Table of Contents

[1. Introduction 2](#_Toc473494367)

[1.1.Problem Statement 2](#_Toc473494368)

[1.2.Aims and Objectives 2](#_Toc473494369)

[1.3. Distribution of Tasks 2](#_Toc473494370)

[2. Analysis 3](#_Toc473494371)

[2.1.Description of the Data Set 3](#_Toc473494372)

[2.2.Description of the System 4](#_Toc473494373)

[2.3.Functional requirements 4](#_Toc473494374)

[2.4.Non-functional requirements 4](#_Toc473494375)

[3. Design 5](#_Toc473494376)

[3.1. Sorting Algorithms (at least three different algorithms) 5](#_Toc473494377)

[3.2. Searching Algorithms (at least three different algorithms) 6](#_Toc473494378)

[3.3.Design issues 6](#_Toc473494379)

[3.3.1.Performance 6](#_Toc473494380)

[3.3.2.Robustness 6](#_Toc473494381)

[3.3.3.Error, Exception Handling & Fault Tolerance 6](#_Toc473494382)

[4. Implementation and Evaluation 7](#_Toc473494383)

[4.1.Tools and environment used 7](#_Toc473494384)

[4.2.Implementation challenges 7](#_Toc473494385)

[4.3. Sample Code of Main Pieces of Logic 7](#_Toc473494386)

[4.4. Sample Screenshots of Results 8](#_Toc473494387)

[4.5.Performance of Sorting Algorithms 8](#_Toc473494388)

[4.6.Performance of Searching Algorithms 8](#_Toc473494389)

[5. Discussion and Conclusion 9](#_Toc473494390)

[5.1.Achievements 9](#_Toc473494391)

[5.2. Challenges 9](#_Toc473494392)

# Introduction

## 1.1. Problem Statement

The current release (November 2016) of SwissProt contains nearly 553,000 sequence entries. A program was coded to extract data from the fasta format file, sort and search the data and output the desired results.

## 1.2. Aims and Objectives

1. To design a data structure(s) in order to facilitate data processing.
2. To write a command-line program to allow a user:

1. To retrieve and display the sequence of a query SwissProt ID. An example of a SwissProt ID is Q6GZX4 and its sequence is “MAFSA...”

2. To output to a FASTA format file a list of sequences with a minimum length specified by the user

3. To output to a CSV file a list of SwissProt ID and their organism (OS) sorted by organism then SwissProt ID

## 1.3. Distribution of Tasks

Tasks were shared equally.

# 2. Analysis

## 2.1. Description of the Data Set

The data set file is in FASTA format and contain 553231 entry conforming to the FASTA header for UniProtKB, as described below;

Each sequence

>db|UniqueIdentifier|EntryName ProteinName OS=OrganismName [GN=GeneName]PE=ProteinExistence SV=SequenceVersion

Where:

1. db is 'sp' for UniProtKB/Swiss-Prot and 'tr' for UniProtKB/TrEMBL.
2. UniqueIdentifier is the primary accession number of the UniProtKB entry.
3. EntryName is the entry name of the UniProtKB entry.
4. ProteinName is the recommended name of the UniProtKB entry as annotated in the RecName field. For UniProtKB/TrEMBL entries without a RecName field, the SubName field is used. In case of multiple SubNames, the first one is used. The 'precursor' attribute is excluded, 'Fragment' is included with the name if applicable.
5. OrganismName is the scientific name of the organism of the UniProtKB entry.
6. GeneName is the first gene name of the UniProtKB entry. If there is no gene name, OrderedLocusName or ORFname, the GN field is not listed.
7. ProteinExistence is the numerical value describing the evidence for the existence of the protein.
8. SequenceVersion is the version number of the sequence.

Example of sequence entry:

>sp|A5U6K4|MQO\_MYCTA Probable malate:quinone oxidoreductase OS=Mycobacterium tuberculosis (strain ATCC 25177 / H37Ra) GN=mqo PE=3 SV=1 MSDLARTDVVLIGAGIMSATLGVLLRRLEPNWSITLIERLDAVAAESSGPWNNAGTGHSA

LCEMNYTPEMPDGSIDITKAVRVNEQFQVTRQFWAYAAENGILTDVRSFLNPVPHVSFVH

GSRGVEYLRRRQKALAGNPLFAGTEFIESPDEFARRLPFMAAKRAFSEPVALNWAADGTD

VDFGALAKQLIGYCVQNGTTALFGHEVRNLSRQSDGSWTVTMCNRRTGEKRKLNTKFVFV

GAGGDTLPVLQKSGIKEVKGFAGFPIGGRFLRAGNPALTASHRAKVYGFPAPGAPPLGAL

HLDLRFVNGKSWLVFGPYAGWSPKFLKHGQISDLPRSIRPDNLLSVLGVGLTERRLLNYL

ISQLRLSEPERVSALREFAPSAIDSDWELTIAGQRVQVIRRDERNGGVLEFGTTVIGDAD

GSIAGLLGGSPGASTAVAIMLDVLQKCFANRYQSWLPTLKEMVPSLGVQLSNEPALFDEV

WSWSTKALKLGAA

## 2.2. Description of the System

The developed system should read the sequences from Fasta file and store each of them in an object which is stored in a node of a linked list. The user has the following options;

1. Search a sequence by inputting the id
2. Output to FASTA format file, list of sequences with a minimum length specified by user
3. Output to a CSV file a list of SwissProt ID and their organism (OS) sorted by organism then SwissProt ID

## 2.3. Functional requirements

1. Read each entry from the file and store in a linked list.

2. Output to fasta and csv format file.

3. Successfully sort array by SwissProt ID for binary search, by organism then SwissProt ID for outputting to csv file.

4. Validation on input should avoid system to crash.

5. Search through linked list for SwissProt ID.

7. Find length of each sequence and compare it to minimum length specified by user.

## 2.4. Non-functional requirements

1. Robustness

The program has the ability to cope with errors during execution through try and catch methods and cope with erroneous input through validation.

2. Computer Performance

Program uses the sorting and searching algorithms found to run the fastest of the three each we had respectively chosen.

3. Software testability

Easy to test the software, due to the separation of codes, for example different classes for sorting, searching, data structure and protein.

# 3. Design

To allow any processing to be done, the data has to be stored in a data structure first. The choice was choice was between

1. Array
2. Linked List

We choose to use both, combining their advantages. Linked list is a dynamic data structure, that is size does not have to be known before running the program. If an updated version of the Swiss-Prot becomes available, the program will be able to read all the entries, which would not be possible with an array.

However, linked list does not allow to jump to a specific id, which is useful in algorithms to sort and search, which is why, array was used in such situation to return faster results.

## 3.1. Sorting Algorithms (at least three different algorithms)

Sorting algorithm is needed to rearrange the data by Swiss-Prot ID, or by os the Swiss-Prot ID. The sorting algorithms below, were chosen based on the data structure and their time complexities as shown in Table 1.

1. Insertion Sort
2. Quicksort
3. Merge Sort

All the algorithms will be implemented and tested. They will also be provided to the user to choose in the menu.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Insertion Sort | Quicksort | Merge Sort |
| Worst Case | O(N2) | O(N2) | O(NLogN) |
| Average Case | O(N2) | O(NLogN) | O(NLogN) |
| Best Case | O(N) | O(NLogN) | O(NLogN) |

Table 1: Time complexities of sorting algorithms.

## 3.2. Searching Algorithms (at least three different algorithms)

Searching algorithms are required to fetch the sequence by specific Swiss-Prot ID or by length. The searching algorithms below, were chosen based on the data structure and their time complexities as shown in Table 2.

1. Sequential Search
2. Binary Search
3. Hash Table

All the algorithms will be implemented and tested. They will also be provided to the user to choose in the menu.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Sequential Search | Binary Search | Hash Table |
| Worst Case | O(n) | O(log n) | O(n) |
| Average Case | O(n) | O(log n) | O(1) |
| Best Case | O(1) | O(1) | O(1) |

Table 2: Time complexities of searching algorithms.

## 3.3. Design issues

### 3.3.1. Performance

The user will have to wait a couple of seconds before the menu is displayed as it take 12 to 13 seconds to read from the file and insert into the linked list.

Performance is critically affected when using insertion sort which highly inefficient when sorting a data set of 500,000 + entries.

### 3.3.2. Robustness

Robustness is the ability of a computer system to cope with errors during execution and cope with erroneous input.

Try Catch

Validation on input

### 3.3.3. Error, Exception Handling & Fault Tolerance

The program has been coded to handle such as file not found or erroneous input from user, with try catch block and validation on input, providing meaningful message to the user.

# 4. Implementation and Evaluation

## 4.1. Tools and environment used

Tools: Eclipse Mars.2 Release (4.5.2), IntelliJ IDEA 2016 1.2

Environment: Java jdk1.8.0\_73

## 4.2. Implementation challenges

|  |  |  |
| --- | --- | --- |
| # | Challenges | Solution |
| 1. | Extract the Swiss-Prot ID, os, and sequence from each sequence entry. | \*\*\*wakeel\*\*\* |
| 2. | To deal with linked list inability to jump. | Use of array |
| 3. | Dealing with the collisions that occurs while filling the hash table. | Use of Double Hashing in order to critically increase the performance of the Hash Search. |
| 4. | Merge sort on a linked list of size 553231 could not be done, as it created too many recursive call which resulted in stack overflow error. | Use of array and iteration |
| 5. | Some sequence do not have Gene Name | \*\*\*wakeel\*\*\* |

## 4.3. Sample Code of Main Pieces of Logic

1. **File Manipulation**

**while** (fr.hasNextLine()) {

sCurrentLine=fr.nextLine();

Matcher matcher2 = p2.matcher(sCurrentLine);

Matcher matcher3 = p3.matcher(sCurrentLine);

Matcher matcher4 = p4.matcher(sCurrentLine);

//id extract

**if** (sCurrentLine.charAt(0)=='>'){

**if** (firstRead){

sequence=(str);

protein.insertAtEnd(id,os,sequence);

str="";

idBuider.setLength(0);

}

**int** idCount=4;

**while** (sCurrentLine.charAt(idCount)!='|'){

idBuider.append(sCurrentLine.charAt(idCount));

idCount++;

}

id=idBuider.toString();

}

//os extract

**if** ((sCurrentLine.charAt(0) == '>')&& (matcher2.find())) {

start=matcher2.start()+3; **if** (matcher3.find()) {

end=matcher3.start()-1;

os=(sCurrentLine.substring(start, end));

}

**else** **if** (matcher4.find()) {

end=matcher4.start()-1;

os=(sCurrentLine.substring(start, end));

}

}

//reading sequence

**if** (sCurrentLine.charAt(0)!='>'){

firstRead=**true**;

str+=sCurrentLine;

**if** (!fr.hasNextLine()){

sequence=(str);

protein.insertAtEnd(id,os,sequence);

}

}

}

1. **Validation on input.**
2. **Sorting Merge Sort**

**public** **class** MergeSort {

**private** protein[] arr\_protein;

**private** String choice="";

**public** protein[] Sort(Linked\_List list,String choice){

**this**.choice=choice;

arr\_protein=list.ListToArray(list);

protein[] sorted\_list;

sorted\_list=Merge\_Sort(arr\_protein);

**return** sorted\_list;

}

**public** protein[] Merge\_Sort(protein[] list){

**if** (list.length<=1){

**return** list;

}

**int** length=list.length;

**int** m\_length=length/2;

protein[] half1;

protein[] half2;

**if** (length%2==0){

half1=**new** protein[m\_length];

half2=**new** protein[m\_length];

}

**else** {

half1=**new** protein[m\_length];

half2=**new** protein[m\_length+1];

}

**for** (**int** i=0;i<m\_length;i++){

half1[i]=list[i];

}

**int** x=m\_length;

**for** (**int** i=0;i<half2.length;i++){

half2[i]=list[x];

++x;

}

protein[] list1=Merge\_Sort(half1);

protein[] list2=Merge\_Sort(half2);

protein[] merged\_list=MergeList(list1,list2);

**return** merged\_list;

}

**public** protein[] MergeList(protein[] list1,protein[] list2){

**int** size=list1.length+list2.length;

protein[] result=**new** protein[size];

String data1="";

String data2="";

**int** i=0;

**int** j=0;

**int** k=0;

**while** ((i<list1.length)&&(j<list2.length)){

**if** (choice=="osid") {

data1 = list1[i].getOsID();

data2 = list2[j].getOsID();

}

**else** {

data1=list1[i].getSpid();

data2=list2[j].getSpid();

}

**if** (data1.compareTo(data2)<0){

result[k]=(list1[i]);

i++;

}

**else** {

result[k]=(list2[j]);

j++;

}

k++;

}

**for** (**int** z=i;z<list1.length;z++){

result[k]=list1[z];

k++;

}

**for** (**int** z=j;z<list2.length;z++){

result[k]=list2[z];

k++;

}

**return** result;

}

}

1. **Searching - Binary Search**

**public** **void** BinarySearch(String spid,protein[] LinkArray) { //searching

**final** **long** startTime = System.*nanoTime*();

**int** low = 0;

**int** high = LinkArray.length;

**while**(low<=high){

**int** mid = low+(high-low)/2;

**int** decisionParameter = (LinkArray[mid].getSpid()).compareTo(spid);

**if**(LinkArray[mid].getSpid().equals(spid)){

System.***out***.println("Id Found");

LinkArray[mid].displayNode();

System.***out***.println("");

**break**;

}

**else** **if** (decisionParameter < 0){//Value is higher than mid

low = mid+1;

}

**else**{//Value is lower than mid

high = mid-1;

}

}

**final** **long** duration = System.*nanoTime*() - startTime;

**long** millis= TimeUnit.***MILLISECONDS***.convert(duration,TimeUnit.***NANOSECONDS***);

System.***out***.println("Duration: "+millis+"ms");

}

1. **Output list of sequence to fasta format file**

**public** **void** outputFasta(Linked\_List protein, **int** mlen){

**try**{

Formatter outfile = **new** Formatter("Sequence.fasta");

Node current = protein.getHead();

outfile.format("Minimum length of sequence: %d\nSpid \t Sequence\n",mlen);

**while**(current != **null**){

**if**(current.getProtein().getSequence().length()>=mlen){

outfile.format("%s \t %s\nNumber of char: %d\n",current.getProtein().getSpid(),current.getProtein().getSequence(),current.getProtein().getSequence().length());

current = current.getNext();

}

**else**{

current = current.getNext();

}

}

outfile.close();

**try** {

rs.*getRuntime*().exec("cmd /c start notepad++ C:\\Users\\sandooyea\\yr2\\lab\\Sequence.fasta");

}

**catch** (Throwable e){

System.***out***.println("Error");

}

}

**catch**(FileNotFoundException fnfe){

System.***out***.println("File Not Found");

}

**catch**(SecurityException se){

System.***out***.println("No permission");

}

}

1. **Output list of Swiss-Prot ID and os sorted by os then Swiss-Prot ID.**

**public** **void** writeCSV(protein[] list){

String File="Sorted.csv";

**try** {

FileWriter fileWriter=**new** FileWriter(File);

**for** (**int** i=0;i<list.length;i++){

fileWriter.append(list[i].getOs());

fileWriter.append(',');

fileWriter.append(list[i].getSpid());

fileWriter.append('\n');

}

fileWriter.close();

**try** {

rs.*getRuntime*().exec("cmd /c start notepad++ C:\\Users\\sandooyea\\yr2\\lab\\Sorted.csv");

}

**catch** (Throwable e){

System.***out***.println("Error");

}

}

**catch** (Exception e){

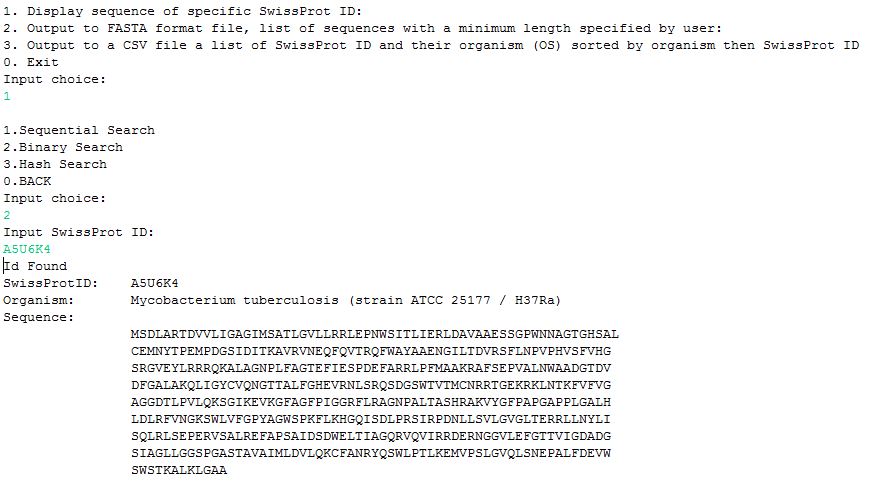
System.***out***.println("Error");

}

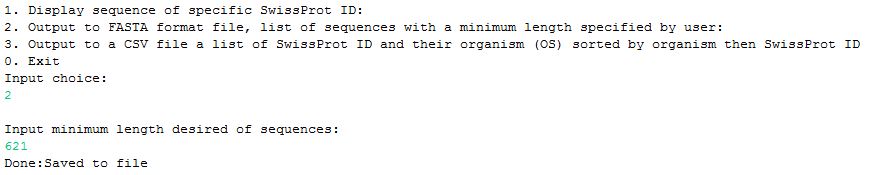
}

## 4.4. Sample Screenshots of Results

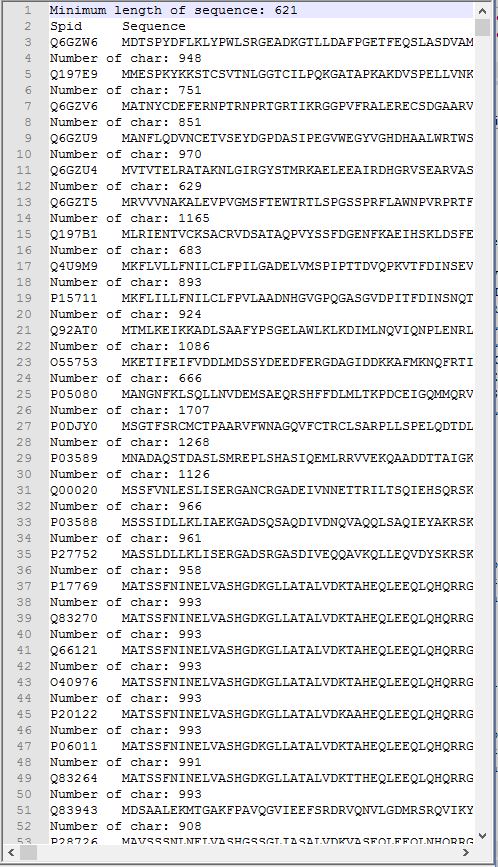
1. User input a Swiss-Prot ID, corresponding sequence is displayed



1. Output list of sequence to fasta format file



Fasta format file is opened and result correspond to what was desired. Number of characters was displayed to make testing easier.



1. Output list of Swiss-Prot ID and os sorted by os then Swiss-Prot ID.

## 4.5. Performance of Sorting Algorithms

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Insertion Sort** | **Quicksort** | **Merge Sort** |
| **Time 1(ms)** | 3903152 | 3014 | 2367 |
| **Time 2(ms)** | 3903140 | 3042 | 2298 |
| **Time 3(ms)** | 3903130 | 3263 | 2315 |
| **Time 4(ms)** | 3903192 | 2932 | 2250 |
| **Time 5(ms)** | 3903182 | 2933 | 2547 |
| **Average(ms)** | **3903159** | **3036** | **2355** |

Table 3: Performance in **milliseconds** of sorting algorithms

## 4.6. Performance of Searching Algorithms

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Sequential** | **Binary Search** | **Hash Table** |
| **Time 1(ms)** | 33 | 1 | 1 |
| **Time 2(ms)** | 30 | 0 | 1 |
| **Time 3(ms)** | 25 | 1 | 1 |
| **Time 4(ms)** | 23 | 1 | 1 |
| **Time 5(ms)** | 23 | 0 | 1 |
| **Average(ms)** | **27** | **1** | **1** |

Table 4: Performance in **milliseconds** of searching algorithms

# 5. Discussion and Conclusion

## 5.1. Achievements

1. Ability to sort and search data.

2. Proper validation on input.

3.Flexible Data Structure.

## 5.2. Challenges

1. To fully optimize the code

2. To use only linked list as data structure.