EcoLab1

Merge Sort

Выполнила

Христолюбова Наталья

21ПИ-3

# Алгоритм

**Сортировка слиянием** (англ. *Merge sort*) — алгоритм сортировки, использующий O(n) дополнительной памяти и работающий за O(n log(n)) времени.

Алгоритм использует принцип «разделяй и властвуй»: задача разбивается на подзадачи меньшего размера, которые решаются по отдельности, после чего их решения комбинируются для получения решения исходной задачи. Конкретно процедуру сортировки слиянием можно описать следующим образом:

1. Если в рассматриваемом массиве один элемент, то он уже отсортирован — алгоритм завершает работу.
2. Иначе массив разбивается на две части, которые сортируются рекурсивно.
3. После сортировки двух частей массива к ним применяется процедура слияния, которая по двум отсортированным частям получает исходный отсортированный массив.

**Псевдокод**

void mergeSort(int A[], int left, int right) {

if (left >= right)

return

int mid = left + (right - left) / 2

mergeSort(A, left, mid)

mergeSort(A, mid + 1, right)

merge(A, left, mid, right)

}

# Реализация

Так как разные типы отличаются по размеру, то на одинаковое количество элементов выделяется разное количество памяти:

* char - 1 байт
* int, float - 4 байта
* double - 8 байт

В остальном отличий между функциями МergeSort1, MergeSort4 и MergeSort8 нет.

char\* ECOCALLMETHOD MergeSort1(void\* base, void\* new\_base, int pos, int end,

int (\_\_cdecl \*compare) (const void\*, const void\*)){

char \*arr = (char \*)base;

char \*new\_arr = (char \*)new\_base;

char \*arr1, \*arr2;

int n = end - pos;

int pos2 = pos + n / 2;

int i = pos, j = pos2, ind = pos;

if (n != 1) {

arr1 = MergeSort1(base, new\_base, pos, pos2, compare);

arr2 = MergeSort1(base, new\_base, pos2, end, compare);

if (arr1 == new\_arr){

new\_arr = arr;

arr = arr1;

}

} else {

return arr;

}

for (;(i < pos2) && (j < end); ind++){

if (compare(&arr[i], &arr2[j]) < 0) {

new\_arr[ind] = arr[i++];

} else {

new\_arr[ind] = arr2[j++];

}

}

if (i < pos2) {

for (; ind < end; ind++){

new\_arr[ind] = arr[i++];

}

} else {

for (; ind < end; ind++){

new\_arr[ind] = arr2[j++];

}

}

return new\_arr;

}

void\* ECOCALLMETHOD CEcoLab1\_MergeSort(struct IEcoLab1\* me, void \*base, size\_t size, int n,

int (\_\_cdecl \*compare) (const void\*, const void\*)){

CEcoLab1\* pCMe = (CEcoLab1\*)me;

void \*new\_base = (void \*)pCMe->m\_pIMem->pVTbl->Alloc(pCMe->m\_pIMem, n \* size);

int i = 0;

if (size == 1){

base = MergeSort1(base, new\_base, 0, n, compare);

} else if (size == 4) {

base = MergeSort4(base, new\_base, 0, n, compare);

} else if (size == 8) {

base = MergeSort8(base, new\_base, 0, n, compare);

}

return base;

}

# Асимптотика

Асимптотика алгоритма сортировки слиянием (MergeSort) заключается в следующем:

1. Каждый уровень состоит из одинакового количества элементов, на каждом уровне происходит серия операций merge. Суммарно на каждом уровне все они выполняются за O(n), так как время на merge линейно зависит от количества чисел в массивах.
2. Количество уровней равно log(n), поэтому весь алгоритм работает за O(n log(n)).

При изменение порядка элементов в массиве, сложность алгоритма не меняется.

# Результаты тестирования

Для сравнения использована встроенная быстрая сортировка.

**Int:**

n: 1000

Time Merge Sort: 0.00 ms

Time Quick Sort: 1.00 ms

n: 10000

Time Merge Sort: 5.00 ms

Time Quick Sort: 5.00 ms

n: 100000

Time Merge Sort: 58.00 ms

Time Quick Sort: 66.00 ms

n: 1000000

Time Merge Sort: 686.00 ms

Time Quick Sort: 625.00 ms

n: 10000000

Time Merge Sort: 8775.00 ms

Time Quick Sort: 6705.00 ms

**Float:**

n: 1000

Time Merge Sort: 0.00 ms

Time Quick Sort: 1.00 ms

n: 10000

Time Merge Sort: 6.00 ms

Time Quick Sort: 6.00 ms

n: 100000

Time Merge Sort: 68.00 ms

Time Quick Sort: 74.00 ms

n: 1000000

Time Merge Sort: 844.00 ms

Time Quick Sort: 797.00 ms

n: 10000000

Time Merge Sort: 9603.00 ms

Time Quick Sort: 8097.00 ms

**Double:**

n: 1000

Time Merge Sort: 0.00 ms

Time Quick Sort: 0.00 ms

n: 10000

Time Merge Sort: 6.00 ms

Time Quick Sort: 7.00 ms

n: 100000

Time Merge Sort: 66.00 ms

Time Quick Sort: 78.00 ms

n: 1000000

Time Merge Sort: 848.00 ms

Time Quick Sort: 858.00 ms

n: 10000000

Time Merge Sort: 9372.00 ms

Time Quick Sort: 8107.00 ms

**Char:**

n: 1000

Time Merge Sort: 0.00 ms

Time Quick Sort: 0.00 ms

n: 10000

Time Merge Sort: 5.00 ms

Time Quick Sort: 3.00 ms

n: 100000

Time Merge Sort: 73.00 ms

Time Quick Sort: 34.00 ms

n: 1000000

Time Merge Sort: 735.00 ms

Time Quick Sort: 297.00 ms

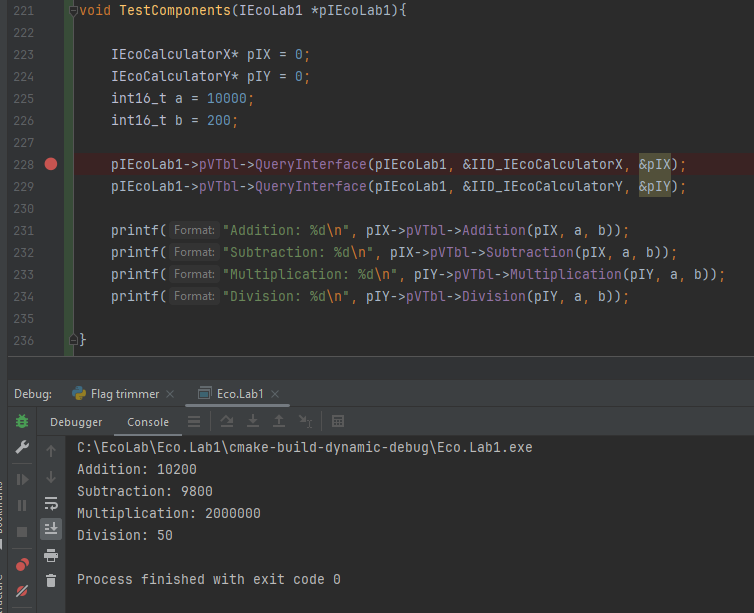
n: 10000000

Time Merge Sort: 8268.00 ms

Time Quick Sort: 2983.00 ms

# Вторая лабораторная

## Пример тестирования:



## Получение компонентов методом включения:

result = pIBus->pVTbl->QueryComponent(pIBus, &CID\_EcoCalculatorA, 0, &IID\_IEcoCalculatorX, (void \*\*) &pCMe->m\_pIX);  
if (result != 0 || pCMe->m\_pIX == 0) {  
 result = pIBus->pVTbl->QueryComponent(pIBus, &CID\_EcoCalculatorC, 0, &IID\_IEcoCalculatorX, (void \*\*) &pCMe->m\_pIX);  
}  
  
result = pIBus->pVTbl->QueryComponent(pIBus, &CID\_EcoCalculatorE, 0, &IID\_IEcoCalculatorY, (void \*\*) &pCMe->m\_pIY);  
if (result != 0 || pCMe->m\_pIY == 0) {  
 result = pIBus->pVTbl->QueryComponent(pIBus, &CID\_EcoCalculatorD, 0, &IID\_IEcoCalculatorY, (void \*\*) &pCMe->m\_pIY);  
}

pCMe->m\_pVTblIX = (IEcoCalculatorXVTbl\*)pCMe->m\_pIX->pVTbl;  
pCMe->m\_pVTblIY = (IEcoCalculatorYVTbl\*)pCMe->m\_pIY->pVTbl;

В данном случае интерфейс X представлен взаимно заменяемыми компонентами A и C, а интерфейс Y – компонентами E и D.

Код, добавленный в QueryInterface

} else if (IsEqualUGUID(riid, &IID\_IEcoCalculatorX)) {  
 \*ppv = &pCMe->m\_pVTblIX;  
 pCMe->m\_pVTblIEcoLab1->AddRef((IEcoLab1 \*) pCMe);  
} else if (IsEqualUGUID(riid, &IID\_IEcoCalculatorY)) {  
 \*ppv = &pCMe->m\_pVTblIY;  
 pCMe->m\_pVTblIEcoLab1->AddRef((IEcoLab1\*) pCMe);

## Получение компонентов методом агрегирования:

result = pIBus->pVTbl->QueryComponent(pIBus, &CID\_EcoCalculatorA, 0, &IID\_IEcoUnknown,  
 (void \*\*) &pCMe->m\_pInnerUnknown);  
if (result != 0 || pCMe->m\_pInnerUnknown == 0) {  
 result = pIBus->pVTbl->QueryComponent(pIBus, &CID\_EcoCalculatorB, 0, &IID\_IEcoUnknown,  
 (void \*\*) &pCMe->m\_pInnerUnknown);  
}  
if (result != 0 || pCMe->m\_pInnerUnknown == 0) {  
 result = pIBus->pVTbl->QueryComponent(pIBus, &CID\_EcoCalculatorC, 0, &IID\_IEcoUnknown,  
 (void \*\*) &pCMe->m\_pInnerUnknown);  
}  
if (result != 0 || pCMe->m\_pInnerUnknown == 0) {  
 result = pIBus->pVTbl->QueryComponent(pIBus, &CID\_EcoCalculatorD, 0, &IID\_IEcoUnknown,  
 (void \*\*) &pCMe->m\_pInnerUnknown);  
}  
if (result != 0 || pCMe->m\_pInnerUnknown == 0) {  
 result = pIBus->pVTbl->QueryComponent(pIBus, &CID\_EcoCalculatorE, 0, &IID\_IEcoUnknown,  
 (void \*\*) &pCMe->m\_pInnerUnknown);  
}

В данном случае представлены все компоненты по очереди. При любой комбинации компонентов, содержащих оба интерфейса X и Y, программа будет выполняться правильно.

Код, добавленный в QueryInterface

} else if (IsEqualUGUID(riid, &IID\_IEcoCalculatorX) && pCMe->m\_pInnerUnknown != 0) {  
 return pCMe->m\_pInnerUnknown->pVTbl->QueryInterface(pCMe->m\_pInnerUnknown, riid, ppv);  
} else if (IsEqualUGUID(riid, &IID\_IEcoCalculatorY) && pCMe->m\_pInnerUnknown != 0) {  
 return pCMe->m\_pInnerUnknown->pVTbl->QueryInterface(pCMe->m\_pInnerUnknown, riid, ppv);