**VECTOR.H**

---------------------------------------------------------------------------------------------

#pragma once

#include <string>

#include <algorithm>

class Vector

{

size\_t n = 0;

int\* vals; // dynamic array

public:

Vector(size\_t n);

Vector(const Vector& vec); // copy-constructor creates a deep copy of vec and saves it to other object. Also copies the vals elements.

std::string status() const;

size\_t size() const { return n; }

int operator[] (size\_t) const;

int& operator[] (size\_t i) { return vals[i]; };

void swap(size\_t i, size\_t j);

// https://airtucha.github.io/SortVis/

void sort() { std::sort(vals, vals + n); }; // C++ built-in sorting algorithm

// https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0\_%D0%BF%D1%83%D0%B7%D1%8B%D1%80%D1%8C%D0%BA%D0%BE%D0%BC

void bubble\_sort();

// https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0\_%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D1%88%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC

void shaker\_sort();

//https://stackoverflow.com/questions/1174169/function-passed-as-template-argument

template<typename Functor>

void quick\_sort(Functor);

static ptrdiff\_t partition\_lomuto(Vector&, ptrdiff\_t, ptrdiff\_t);

static ptrdiff\_t partition\_hoare(Vector&, ptrdiff\_t, ptrdiff\_t);

template<typename Functor>

void qsort\_recursive(ptrdiff\_t, ptrdiff\_t, Functor);

void merge\_sort();

private:

Vector\* merge(Vector\* B, Vector\* C);

};

class test\_vector

{

public:

static void test\_1();

};

**VECTOR.CPP**

---------------------------------------------------------------------------------------------

#include "Vector.h"

#include <iostream>

#include <cstdlib>

#include <algorithm>

#include <vector>

Vector::Vector(size\_t n)

:n{ n }

{

/\*vals = new int[n];

for (int i = 0; i < n; i++)

vals[i] = 1 + std::rand() / ((RAND\_MAX + 1u) / 25);

vals[0] = 19;\*/

vals = new int[n]{ 64, 42, 80, 20, 69, 63, 14, 84, 76, 22 };

vals = new int[n] { 5,2,3,1,0,0,0,0,0,0 };

}

Vector::Vector(const Vector& vec)

{

n = vec.size();

vals = new int[n];

for (size\_t i = 0; i < n; ++i)

vals[i] = vec[i];

}

int Vector::operator[] (size\_t i) const { return vals[i]; }

std::string Vector::status() const

{

std::string result;

for (size\_t i = 0; i < n; ++i)

result += std::to\_string(vals[i]) + "; ";

return result;

//int\* p;

//p = nullptr;

//p = NULL;

//p = 0;

}

void Vector::bubble\_sort()

{

for (size\_t i = 1; i < n; i++)

{

for (size\_t j = 1; j < n - i + 1; j++)

{

if (vals[j] < vals[j - 1])

swap(j, j - 1);

}

std::cout << (\*this).status() << std::endl;

}

}

void Vector::shaker\_sort()

{

size\_t left = 0;

size\_t right = n;

size\_t control = n;

while (left < right-1)

{

for (size\_t i = left+1; i < right; i++)

{

if (vals[i - 1] > vals[i])

{

swap(i - 1, i);

control = i;

}

}

std::cout << control << ": " << (\*this).status() << std::endl;

right = control;

for (size\_t i = right-1; i > left; i--)

{

if (vals[i - 1] > vals[i])

{

swap(i - 1, i);

control = i;

}

}

std::cout << control << ": " << (\*this).status() << std::endl;

left = control;

}

}

template<typename Functor>

void Vector::quick\_sort(Functor f)

{

qsort\_recursive(0, n-1, f);

}

template<typename Functor>

void Vector::qsort\_recursive(ptrdiff\_t low, ptrdiff\_t high, Functor partition\_template)

{

if (low < high)

{

std::swap(vals[low + rand() % (high - low + 1)], vals[low]);

ptrdiff\_t p = partition\_lomuto(\*this, low, high);

qsort\_recursive(low, p - 1ll, partition\_template);

qsort\_recursive(p + 1ll, high, partition\_template);

}

}

ptrdiff\_t Vector::partition\_lomuto(Vector& vec, ptrdiff\_t low, ptrdiff\_t high)

{

size\_t pivot = vec[high];

size\_t i = low;

for (size\_t j = low; j < high; ++j)

{

if (vec[j] < pivot)

{

std::swap(vec[i], vec[j]);

i++;

}

}

std::swap(vec[high], vec[i]);

return i;

}

ptrdiff\_t Vector::partition\_hoare(Vector& vec, ptrdiff\_t low, ptrdiff\_t high)

{

ptrdiff\_t pivot = vec[low];

ptrdiff\_t i = low + 1;

ptrdiff\_t j = high;

while (true)

{

while (pivot < vec[j]) --j;

while (i < high + 1 && pivot > vec[i]) ++i;

if (i < j)

{

std::swap(vec[i], vec[j]);

++i; --j;

}

else

{

std::swap(vec[low], vec[j]);

return j;

}

}

}

Vector\* Vector::merge(Vector\* B, Vector\* C)

{

size\_t i = 0, j = 0, k = 0;

size\_t p = B->size();

size\_t q = C->size();

Vector\* A = new Vector(p + q);

while (i < p && j < q)

{

if (B->vals[i] < C->vals[j])

{

A->vals[k] = B->vals[i];

i++;

}

else

{

A->vals[k] = C->vals[j];

j++;

}

k++;

}

while (i < p)

{

A->vals[k] = B->vals[i];

i++;

}

while (j < q)

{

A->vals[k] = C->vals[j];

j++;

}

return A;

}

void Vector::merge\_sort()

{}

void Vector::swap(size\_t i, size\_t j)

{

int temp = vals[i];

vals[i] = vals[j];

vals[j] = temp;

}

void test\_vector::test\_1()

{

Vector my\_vector(10);

std::cout << "First vector: " << std::endl

<<my\_vector.status() << std::endl;

Vector my\_vector\_copy{ my\_vector };

std::cout << "Copy through initializer list: " << std::endl

<< my\_vector\_copy.status() << std::endl;

std::cout << "\nC++ built-in sort: " << std::endl;

Vector vec{ my\_vector };

std::cout << "Initial state: " << std::endl;

std::cout << vec.status() << std::endl;

vec.sort();

std::cout << "Sorted vector: " << std::endl;

std::cout << vec.status() << std::endl;

std::cout << "\nPerform bubble sort: \n";

vec = Vector{ my\_vector };

std::cout << "Initial state: " << std::endl;

std::cout << vec.status() << std::endl;

std::cout << "Bubble sort steps: " << std::endl;

vec.bubble\_sort();

std::cout << "\nPerform shaker sort: \n";

vec = Vector{ my\_vector };

std::cout << "Initial state: " << std::endl;

std::cout << vec.status() << std::endl;

std::cout << "Shaker sort steps: " << std::endl;

vec.shaker\_sort();

std::cout << "\nPerform partition Lomuto: \n";

vec = Vector{ my\_vector };

std::cout << "Initial state: " << std::endl;

std::cout << vec.status() << std::endl;

std::cout << "Lomuto partitioned array from 0 to " << vec.size() << ": " << std::endl;

size\_t j = Vector::partition\_lomuto(vec, 0, vec.size());

std::cout << "with pivot at position j = " << j << std::endl;

std::cout << "Lomuto partitioned array from " << 0 << " to " << j << ": " << std::endl;

j = Vector::partition\_lomuto(vec, 0, j);

std::cout << "with pivot at position j = " << j << std::endl;

std::cout << "\nPerform partition Hoare: \n";

vec = Vector{ my\_vector };

std::cout << "Initial state: " << std::endl;

std::cout << vec.status() << std::endl;

std::cout << "Hoare partitioned array from 0 to " << vec.size() << ": " << std::endl;

j = Vector::partition\_hoare(vec, 0, vec.size());

std::cout << "with pivot at position j = " << j << std::endl;

std::cout << "Hoare partitioned array from " << j+1 << " to " << vec.size() << ": " << std::endl;

j = Vector::partition\_hoare(vec, j, vec.size());

std::cout << "with pivot at position j = " << j << std::endl;

std::cout << "\nPerform quick sort: \n";

vec = Vector{ my\_vector };

std::cout << "Initial state: " << std::endl;

std::cout << vec.status() << std::endl;

std::cout << "Quick sort steps with Lomuto partition: " << std::endl;

vec.quick\_sort(Vector::partition\_lomuto);

std::cout << "\nPerform quick sort: \n";

vec = Vector{ my\_vector };

std::cout << "Initial state: " << std::endl;

std::cout << vec.status() << std::endl;

std::cout << "Quick sort steps with Hoare partition: " << std::endl;

vec.quick\_sort(Vector::partition\_hoare);

}

**SORT.CPP**

---------------------------------------------------------------------------------------------

include <iostream>

#include "Vector.h"

int main()

{

test\_vector::test\_1();

}

**Program output**

First vector:

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

Copy through initializer list:

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

C++ built-in sort:

Initial state:

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

Sorted vector:

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

Perform bubble sort:

Initial state:

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

Bubble sort steps:

2; 3; 1; 0; 0; 0; 0; 0; 0; 5;

2; 1; 0; 0; 0; 0; 0; 0; 3; 5;

1; 0; 0; 0; 0; 0; 0; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

Perform shaker sort:

Initial state:

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

Shaker sort steps:

9: 2; 3; 1; 0; 0; 0; 0; 0; 0; 5;

1: 0; 2; 3; 1; 0; 0; 0; 0; 0; 5;

8: 0; 2; 1; 0; 0; 0; 0; 0; 3; 5;

2: 0; 0; 2; 1; 0; 0; 0; 0; 3; 5;

7: 0; 0; 1; 0; 0; 0; 0; 2; 3; 5;

3: 0; 0; 0; 1; 0; 0; 0; 2; 3; 5;

6: 0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

6: 0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

Perform partition Lomuto:

Initial state:

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

Lomuto partitioned array from 0 to 10:

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

with pivot at position j = 10

Lomuto partitioned array from 0 to 10:

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

with pivot at position j = 10

Perform partition Hoare:

Initial state:

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

Hoare partitioned array from 0 to 10:

-33686019; 2; 3; 1; 0; 0; 0; 0; 0; 0;

with pivot at position j = 10

Hoare partitioned array from 11 to 10:

-33686019; 2; 3; 1; 0; 0; 0; 0; 0; 0;

with pivot at position j = 10

Perform quick sort:

Initial state:

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

Quick sort steps with Lomuto partition:

0; 5; 3; 1; 0; 0; 0; 0; 0; 2;

0; 2; 3; 1; 0; 0; 0; 0; 0; 5;

0; 2; 3; 1; 0; 0; 0; 0; 0; 5;

0; 2; 3; 1; 0; 0; 0; 0; 0; 5;

0; 2; 3; 1; 0; 0; 0; 0; 0; 5;

0; 2; 3; 1; 0; 0; 0; 0; 0; 5;

0; 2; 3; 1; 0; 0; 0; 0; 0; 5;

0; 2; 3; 1; 0; 0; 0; 0; 0; 5;

0; 2; 3; 1; 0; 0; 0; 0; 0; 5;

0; 2; 3; 1; 0; 0; 0; 0; 0; 5;

0; 0; 3; 1; 0; 0; 0; 2; 0; 5;

0; 0; 0; 1; 0; 0; 0; 3; 2; 5;

0; 0; 0; 0; 2; 0; 0; 3; 1; 5;

0; 0; 0; 0; 0; 2; 0; 3; 1; 5;

0; 0; 0; 0; 0; 0; 2; 3; 1; 5;

0; 0; 0; 0; 0; 0; 1; 3; 2; 5;

0; 0; 0; 0; 0; 0; 1; 3; 2; 5;

0; 0; 0; 0; 0; 0; 1; 3; 2; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

Perform quick sort:

Initial state:

5; 2; 3; 1; 0; 0; 0; 0; 0; 0;

Quick sort steps with Hoare partition:

0; 2; 5; 1; 0; 0; 0; 0; 0; 3;

0; 1; 5; 2; 0; 0; 0; 0; 0; 3;

0; 1; 2; 5; 0; 0; 0; 0; 0; 3;

0; 1; 2; 0; 5; 0; 0; 0; 0; 3;

0; 1; 2; 0; 0; 5; 0; 0; 0; 3;

0; 1; 2; 0; 0; 0; 5; 0; 0; 3;

0; 1; 2; 0; 0; 0; 0; 5; 0; 3;

0; 1; 2; 0; 0; 0; 0; 0; 5; 3;

0; 1; 2; 0; 0; 0; 0; 0; 3; 5;

0; 0; 2; 0; 0; 0; 0; 1; 3; 5;

0; 0; 1; 0; 0; 0; 0; 2; 3; 5;

0; 0; 1; 0; 0; 0; 0; 2; 3; 5;

0; 0; 1; 0; 0; 0; 0; 2; 3; 5;

0; 0; 1; 0; 0; 0; 0; 2; 3; 5;

0; 0; 1; 0; 0; 0; 0; 2; 3; 5;

0; 0; 1; 0; 0; 0; 0; 2; 3; 5;

0; 0; 0; 1; 0; 0; 0; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;

0; 0; 0; 0; 0; 0; 1; 2; 3; 5;