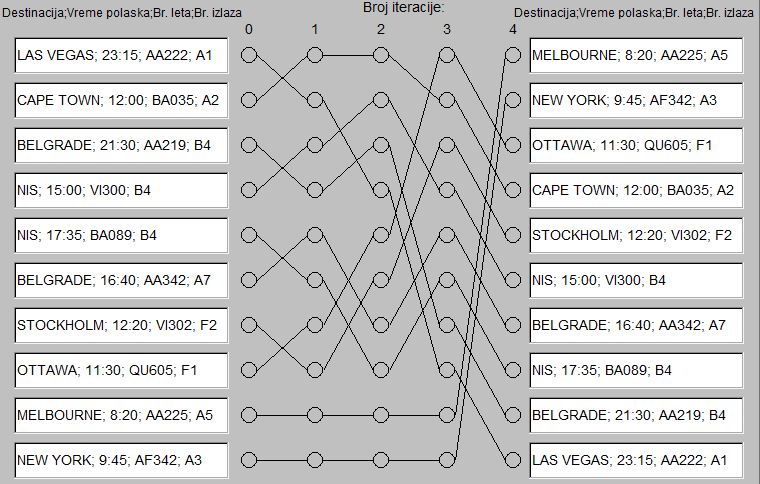
Уколико сортирамо по броју излаза, са selection sort-ом и растућим поретком:



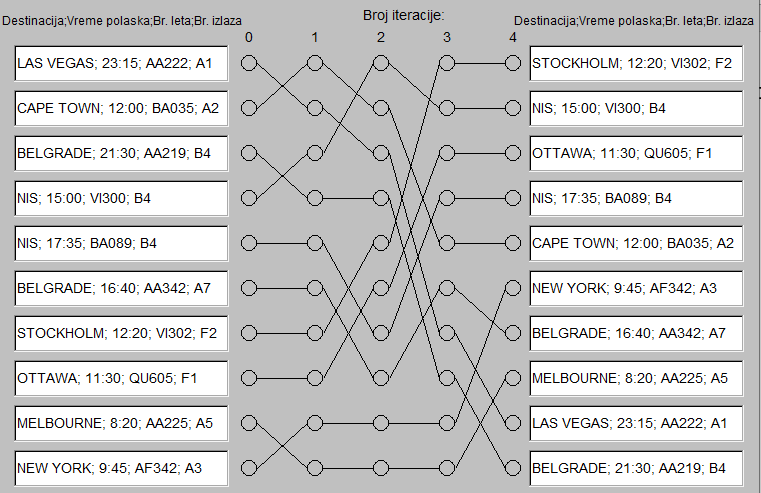
Фајл in2.txt:



Уколико сортирамо по времену поласка, са merge sort-ом и растућим поретком:

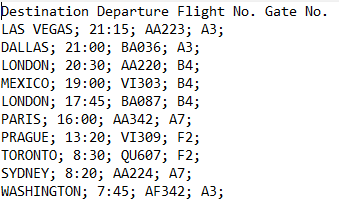


Уколико сортирамо по броју лета, са merge sort-ом и опадајућим поретком:

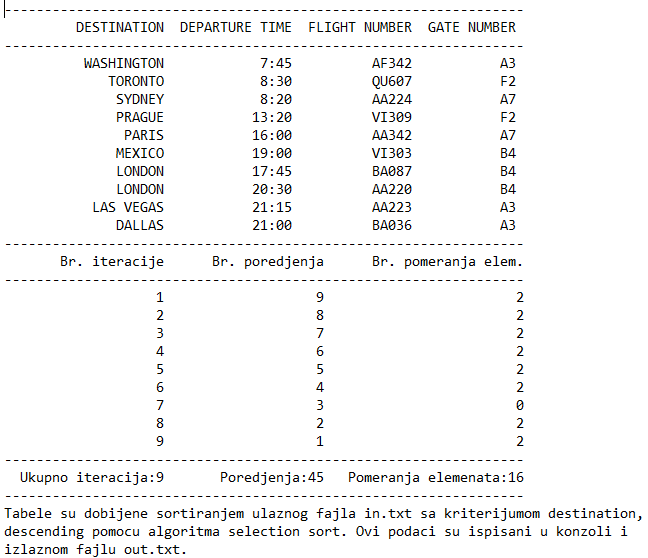


# Излазни резултати

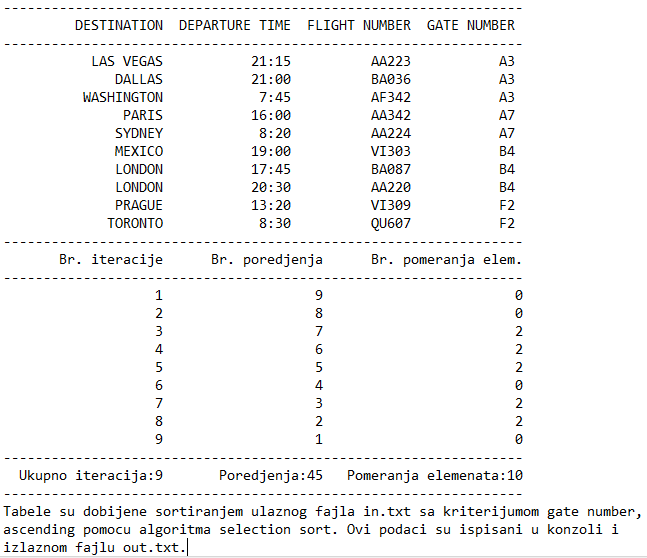
Фајл in.txt:



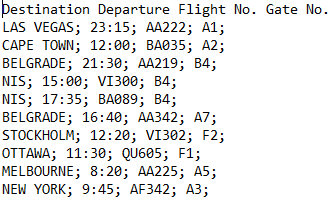
Уколико сортирамо по дестинацији, са selection sort-ом и опадајућим поретком:



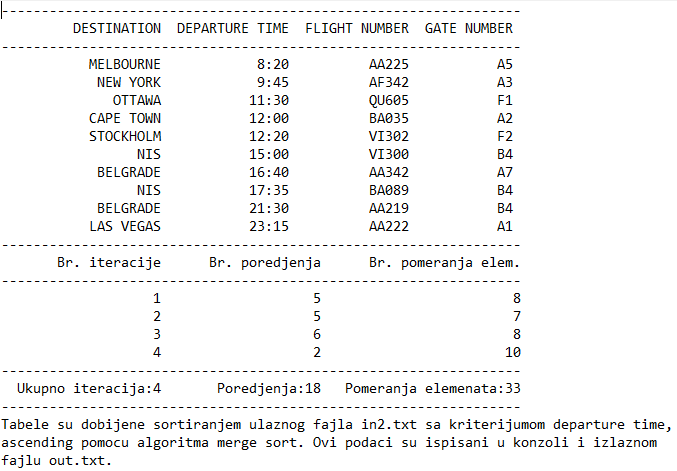
Уколико сортирамо по броју излаза, са selection sort-ом и растућим поретком:



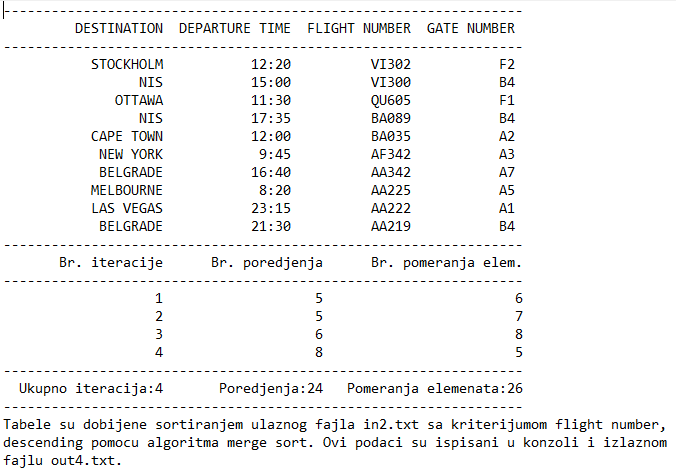
Фајл in2.txt:



Уколико сортирамо по времену поласка, са merge sort-ом и растућим поретком:



Уколико сортирамо по броју лета, са merge sort-ом и опадајућим поретком:



По овим резултатима можемо закључити да је merge sort много бржи алгоритам од selection sort-a. Због тога што су величине нашег улазног фајла мале и он има само 10 различитих летова по фајлу, сама брзина која би била потребна да се сортирање изврши на модерним рачунарима би била незнатна. Ипак, видљив је тренд веома брзог раста код selection sort-a, који је квадратни тј. O(n2) алгоритам. Број главних итерација потребан за сортирање је једнак количини података тј. броју различитих летова минус један, а број поређења је константан – 45 у овом случају јер ће за сваку главну итерацију бити одрађена још једна петља која ће вршити поређења са свим преосталим елементима. Због тога ће прва итерација вршити 9 поређења, друга 8 итд... до последње која врши 1 поређење.

Са друге стране, merge sort захтева само 4 итерације за наш улаз од 10 елемената. То је због тога што ће итеративни merge sort прво поделити низ на 5 поднизова од по 2 елемента у првој итерацији, затим 2 подниза од 4 елемента и 1 подниз од 2 елемента у другој итерацији, па 1 подниз од 8 елемената и 1 подниз од 2 елемента у трећој итерацији и на крају ће све то спојити у 1 сортиран низ од 10 елемената. Он ће увек делити поднизове у једнаке делове по степенима двојке, што значи да би исти број итерација био потребан и да је наш улаз имао 16 елемената (24).

Када би поредили ова два алгоритма за веће улазе, видели би да би merge sort достигао 10 главних итерација тек када добије улаз од 210 елемената, што је 1024. Selection sort би са тим улазом имао 1024 главних итерација, као и енорман број поређења (1023 у првој итерацији, 1022 у другој итд...). Видимо и да merge sort генерално има значајно мањи број поређења, и знамо да би тај број много спорије растао јер је merge sort O(nlogn) алгоритам а логаритамска функција расте много спорије од квадратне функције.

Треба напоменути и то да видимо да је број померања елемената често прилично већи код merge sort-a него код selection sort-а, али ово нас не треба довести до погрешног закључка. Selection sort је дизајниран да има мали број замена елемената јер он замењује елемент тек тада када утврди да он треба да се налази на неком месту тако што га пореди са свим осталим бројевима, што због тог поређења са осталим бројевима ипак чини алгоритам много спорији него merge алгоритам који ће исти елемент сортирати више пута у више поднизова различитих величина али ће учинити много мање поређења и итерација да би стигао до резултата.

# Уочени проблеми и ограничења

Скалабилност би био могући проблем у случају да постоји потреба да се програм искористи за улазне фајлове који имају више од 10 елемената. Ово је због тога што је графички интерфејс за визуелизацију осмишљен са 10 елемената као максимумом у виду, и он се прилагођава али само за тај број елемената у зависности од броја итерација. У случају да имамо велики број елемената поготово код selection sort-a коме би требао и велики број итерација визуелизација више не би радила и морало би се мислити о другим решењима. Додатне операције и простор који заузима чување података о сортирању током сортирања су незнатне са овим бројем елемената, али би могле бити проблем за веома велике количине података где би чак и merge sort био успорен због тог прикупљања велике количине података о свим итерацијама и операцијама извршеним, тако да је могуће да би биле потребне неке измене.