Class Documentation

Lab1 Class Reference

Public Member Functions

- **Lab1** ()=default
- **Lab1** (std::string namefile)
- void **bubbleSort** ()
- void **selectionSort** ()
- void **heapify** (size_t n, size_t i)
- void **heapSort** ()
- void **output** ()
- **~Lab1** ()=default

Public Attributes

- 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 **6** of file **Source.cpp**.

Constructor & Destructor Documentation

Lab1::Lab1 () `[default]`

Lab1::Lab1 (std::string *namefile*)

Definition at line **83** of file **Source.cpp**.

Lab1::~Lab1 () `[default]`

Member Function Documentation

void Lab1::bubbleSort ()

Definition at line **156** of file **Source.cpp**.

void Lab1::heapify (size_t *n*, size_t *i*)

Definition at line **190** of file **Source.cpp**.

void Lab1::heapSort ()

Definition at line **207** of file **Source.cpp**.

void Lab1::output ()

Definition at line **218** of file **Source.cpp**.

void Lab1::selectionSort ()

Definition at line **176** of file **Source.cpp**.

Friends And Related Function Documentation

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

Definition at line **50** of file **Source.cpp**.

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

Definition at line **63** of file **Source.cpp**.

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

Definition at line **56** of file **Source.cpp**.

Member Data Documentation

std::string Lab1::filename

Definition at line **73** 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>
```

Classes

class Lab1

Functions

- int **main** ()

Function Documentation

int main ()

Definition at line **229** of file **Source.cpp**.

Source.cpp

```
00001 #include <iostream>
00002 #include <vector>
00003 #include <string>
00004 #include <fstream>
00005 #include <chrono>
00006 class Lab1 {
00007     struct Elements {
00008     std::string date;
00009     size_t win;
00010     size_t number;
00011     size_t cost;
00012     friend bool operator== (const Elements& c1, const Elements& c2) {
00013         if (c1.date == c2.date) {
00014             if (c1.win == c2.win) {
00015                 if (c1.number == c1.number) {
00016                     return true;
00017                 }
00018             }
00019         }
00020         return false;
00021     }
00022     friend bool operator >(const Elements& c1, const Elements& c2) {
00023         if (std::strcmp(c1.date.c_str(), c2.date.c_str()) == -1) {
00024             return false;
00025         }
00026         if (std::strcmp(c1.date.c_str(), c2.date.c_str()) == 1) {
00027             return true;
00028         }
00029         if (std::strcmp(c1.date.c_str(), c2.date.c_str()) == 0) {
00030             if (c1.win < c2.win) {
00031                 return true;
00032             }
00033             if (c1.win > c2.win) {
00034                 return false;
00035             }
00036             if (c1.win == c2.win) {
00037                 if (c1.number > c2.number) {
00038                     return true;
00039                 }
00040                 if (c1.number < c2.number) {
00041                     return false;
00042                 }
00043                 if (c1.number == c2.number) {
00044                     return false;
00045                 }
00046             }
00047         }
00048     }
00049     };
00050     friend bool operator <(const Elements& c1, const Elements& c2) {
00051         if (c1 == c2) {
00052             return false;
00053         }
00054         return !(c1 > c2);
00055     }
00056     friend bool operator >=(const Elements& c1, const Elements& c2) {
00057         if (c1 == c2) {
00058             return true;
00059         }
00060         return c1 > c2;
00061 
00062     }
00063     friend bool operator <=(const Elements& c1, const Elements& c2) {
00064         if (c1==c2) {
00065             return true;
00066         }
00067         return c1 < c2;
00068     }
00069 
00070     std::vector<Lab1::Elements> data;
00071 
00072 public:
00073     std::string filename;
```

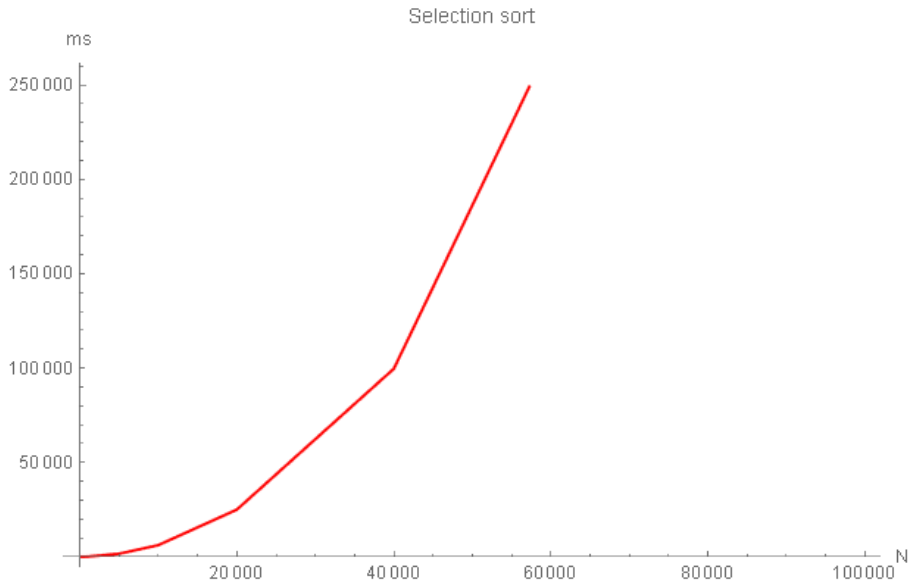
```
00074     Lab1() = default;
00075     Lab1(std::string namefile);
00076     void bubbleSort();
00077     void selectionSort();
00078     void heapify(size_t n, size_t i);
00079     void heapSort();
00080     void output();
00081     ~Lab1() = default;
00082 };
00083 Lab1::Lab1(std::string namefile)
00084 {
00085     std::ifstream inf(namefile);
00086     filename = namefile;
00087 
00088     if (!inf.is_open())
00089     {
00090         std::cerr << "The file could not be opened for reading!\n";
00091     }
00092     std::string s;
00093     while (std::getline(inf, s)) {
00094         Lab1::Elements obj;
00095         size t i = 0;
00096         bool flag = 1;
00097         std::string num;
00098         while (flag) {
00099             if (s[i] == ' ') {
00100                 flag = 0;
00101                 i++;
00102                 break;
00103             }
00104             else {
00105                 num += s[i];
00106                 i++;
00107             }
00108         }
00109         obj.number = std::stoi(num.c_str());
00110         std::string cost;
00111         flag = 1;
00112         while (flag) {
00113             if (s[i] == ' ') {
00114                 flag = 0;
00115                 i++;
00116                 break;
00117             }
00118             else {
00119                 cost += s[i];
00120                 i++;
00121             }
00122         }
00123         obj.cost = std::stoi(cost.c_str());
00124         std::string date;
00125         flag = 1;
00126         while (flag) {
00127             if (s[i] == ' ') {
00128                 flag = 0;
00129                 i++;
00130                 break;
00131             }
00132             else {
00133                 date += s[i];
00134                 i++;
00135             }
00136         }
00137         obj.date = date;
00138         flag = 1;
00139         std::string sum;
00140         while (flag) {
00141             if (i==s.size()) {
00142                 flag = 0;
00143                 break;
00144             }
00145             else {
00146                 sum += s[i];
00147                 i++;
00148             }
00149         }
00150 
```

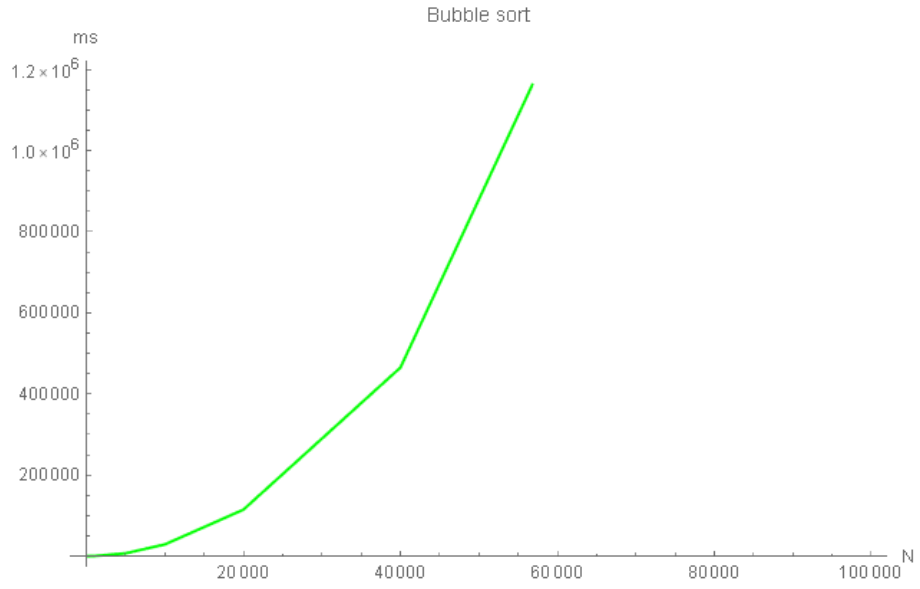
```
00151         obj.win = std::stoi(sum.c_str());
00152         data.push_back(obj);
00153     }
00154     inf.close();
00155 }
00156 void Lab1::bubbleSort()
00157 {
00158     size_t len = data.size();
00159     while (len--)
00160     {
00161         bool swapped = false;
00162 
00163         for (size_t i = 0; i < len; i++)
00164         {
00165             if (data[i] > data[i + 1])
00166             {
00167                 std::swap(data[i], data[i + 1]);
00168                 swapped = true;
00169             }
00170         }
00171 
00172         if (swapped == false)
00173             break;
00174     }
00175 }
00176 void Lab1::selectionSort() {
00177     size_t i, j, min_idx;
00178     for (i = 0; i < data.size() - 1; i++)
00179     {
00180         min_idx = i;
00181         for (j = i + 1; j < data.size(); j++)
00182         {
00183             if (data[j] < data[min_idx])
00184                 min_idx = j;
00185         }
00186         if (min_idx != i)
00187             std::swap(data[min_idx], data[i]);
00188     }
00189 }
00190 void Lab1::heapify(size_t n,size_t i) {
00191 
00192     size_t largest = i;
00193     size_t l = 2 * i + 1;
00194     size_t r = 2 * i + 2;
00195 
00196     if (l < n && data[l] > data[largest])
00197         largest = l;
00198 
00199     if (r < n && data[r] > data[largest])
00200         largest = r;
00201     if (largest != i)
00202     {
00203         std::swap(data[i], data[largest]);
00204         heapify( n, largest);
00205     }
00206 }
00207 void Lab1::heapSort() {
00208 
00209     for (int i = data.size() / 2 - 1; i >= 0; i--) {
00210         heapify(data.size(), i);
00211     }
00212     for (int i = data.size() - 1; i >= 0; i--)
00213     {
00214         std::swap(data[0], data[i]);
00215         heapify(i, 0);
00216     }
00217 }
00218 void Lab1::output()
00219 {
00220     std::ofstream out;
00221     out.open("SORTED MODE"+filename);
00222     if (out.is_open())
00223     {
00224         for (auto& it : data) {
00225             out << it.number << " " << it.cost << " " << it.date << " " << it.win
<< "\n";
00226         }
```

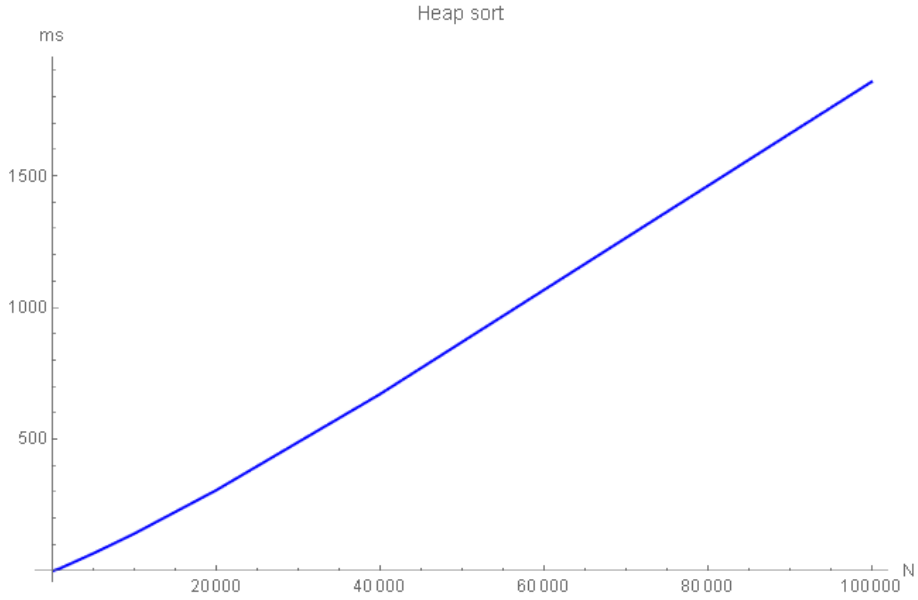
```
00227     }
00228 }
00229 int main() {
00230     std::string p = "100.txt";
00231     Lab1 v(p);
00232     v.filename = "heapSort"+v.filename;
00233     auto begin = std::chrono::steady_clock::now();
00234     v.heapSort();
00235     auto end = std::chrono::steady_clock::now();
00236     v.output();
00237     auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00238     std::cout << "The time heapSort 100: " << elapsed_ms.count() << " ms\n";
00239     p = "1000.txt";
00240     Lab1 v1(p);
00241     v1.filename = "heapSort" + v1.filename;
00242     begin = std::chrono::steady_clock::now();
00243     v1.heapSort();
00244     end = std::chrono::steady_clock::now();
00245     v1.output();
00246     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00247     std::cout << "The time heapSort 1000: " << elapsed_ms.count() << " ms\n";
00248     p = "5000.txt";
00249     Lab1 v2(p);
00250     v2.filename = "heapSort" + v2.filename;
00251     begin = std::chrono::steady_clock::now();
00252     v2.heapSort();
00253     end = std::chrono::steady_clock::now();
00254     v2.output();
00255     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00256     std::cout << "The time heapSort 5000: " << elapsed_ms.count() << " ms\n";
00257     p = "10000.txt";
00258     Lab1 v3(p);
00259     v3.filename = "heapSort" + v3.filename;
00260     begin = std::chrono::steady_clock::now();
00261     v3.heapSort();
00262     end = std::chrono::steady_clock::now();
00263     v3.output();
00264     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00265     std::cout << "The time heapSort 10000: " << elapsed_ms.count() << " ms\n";
00266     p = "20000.txt";
00267     Lab1 v4(p);
00268     v4.filename = "heapSort" + v4.filename;
00269     begin = std::chrono::steady_clock::now();
00270     v4.heapSort();
00271     end = std::chrono::steady_clock::now();
00272     v4.output();
00273     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00274     std::cout << "The time heapSort 20000: " << elapsed_ms.count() << " ms\n";
00275     p = "40000.txt";
00276     Lab1 v5(p);
00277     v5.filename = "heapSort" + v5.filename;
00278     begin = std::chrono::steady_clock::now();
00279     v5.heapSort();
00280     end = std::chrono::steady_clock::now();
00281     v5.output();
00282     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00283     std::cout << "The time heapSort 40000: " << elapsed_ms.count() << " ms\n";
00284     p = "100000.txt";
00285     Lab1 v6(p);
00286     v6.filename = "heapSort" + v6.filename;
00287     begin = std::chrono::steady_clock::now();
00288     v6.heapSort();
00289     end = std::chrono::steady_clock::now();
00290     v6.output();
00291     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00292     std::cout << "The time heapSort 100000: " << elapsed_ms.count() << " ms\n";
00293
00294     std::cout << "\n";
00295     p = "100.txt";
00296     Lab1 h(p);
```

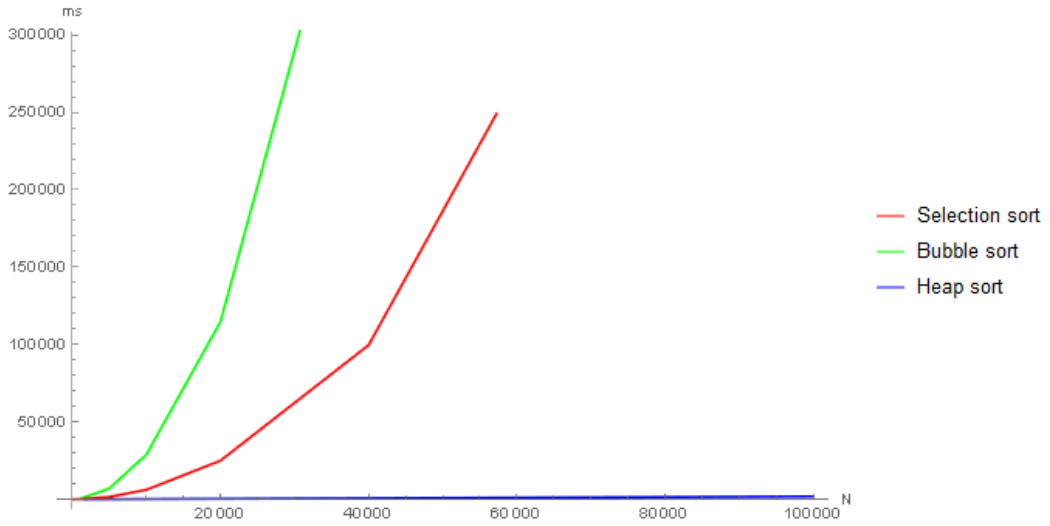
```
00297     h.filename = "bubbleSort" + h.filename;
00298     begin = std::chrono::steady_clock::now();
00299     h.bubbleSort();
00300     end = std::chrono::steady_clock::now();
00301     h.output();
00302     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00303     std::cout << "The time bubbleSort 100: " << elapsed_ms.count() << " ms\n";
00304     p = "1000.txt";
00305     Lab1 h1(p);
00306     h1.filename = "bubbleSort" + h1.filename;
00307     begin = std::chrono::steady_clock::now();
00308     h1.bubbleSort();
00309     end = std::chrono::steady_clock::now();
00310     h1.output();
00311     elapsed ms = std::chrono::duration cast<std::chrono::milliseconds>(end -
begin);
00312     std::cout << "The time bubbleSort 1000: " << elapsed_ms.count() << " ms\n";
00313     p = "5000.txt";
00314     Lab1 h2(p);
00315     h2.filename = "bubbleSort" + h2.filename;
00316     begin = std::chrono::steady clock::now();
00317     h2.bubbleSort();
00318     end = std::chrono::steady_clock::now();
00319     h2.output();
00320     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00321     std::cout << "The time bubbleSort 5000: " << elapsed_ms.count() << " ms\n";
00322     p = "10000.txt";
00323     Lab1 h3(p);
00324     h3.filename = "bubbleSort" + h3.filename;
00325     begin = std::chrono::steady_clock::now();
00326     h3.bubbleSort();
00327     end = std::chrono::steady_clock::now();
00328     h3.output();
00329     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00330     std::cout << "The time bubbleSort 10000: " << elapsed_ms.count() << " ms\n";
00331     p = "20000.txt";
00332     Lab1 h4(p);
00333     h4.filename = "bubbleSort" + h4.filename;
00334     begin = std::chrono::steady_clock::now();
00335     h4.bubbleSort();
00336     end = std::chrono::steady_clock::now();
00337     h4.output();
00338     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00339     std::cout << "The time bubbleSort 20000: " << elapsed_ms.count() << " ms\n";
00340     p = "40000.txt";
00341     Lab1 h5(p);
00342     h5.filename = "bubbleSort" + h5.filename;
00343     begin = std::chrono::steady_clock::now();
00344     h5.bubbleSort();
00345     end = std::chrono::steady_clock::now();
00346     h5.output();
00347     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00348     std::cout << "The time bubbleSort 40000: " << elapsed_ms.count() << " ms\n";
00349     p = "100000.txt";
00350     Lab1 h6(p);
00351     h6.filename = "bubbleSort" + h6.filename;
00352     begin = std::chrono::steady_clock::now();
00353     h6.bubbleSort();
00354     end = std::chrono::steady_clock::now();
00355     h6.output();
00356     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00357     std::cout << "The time bubbleSort 100000: " << elapsed_ms.count() << " ms\n";
00358
00359     std::cout << "\n";
00360     p = "100.txt";
00361     Lab1 k(p);
00362     k.filename = "selSort" + k.filename;
00363     begin = std::chrono::steady_clock::now();
00364     k.selectionSort();
00365     end = std::chrono::steady_clock::now();
00366     k.output();
```

```
00367     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00368     std::cout << "The time selSort 100: " << elapsed_ms.count() << " ms\n";
00369     p = "1000.txt";
00370     Lab1 k1(p);
00371     k1.filename = "selSort" + k1.filename;
00372     begin = std::chrono::steady_clock::now();
00373     k1.selectionSort();
00374     end = std::chrono::steady_clock::now();
00375     k1.output();
00376     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00377     std::cout << "The time selSort 1000: " << elapsed_ms.count() << " ms\n";
00378     p = "5000.txt";
00379     Lab1 k2(p);
00380     k2.filename = "selSort" + k2.filename;
00381     begin = std::chrono::steady_clock::now();
00382     k2.selectionSort();
00383     end = std::chrono::steady_clock::now();
00384     k2.output();
00385     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00386     std::cout << "The time selSort 5000: " << elapsed_ms.count() << " ms\n";
00387     p = "10000.txt";
00388     Lab1 k3(p);
00389     k3.filename = "selSort" + k3.filename;
00390     begin = std::chrono::steady_clock::now();
00391     k3.selectionSort();
00392     end = std::chrono::steady_clock::now();
00393     k3.output();
00394     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00395     std::cout << "The time selSort 10000: " << elapsed_ms.count() << " ms\n";
00396     p = "20000.txt";
00397     Lab1 k4(p);
00398     k4.filename = "selSort" + k4.filename;
00399     begin = std::chrono::steady_clock::now();
00400     k4.selectionSort();
00401     end = std::chrono::steady_clock::now();
00402     k4.output();
00403     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00404     std::cout << "The time selSort 20000: " << elapsed_ms.count() << " ms\n";
00405     p = "40000.txt";
00406     Lab1 k5(p);
00407     k5.filename = "selSort" + k5.filename;
00408     begin = std::chrono::steady_clock::now();
00409     k5.selectionSort();
00410     end = std::chrono::steady_clock::now();
00411     k5.output();
00412     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00413     std::cout << "The time selSort 40000: " << elapsed_ms.count() << " ms\n";
00414     p = "100000.txt";
00415     Lab1 k6(p);
00416     k6.filename = "selSort" + k6.filename;
00417     begin = std::chrono::steady_clock::now();
00418     k6.selectionSort();
00419     end = std::chrono::steady_clock::now();
00420     k6.output();
00421     elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end -
begin);
00422     std::cout << "The time selSort 100000: " << elapsed_ms.count() << " ms\n";
00423 }
00424
```


Selection sort
ms
250 000
200 000
150 000
100 000
50 000
20 000
40 000
60000
80000
100000
N


Bubble sort
ms
1.2 × 10^6
1.0 × 10^6
800000
600000
400000
200000
20 000
40000
60 000
80000
100 000
N


Heap sort
ms
1500
1000
500
20000
40 000
60 000
80 000
100000
N



Таким образом, пирамидальную сортировку предпочтительно использовать на любых объемах данных по сравнению с сортировкой пузырьком и выборочной сортировкой.

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