Class Documentation

Lab2 Class Reference

Public Member Functions

- **Lab2** ()=default
- **Lab2** (std::string namefile)
- void **heapify** (size_t n, size_t i)
- void **heapSort** ()
- std::vector< int > **LinearSearch** (std::string key)
- int **getLowerBound** (std::string key)
- int **getUpperBound** (std::string key)
- std::vector< int > **binarySearch** (std::string key)
- **~Lab2** ()=default

Public Attributes

- std::multimap< std::string, Lab2::Elements > **dataMap**
- std::string **filename**

Friends

- bool **operator<** (const Elements &c1, const Elements &c2)
- bool **operator>=** (const Elements &c1, const Elements &c2)
- bool **operator<=** (const Elements &c1, const Elements &c2)

Detailed Description

Definition at line **7** of file **Source.cpp**.

Constructor & Destructor Documentation

Lab2::Lab2 () `[default]`

Lab2::Lab2 (std::string *namefile*)

Definition at line **174** of file **Source.cpp**.

Lab2::~Lab2 () `[default]`

Member Function Documentation

std::vector< int > Lab2::binarySearch (std::string *key*) `[inline]`

Definition at line **158** of file **Source.cpp**.

int Lab2::getLowerBound (std::string *key*) `[inline]`

Definition at line **116** of file **Source.cpp**.

int Lab2::getUpperBound (std::string *key*) `[inline]`

Definition at line **137** of file **Source.cpp**.

void Lab2::heapify (size_t *n*, size_t *i*) `[inline]`

Definition at line **78** of file **Source.cpp**.

void Lab2::heapSort () `[inline]`

Definition at line **95** of file **Source.cpp**.

std::vector< int > Lab2::LinearSearch (std::string *key*) `[inline]`

Definition at line **106** of file **Source.cpp**.

Friends And Related Function Documentation

bool operator< (const Elements & *c1*, const Elements & *c2*) `[friend]`

Definition at line **51** of file **Source.cpp**.

bool operator<= (const Elements & *c1*, const Elements & *c2*) `[friend]`

Definition at line **64** of file **Source.cpp**.

bool operator>= (const Elements & *c1*, const Elements & *c2*) `[friend]`

Definition at line **57** of file **Source.cpp**.

Member Data Documentation

std::multimap<std::string, Lab2::Elements> Lab2::dataMap

Definition at line **74** of file **Source.cpp**.

std::string Lab2::filename

Definition at line **75** of file **Source.cpp**.

The documentation for this class was generated from the following file:
Source.cpp

File Documentation

Source.cpp File Reference

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <chrono>
#include <map>
```

Classes

class Lab2Functions

- int **main** ()

Function Documentation

int main ()

Definition at line **249** of file **Source.cpp**.

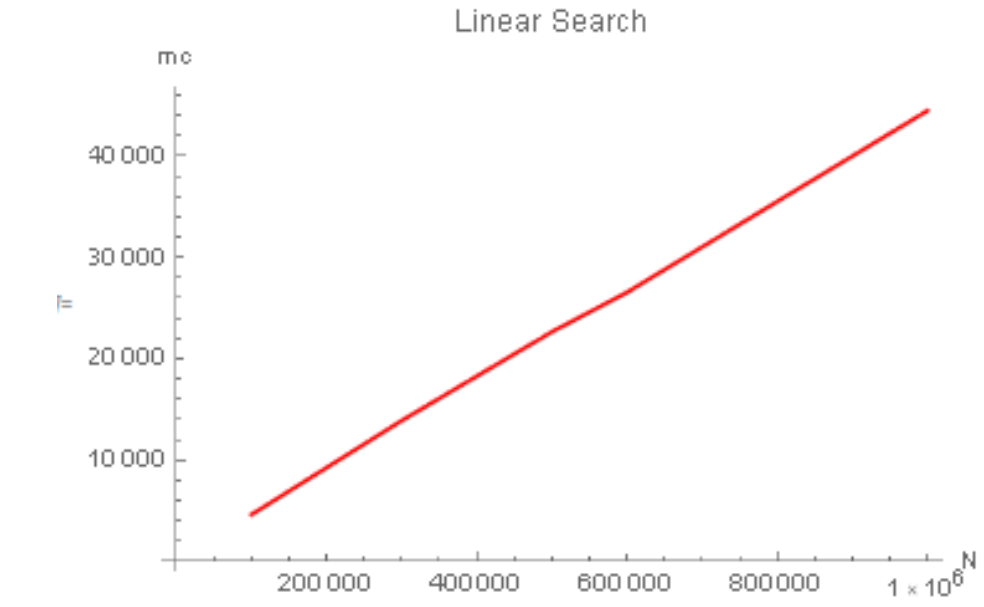
Source.cpp

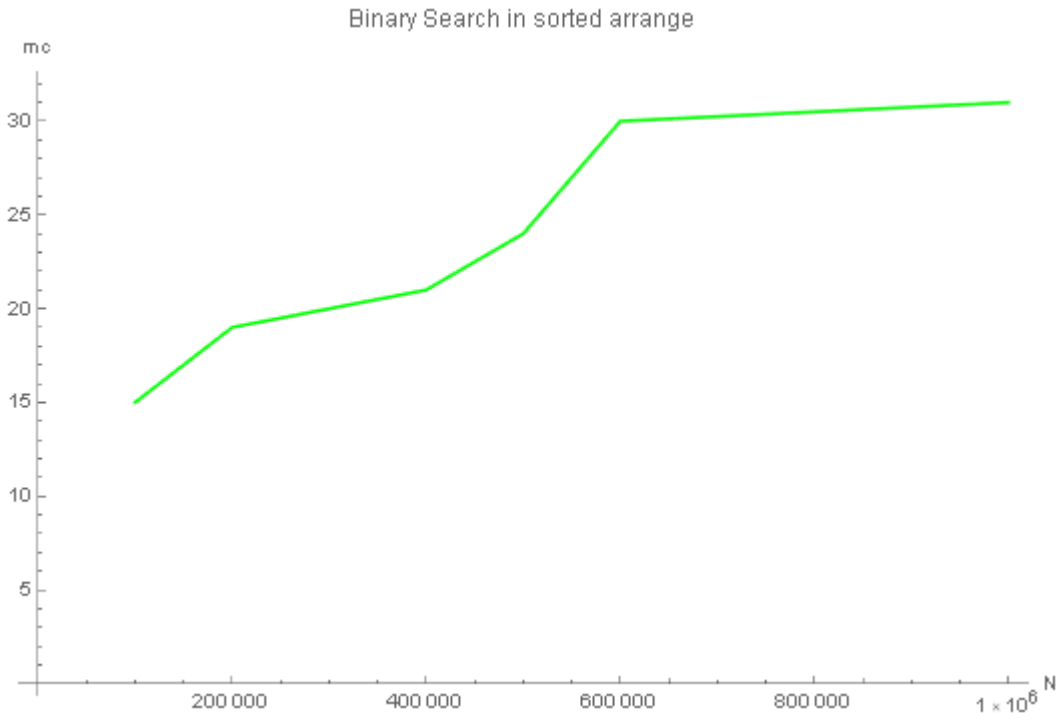
```
Go to the documentation of this file.00001 #include <iostream>
00002 #include <vector>
00003 #include <string>
00004 #include <fstream>
00005 #include <chrono>
00006 #include <map>
00007 class Lab2 {
00008     struct Elements {
00009         std::string date;
00010         size_t win;
00011         size_t number;
00012         size_t cost;
00013         friend bool operator== (const Elements& c1, const Elements& c2) {
00014             if (c1.date == c2.date) {
00015                 if (c1.win == c2.win) {
00016                     if (c1.number == c1.number) {
00017                         return true;
00018                     }
00019                 }
00020             }
00021             return false;
00022         }
00023         friend bool operator >(const Elements& c1, const Elements& c2) {
00024             if (std::strcmp(c1.date.c_str(), c2.date.c_str()) == -1) {
00025                 return false;
00026             }
00027             if (std::strcmp(c1.date.c_str(), c2.date.c_str()) == 1) {
00028                 return true;
00029             }
00030             if (std::strcmp(c1.date.c_str(), c2.date.c_str()) == 0) {
00031                 if (c1.win < c2.win) {
00032                     return true;
00033                 }
00034                 if (c1.win > c2.win) {
00035                     return false;
00036                 }
00037                 if (c1.win == c2.win) {
00038                     if (c1.number > c2.number) {
00039                         return true;
00040                     }
00041                     if (c1.number < c2.number) {
00042                         return false;
00043                     }
00044                     if (c1.number == c2.number) {
00045                         return false;
00046                     }
00047                 }
00048             }
00049         }
00050     };
00051     friend bool operator <(const Elements& c1, const Elements& c2) {
00052         if (c1 == c2) {
00053             return false;
00054         }
00055         return !(c1 > c2);
00056     }
00057     friend bool operator >=(const Elements& c1, const Elements& c2) {
00058         if (c1 == c2) {
00059             return true;
00060         }
00061         return c1 > c2;
00062 
00063     }
00064     friend bool operator <=(const Elements& c1, const Elements& c2) {
00065         if (c1 == c2) {
00066             return true;
00067         }
00068         return c1 < c2;
00069     }
00070 
00071     std::vector<Lab2::Elements> data;
00072 
00073 public:
```

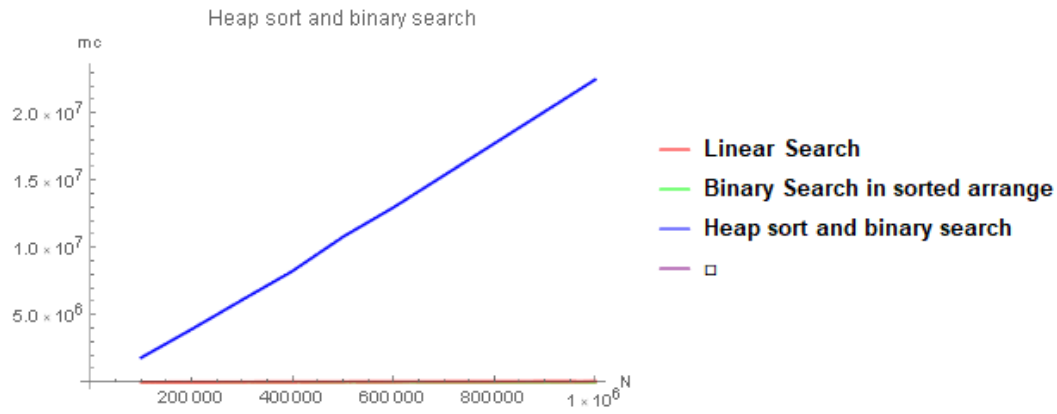
```
00074     std::multimap<std::string, Lab2::Elements> dataMap;
00075     std::string filename;
00076     Lab2() = default;
00077     Lab2(std::string namefile);
00078     void heapify(size_t n, size_t i) {
00079 
00080         size_t largest = i;
00081         size_t l = 2 * i + 1;
00082         size_t r = 2 * i + 2;
00083 
00084         if (l < n && data[l] > data[largest])
00085             largest = l;
00086 
00087         if (r < n && data[r] > data[largest])
00088             largest = r;
00089         if (largest != i)
00090         {
00091             std::swap(data[i], data[largest]);
00092             heapify(n, largest);
00093         }
00094     }
00095     void heapSort() {
00096 
00097         for (int i = data.size() / 2 - 1; i >= 0; i--) {
00098             heapify(data.size(), i);
00099         }
00100         for (int i = data.size() - 1; i >= 0; i--)
00101         {
00102             std::swap(data[0], data[i]);
00103             heapify(i, 0);
00104         }
00105     }
00106     std::vector<int> LinearSearch(std::string key) {
00107         std::vector<int> res;
00108         for (size_t i = 0; i != data.size(); ++i) {
00109             if (data[i].date == key) {
00110                 res.push_back(i);
00111             }
00112         }
00113         return res;
00114     }
00115 
00116     int getLowerBound(std::string key) {
00117         int mid = 0, left = 0, right = data.size();
00118         while (1)
00119         {
00120             mid = (left + right) / 2;
00121 
00122             if (mid < 0 || mid >= data.size())
00123                 return -1;
00124 
00125             if (key <= data[mid].date)
00126                 right = mid - 1;
00127             else if (key > data[mid].date && mid + 1 < data.size() && key == data[mid
+ 1].date)
00128                 return mid+1;
00129             else if (key > data[mid].date)
00130                 left = mid + 1;
00131 
00132             if (left > right)
00133                 return -1;
00134         }
00135     }
00136 
00137     int getUpperBound(std::string key) {
00138         int mid = 0, left = 0, right = data.size();
00139         while (1)
00140         {
00141             mid = (left + right) / 2;
00142 
00143             if (mid < 0 || mid >= data.size())
00144                 return -1;
00145 
00146             if (key < data[mid].date && mid - 1 >= 0 && key == data[mid - 1].date)
00147                 return mid-1;
00148             else if (key < data[mid].date)
00149                 right = mid - 1;
```

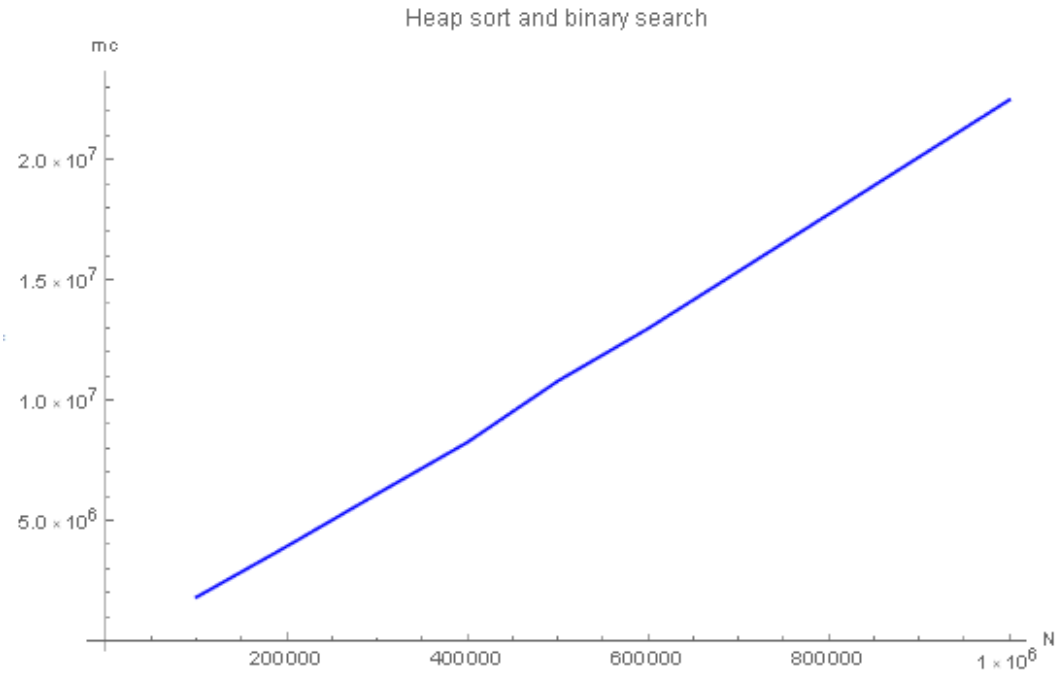
```
00150             else if (key >= data[mid].date)
00151                 left = mid + 1;
00152
00153             if (left > right)
00154                 return -1;
00155         }
00156     }
00157
00158     std::vector<int> binarySearch(std::string key) {
00159         int left = getLowerBound(key);
00160         int right = getUpperBound(key);
00161         std::vector<int> res;
00162
00163         if (left == -1 || right == -1)
00164             return res;
00165
00166         for (size_t i = left; i <= right; ++i) {
00167             res.push_back(i);
00168         }
00169
00170         return res;
00171     }
00172     ~Lab2() = default;
00173 };
00174 Lab2::Lab2(std::string namefile)
00175 {
00176     std::ifstream inf(namefile);
00177     filename = namefile;
00178
00179     if (!inf.is_open())
00180     {
00181         std::cerr << "The file could not be opened for reading!\n";
00182     }
00183     std::string s;
00184     while (std::getline(inf, s)) {
00185         Lab2::Elements obj;
00186         size_t i = 0;
00187         bool flag = 1;
00188         std::string num;
00189         while (flag) {
00190             if (s[i] == ' ') {
00191                 flag = 0;
00192                 i++;
00193                 break;
00194             }
00195             else {
00196                 num += s[i];
00197                 i++;
00198             }
00199         }
00200         obj.number = std::stoi(num.c_str());
00201         std::string cost;
00202         flag = 1;
00203         while (flag) {
00204             if (s[i] == ' ') {
00205                 flag = 0;
00206                 i++;
00207                 break;
00208             }
00209             else {
00210                 cost += s[i];
00211                 i++;
00212             }
00213         }
00214         obj.cost = std::stoi(cost.c_str());
00215         std::string date;
00216         flag = 1;
00217         while (flag) {
00218             if (s[i] == ' ') {
00219                 flag = 0;
00220                 i++;
00221                 break;
00222             }
00223             else {
00224                 date += s[i];
00225                 i++;
00226             }
```

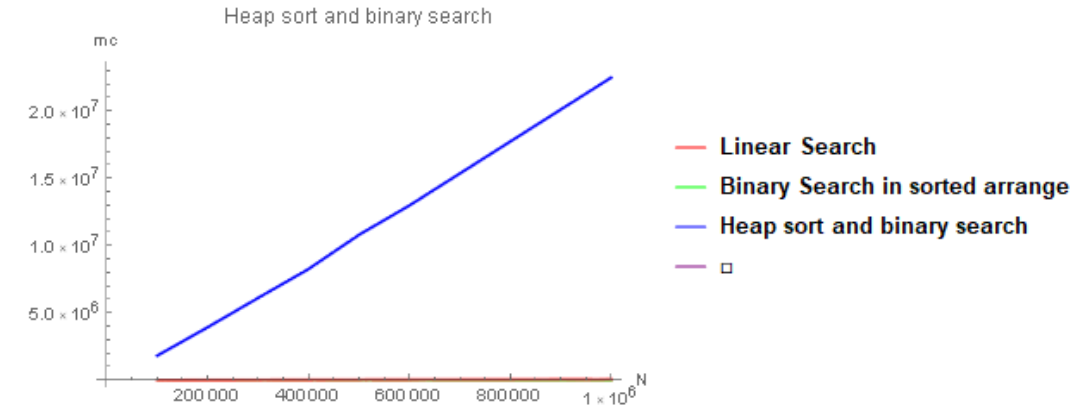
```
00227         }
00228         obj.date = date;
00229         flag = 1;
00230         std::string sum;
00231         while (flag) {
00232             if (i == s.size()) {
00233                 flag = 0;
00234                 break;
00235             }
00236             else {
00237                 sum += s[i];
00238                 i++;
00239             }
00240         }
00241
00242         obj.win = std::stoi(sum.c str());
00243         data.push_back(obj);
00244         dataMap.insert({ date,obj });
00245     }
00246     inf.close();
00247 }
00248
00249 int main() {
00250     std::string m[7] = {
"100000.txt","200000.txt","300000.txt","400000.txt","500000.txt","600000.txt","1000000
.txt" };
00251     std::string m1[7] = {
"heapSort100000.txt","heapSort200000.txt","heapSort300000.txt","heapSort400000.txt","h
eapSort500000.txt","heapSort600000.txt","heapSort1000000.txt" };
00252     for (int i = 0; i < 7; ++i) {
00253         Lab2 v(m[i%7]);
00254         auto begin = std::chrono::steady_clock::now();
00255         v.LinearSearch("1999/04/14");
00256         auto end = std::chrono::steady_clock::now();
00257         auto elapsed_ms =
std::chrono::duration_cast<std::chrono::microseconds>(end - begin);
00258         std::cout << "The time of linear search " << m[i % 7] << ": " <<
elapsed_ms.count() << " mc\n";
00259         Lab2 v1(m1[i%7]);
00260         begin = std::chrono::steady_clock::now();
00261         v1.binarySearch("1999/04/14");
00262         end = std::chrono::steady_clock::now();
00263         elapsed_ms = std::chrono::duration_cast<std::chrono::microseconds>(end -
begin);
00264         std::cout << "The time of binary search in sorted arrange " << m[i % 7] <<
": " << elapsed_ms.count() << " mc\n";
00265         begin = std::chrono::steady_clock::now();
00266         v.heapSort();
00267         v.binarySearch("1999/04/14");
00268         end = std::chrono::steady_clock::now();
00269         elapsed_ms = std::chrono::duration_cast<std::chrono::microseconds>(end -
begin);
00270         std::cout << "The time of binary search in unsorted arrange " << m[i % 7]
<< ": " << elapsed_ms.count() << " mc\n";
00271         Lab2 v2(m[i % 7]);
00272         begin = std::chrono::steady_clock::now();
00273         v2.dataMap.equal_range("1999/04/14");
00274         end = std::chrono::steady_clock::now();
00275         elapsed_ms = std::chrono::duration_cast<std::chrono::microseconds>(end -
begin);
00276         std::cout << "The time of multimap " << m[i % 7] << ": " << elapsed_ms.count()
<< " mc\n";
00277     }
00278 }
```

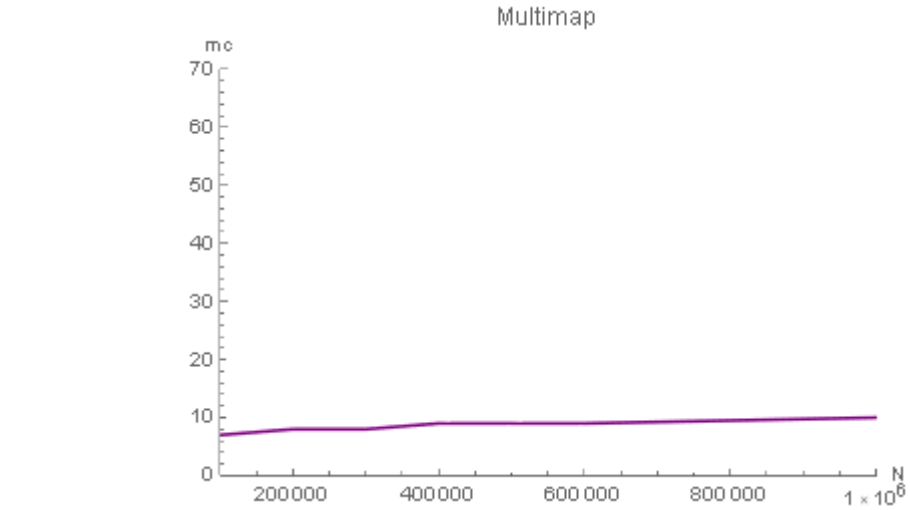

Linear Search
mc
40 000
30 000
20 000
10 000
T
200 000
400000
600 000
800000
1 × 10^6
N


Binary Search in sorted arrange
mc
30
25
20
15
10
5
200 000
400 000
600 000
800 000
1 × 10^6
N

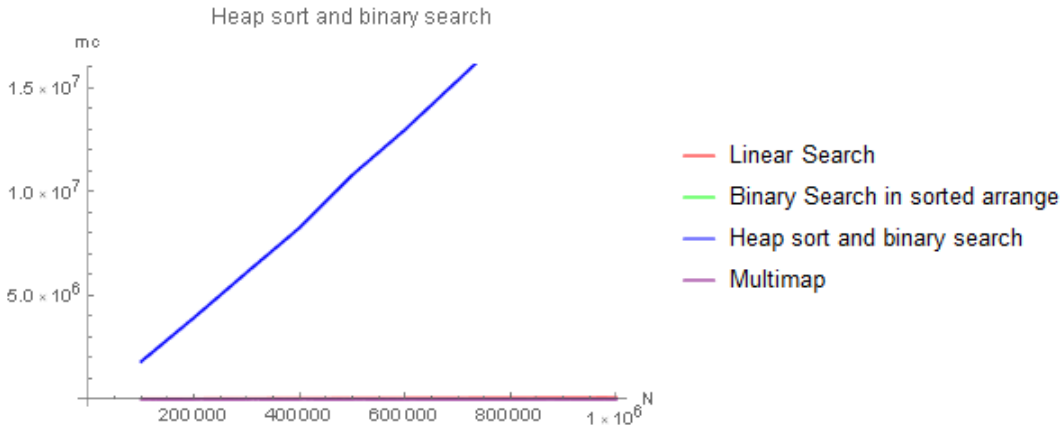

Heap sort and binary search
mc
2.0 × 10^7
1.5 × 10^7
1.0 × 10^7
5.0 × 10^6
200 000
400000
600 000
800000
1 × 10^6
N
Linear Search
Binary Search in sorted arrange
Heap sort and binary search







Таким образом, поиск по ключу осуществляется быстрее всех остальных способов из п. 3 задания.



Ссылка на репозиторий: https://github.com/AssRoar/HSE_MethProg