МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Реализация и исследование алгоритма сортировки TimSort

Студент гр. 3383 _____ Матвеев Н.С.

Преподаватель _____ Калишенко Е.Л.

Санкт-Петербург

2024

Цель работы

Цель работы заключается в изучении различных алгоритмов сортировки, а также в реализации и анализе гибридной сортировки под названием TimSort на языке C++.

Задание

Имеется массив данных для сортировки int arr[] размера n

Необходимо отсортировать его алгоритмом сортировки TimSort по убыванию модуля.

Так как TimSort - это гибридный алгоритм, содержащий в себе сортировку слиянием и сортировку вставками, то вам предстоит использовать оба этих алгоритма. Поэтому нужно выводить разделённые блоки, которые уже отсортированы сортировкой вставками.

Кратко алгоритм сортировки можно описать так:

Вычисление min_run по размеру массива n (для упрощения отладки n уменьшается, пока не станет меньше 16, а не 64)

Разбиение массива на частично-упорядоченные (в т.ч. и по убыванию) блоки длины не меньше min_run

Сортировка вставками каждого блока

Слияние каждого блока с сохранением инварианта и использованием галопа (галоп начинать после 3-х вставок подряд)

Исследование

После успешного решения задачи в рамках курса проведите исследование данной сортировки на различных размерах данных (10/1000/100000), сравнив полученные результаты с теоретической оценкой (для лучшего, среднего и худшего случаев), и разного размера min_run. Результаты исследования предоставьте в отчете.

Для исследования используйте стандартный алгоритм вычисления min_run и начинайте галоп после 7-ми вставок подряд.

Примечание:

Нельзя пользоваться готовыми библиотечными функциями для сортировки, нужно сделать реализацию сортировки вручную.

Сортировка должна быть устойчивой.

Обратите внимание на пример.

Формат ввода

Первая строка содержит натуральное число n - размерность массива, следующая строка содержит элементы массива через пробел.

Формат вывода

Выводятся разделённые блоки для сортировки в формате "Part i: *отсортированный разделённый массив*"

Затем для каждого слияния выводится количество вхождений в режим галопа и получившийся массив в формате

"Gallops i: *число вхождений в галоп*

Merge i: *итоговый массив после слияния*"

Последняя строчка содержит финальный результат сортировки массива с надписью "Answer: "

ХОД ВЫПОЛНЕНИЯ РАБОТЫ

Подключение необходимых библиотек:

- iostream – библиотека, необходимая для реализации ввода и вывода
- vector – заголовочный файл, содержащий описание класса vector и все его методы
- stack – заголовочный файл, содержащий описание класса стек

Функция MinRunLen():

На вход функция принимает количество элементов в массиве. Флаг становится равным 1, если среди сдвинутых бит будет хотя бы один не равный единице. Циклом while() сдвигается двоичная запись числа вправо пока поданное число больше или равно 16. Функция возвращает минимальную длину подмассива.

Функция InsertSort():

Функция сортировки по убыванию массива алгоритмом вставки. На вход получает ссылку на массив. Циклом for() перебираются элементы массивы начиная с первого, внутри цикл while() проверяет массив перед элементом, который рассматривается на данном шаге цикла for(). Рассматриваются от элемента к началу. Если элемент меньше предыдущего, то они меняются местами.

Функция Cut():

Делит исходный массив на подмассивы. Принимает ссылку на исходный массив, минимальную длину подмассива, индекс откуда брать новый подмассив, ссылку на двумерный массив для подмассивов. Если индекс равен длине массива, функция не выполняется. Создается флаг, для указания идут ли числа в подмассиве по возрастанию или убыванию. Циклом while() записываются элементы в подмассив, пока они в соответствии с флагом

убывают или возрастают. Если длина полученного подмассива меньше минимальной, то добираются элементы до минимальной длины. Полученный подмассив предается в InsertSort() для сортировки по убыванию. Записывается полученный подмассив в двумерный вектор. Вызывается рекурсивно данную функцию но уже с новым индексом.

Функция Search():

Функция неточного поиска (модификация бинарного). На вход получает размер массива, значение с которым сравнивать, ссылка на массив, в котором происходит поиск и индекс откуда применять данный поиск. Левый край становится равным индексу, правый размеру массива. Пока левый край меньше правого находится середина данной части и если элемент из середины меньше числа, которое передавалось для сравнения мы сохраняем индекс этого элемента и правый край равен этому элементу, иначе левый край становится равным середина + 1.

Функция Merge():

Функция сливания двух массивов. Принимает ссылку на первый массив, куда будет записано слияние и константную ссылку на второй массив. Заводятся счетчики для вызова галопа, для шагов перед галопом в каждом массиве. В цикле while() сравниваются элементы двух массивов. Если добавляется элемент из первого, то и счетчик первого увеличивается, аналогично для второго массива и его счетчика. Если счетчик уже равен 3, то увеличивается счетчик галопа и вызывается функция неточного поиска для нахождения индекса, до которого все элементы соответствуют знаку сравнения. Если после завершения цикла, остались элементы в первом или втором массиве они дописываются в конец результата слияния.

Функция Invariant():

Функция для проверки инварианта для стека из подмассивов. Если размер 3, то проверяется Z>X+Y□Y>X. Если размер 2, то Y>X. Принимает массив из двух-трех элементов: размеры подмассивов. Возвращает true или false.

Функция Blocks():

Функция для определения порядка сливания подмассивов. Принимает ссылку на двумерный массив подмассивов. Создается стек из векторов для хранения подмассивов и вектор для хранения размеров трех-двух подмассивов (для проверки инварианта). Циклом for() перебираются все подмассивы и заносятся в стек и их размер в вектор для инварианта. Если в векторе размеров элементов 3, то проверяется инвариант для 3х, если он не выполняется, то если размер подмассива Z < размера подмассива X, то происходит слияние Y и Z. Вытаскиваются 3 верхних элемента из стека при помощи pop(), второй с третьим сливаются при помощи Merge(). Удаляется первый элемент из массива размеров и меняется второй. Похожим образом сливаются первый со вторым из верхушки стека, если размер первого больше третьего. Только удаляются не 3 элемента, а 2 из стека и массив размеров меняется соответствующе. Если после слияния в тройке, можно слить полученную двойку, то она сливается. Если же инвариант для тройки выполняется, то мы убираем размер подмассива на первом месте из массива размеров — делается это для освобождения места под новый размер.

Если размер равен двум, то проверяется инвариант для двух — если он не выполняется происходит слияние.

Если в конце концов выполняется инвариант, но в массиве более одного элемента, то начиная с верхушки сливаются массивы пока не станет одного. Слияние происходит также как и при добавлении новых подмассивов в стек.

Функция возвращает последний элемент стека, то есть весь отсортированный массив.

Функция int main():

В функции main() считывается с клавиатуры количество элементов в массиве, а также сам массив чисел через пробел. Далее создается двумерный вектор для подмассивов. При помощи функции MinRunLen() высчитыается минимальная длина подмассива. При помощи функции Cut() делим исходный массив на подмассивы. Выводятся полученные подмассивы. Далее для сортировки вызывается функция Blocks(). Выводится результат.

АНАЛИТИЧЕСКАЯ ЧАСТЬ

Для сравнения эффективности работы с точки зрения времени была подключена библиотек <ctime> и использована функция *clock()*. Для нахождения времени выполнения находилась разница между точкой старта и точкой конца. Время работы сравнивалось на разных объемах данных и на разных случаях. Результаты сравнения представлены в таблице 1.

Таблица 1: Время выполнения сортировки TimSort

	Лучший случай	Средний случай	Худший случай
10	0.000019	0.000031	0.000033
1000	0.00012	0.001188	0.001692
100000	0.0038	0.3058	0.4822

Выводы

В ходе выполнения лабораторной работы были изучены различные алгоритмы сортировки данных, а также была реализована и проанализирована гибридная сортировка под названием TimSort на языке программирования C++.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

файл main.cpp:

```
#include "Blocks.h"
int main() {
    size_t n;
    std::cin >> n;
    size_t min_run = MinRunLen(n);
    std::vector<int> input_nums(n);
    for (size_t i =0; i<n; ++i){
        std::cin >> input_nums[i];
    }
    std::vector<std::vector<int>> all_runs;
    Cut(input_nums, min_run, 0, all_runs);
    size_t c1 =0;
    for (std::vector<int>& i : all_runs){
        std::cout << "Part " << c1++ << ":";
        for (int j : i){
            std::cout << " " << j;
        }
        std::cout << "\n";
    }
    input_nums = Blocks(all_runs);
    std::cout << "Answer:";
    for (int i : input_nums){
        std::cout << " " << i;
    }
}


    for (size_t i =0; i< 100; ++i){
        data2.push_back(rand() % (10- 0 + 1) + 0);
    }
    check(data, data2);
    data.clear();
    data2.clear();
}
```

файл Tests.cpp:

```
#include <gtest/gtest.h>
#include "Blocks.h"

TEST(TimSortTests, Test1) {
   EXPECT_EQ(MinRunLen(1000), 16);
   EXPECT_EQ(MinRunLen(10000), 10);
   EXPECT_EQ(MinRunLen(25), 13);
}
TEST(TimSortTests, Test2) {
   std::vector<int> nums = {5, 12, 3, -10, 9};
   std::vector<int> answer = {12, -10, 9, 5, 3};
   size_t n =5;
   std::vector<std::vector<int>> res;
   Cut(nums, MinRunLen(5), 0, res);
   EXPECT_EQ(res[0], answer);
}
TEST(TimSortTests, Test3) {
   std::vector<int> nums = {1, -2, 3, -4, 5, 6, -7, -8, 9, -10, 11, -10, -9, 8, 7, -7, -6, 6, 5, 4};
   std::vector<int> answer = {11, -10, -10, 9 ,-9, -8, 8, -7, 7, -7, 6, -6, 6, 5, 5, -4 ,4, 3, -2, 1};
   std::vector<std::vector<int>> res;
   Cut(nums, MinRunLen(20), 0, res);
   std::vector<int> part0 = {11, -10, 9, -8, -7, 6, 5, -4, 3, -2, 1};
   std::vector<int> part1 = {-10, -9, 8, 7, -7, -6, 6, 5, 4};
   EXPECT_EQ(res[0], part0);
   EXPECT_EQ(res[1], part1);
   nums = Blocks(res);
   EXPECT_EQ(nums, answer);
}
TEST(TimSortTests, Test4) {
   std::vector<int> nums = {-1, 2, 3, 4, 5, -6, 7, 8, -8, -8, 7, -7, 7, 6, -5, 4};
   std::vector<int> answer = {8, -8, -8, 7, 7, -7, 7, 6, -6, 5, -5, 4, 4, 3, 2, -1};
   std::vector<std::vector<int>> res;
   Cut(nums, MinRunLen(16), 0, res);
   std::vector<int> part0 = {8, 7, -6, 5, 4, 3, 2, -1 };
   std::vector<int> part1 = {-8, -8, 7, -7, 7,6,-5, 4 };
```

```
    EXPECT_EQ(res[0], part0);
    EXPECT_EQ(res[1], part1);
    nums = Blocks(res);
    EXPECT_EQ(nums, answer);
}
TEST(TimSortTests, Test5) {
    std::vector<int> nums = {51, -28, 3, 1, 0};
    std::vector<int> mas = {12, -10, 9, 5, 3};
    std::vector<int> answer = {51, -28, 12, -10, 9, 5, 3, 3, 1, 0};
    Merge(nums, mas);
    EXPECT_EQ(nums, answer);
}
TEST(TimSortTests, Test6) {
    std::vector<int> nums = {-10, 10, 9};
    std::vector<int> mas = {-10, -10, 7};
    std::vector<int> answer = {-10, 10, -10, -10, 9, 7};
    Merge(nums, mas);
    EXPECT_EQ(nums, answer);
}
int main(int argc, char **argv){
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

файл Cut.h:

```
#ifndef UNTITLED1_CUT_H
#define UNTITLED1_CUT_H
#include <iostream>
#include <vector>
#include <stack>
size_t MinRunLen(size_t n);
void InsertSort(std::vector<int>& mas);
void Cut(std::vector<int>& mas, size_t min_run, size_t index, std::vector<std::vector<int>>&
parts);
#endif //UNTITLED1_CUT_H
```

файл Cut.cpp:

```
#include "Cut.h"


size_t MinRunLen(size_t n){
    size_t flag =0;
    while (n>=64){
        flag |= n & 1;
        n >>= 1;
    }
    return n+flag;
}
void InsertSort(std::vector<int>& mas){
    for (size_t i = 1; i<mas.size(); ++i){
        size_t j =i;
        while(abs(mas[j-1]) < abs(mas[j]) && j > 0){
            int p = mas[j-1];
            mas[j-1] = mas[j];
            mas[j] = p;
            j-=1;
        }
    }
}
void Cut(std::vector<int>& mas, size_t min_run, size_t index, std::vector<std::vector<int>>&
parts){
    if (index >= mas.size()) return;
    std::vector<int> run;
    size_t len = 1;
    int flag;
    if (abs(mas[index]) <  abs(mas[index+1])) flag = 1;//increasing
    else flag =0;
    run.push_back(mas[index]);
    size_t i = index+1;
    if (flag==0){ //decreasing
        while (abs(mas[i]) <= abs(mas[i-1]) && i <mas.size()){
            run.push_back(mas[i]);
            len++;
```

```
            i++;
        }
    }
    else{ //increasing
        while (abs(mas[i]) > abs(mas[i-1]) && i <mas.size()){
            run.push_back(mas[i]);
            len++;
            i++;
        }
    }
    while(i < mas.size() && len < min_run){
        run.push_back(mas[i]);
        len++;
        i++;
    }
    if(len!=mas.size()) InsertSort(run);
    parts.push_back(run);
    run.clear();
    Cut(mas, min_run, index+len, parts);
}
```

файл Merge.h:

```
#ifndef UNTITLED1_MERGE_H
#define UNTITLED1_MERGE_H
#include "Cut.h"
size_t Search(size_t size, int value, const std::vector<int>& mas, size_t index);
void Merge(std::vector<int>& mas1, const std::vector<int>& mas2);
#endif //UNTITLED1_MERGE_H
```

файл Merge.cpp:

```
#include "Merge.h"
int counter_gallops = 0;
size_t Search(size_t size, int value, const std::vector<int>& mas, size_t index){
    size_t l = index;
    size_t r = size;
    int res = -1;
```

```
    while (l<r){
        size_t m = (l+r-1) / 2;
        if (abs(mas[m]) < abs(value)) {
            res = static_cast<int>(m);
            r=m;
        }
        else if (abs(mas[m]) >= abs(value)){
            l=m+1;
        }
    }
    if (res==-1) return size;
    else return res;
}
void Merge(std::vector<int>& mas1, const std::vector<int>& mas2){ //mas1 change
    std::vector<int> res;
    size_t i = 0, j = 0;
    size_t c1 = 0, c2 = 0, gallops = 0;
    size_t id; // index after gallop
    while (i < mas1.size() && j < mas2.size()) {
        if (abs(mas1[i]) >= abs(mas2[j])) {
            c2=0;
            res.push_back(mas1[i++]);
            c1++;
            if (c1==7){
                gallops++;
                id = Search(mas1.size(), mas2[j], mas1, i);
                c1 =0;
                for (size_t r =i; r<id; ++r){
                    res.push_back(mas1[r]);
                }
                i =id;
            }
        }
        else {
            c1=0;
            res.push_back(mas2[j++]);
            c2++;
```

```
            if (c2==7){
                gallops++;
                id = Search(mas2.size(), mas1[i], mas2, j);
                c2 =0;
                for (size_t r =j; r<id; ++r) res.push_back(mas2[r]);
                j =id;
            }
        }
    }
    if (j<mas2.size()){
        while(j < mas2.size()) res.push_back(mas2[j++]);
    }
    else if (i<mas1.size()){
        while(i<mas1.size()) res.push_back(mas1[i++]);
    }
    std::cout << "Gallops " << counter_gallops++ << ": "<< gallops << "\n";
    mas1 = res;
}
```

файл Blocks.h:

```
#ifndef UNTITLED1_BLOCKS_H
#define UNTITLED1_BLOCKS_H
#include "Merge.h"
bool Invariant(size_t size, std::vector<size_t>& mas);
std::vector<int> Blocks(std::vector<std::vector<int>>& runs);
#endif //UNTITLED1_BLOCKS_H
```

файл Blocks.cpp:

```
#include "Blocks.h"
bool Invariant(size_t size, std::vector<size_t>& mas){
    size_t X,Y,Z;
    if (size==3){
        X = mas[2];
        Y= mas[1];
        Z=mas[0];
    }
```

```
    else{
        X = mas[1];
        Y= mas[0];
    }
    if (size==3){
        if (Z > X + Y && Y > X) return true;
        else return false;
    }
    else{
        if (Y > X) return true;
        else return false;
    }
}

std::vector<int> Blocks(std::vector<std::vector<int>>& runs){
    size_t counter = 0;
    std::stack<std::vector<int>> blocks;
    std::vector<size_t> size_of_blocks;
    std::vector<int> Y;
    std::vector<int> X;
    for (std::vector<int>& i : runs){
        blocks.push(i);
        size_of_blocks.push_back(i.size());
        if (size_of_blocks.size() == 3){
            if (!Invariant(3, size_of_blocks)){
                if (size_of_blocks[0] < size_of_blocks[2]){ //Z<X
                    size_of_blocks[1] = size_of_blocks[0] + size_of_blocks[1];
                    X = blocks.top();
                    blocks.pop();
                    Y = blocks.top();
                    blocks.pop();
                    Merge(Y, blocks.top());
                    std::cout << "Merge " << counter << ":";
                    for (int num : Y){
                        std::cout << " " << num;
                    }
                    std::cout << "\n";
```

```
    counter++;
    blocks.pop();
    blocks.push(Y);
    blocks.push(X);
    size_of_blocks.erase(size_of_blocks.begin());
}
else{
    size_of_blocks[1] = size_of_blocks[2] + size_of_blocks[1];
    X = blocks.top();
    blocks.pop();
    Y = blocks.top();
    blocks.pop();
    Merge(Y, X);
    std::cout << "Merge " << counter << ":";
    for (int num : Y){
        std::cout << " " << num;
    }
    std::cout << "\n";
    counter++;
    blocks.push(Y);
    size_of_blocks.erase(size_of_blocks.begin()+2); //delete x
}
if (!Invariant(2, size_of_blocks)){
    X = blocks.top();
    blocks.pop();
    Y = blocks.top();
    blocks.pop();
    size_of_blocks[0] = size_of_blocks[0]+size_of_blocks[1];
    Merge(Y, X);
    std::cout << "Merge " << counter << ":";
    for (int num : Y){
        std::cout << " " << num;
    }
    std::cout << "\n";
    counter++;
    blocks.push(Y);
    size_of_blocks.pop_back();
```

```
            }
        }
        else{ // next "the tree" in stack
            size_of_blocks.erase(size_of_blocks.begin());
        }
    }
    else if (size_of_blocks.size()==2){
        if (!Invariant(2, size_of_blocks)){
            X = blocks.top();
            blocks.pop();
            Y = blocks.top();
            blocks.pop();
            size_of_blocks[0] = size_of_blocks[0]+size_of_blocks[1];
            Merge(Y, X);
            std::cout << "Merge " << counter << ":";
            for (int num : Y){
                std::cout << " " << num;
            }
            std::cout << "\n";
            counter++;
            blocks.push(Y);
            size_of_blocks.pop_back();
        }
    }
}
size_of_blocks.clear();
X.clear(), Y.clear();
std::vector<int> Z;
while(blocks.size()>1){ // stack is not clear
    if (blocks.size()==2){
        X = blocks.top();
        blocks.pop();
        Y = blocks.top();
        blocks.pop();
        Merge(Y, X);
        std::cout << "Merge " << counter << ":";
        for (int num : Y){
```

```
            std::cout << " " << num;
        }
        std::cout << "\n";
        counter++;
        blocks.push(Y);
    }
    else{
        X = blocks.top();
        blocks.pop();
        Y = blocks.top();
        blocks.pop();
        if (blocks.top().size() < X.size()){ //Z<X
            Merge(Y, blocks.top());
            std::cout << "Merge " << counter << ":";
            for (int num : Y){
                std::cout << " " << num;
            }
            std::cout << "\n";
            counter++;
            blocks.pop();
            blocks.push(Y);
            blocks.push(X);
        }
        else{
            Merge(Y, X);
            std::cout << "Merge " << counter << ":";
            for (int num : Y){
                std::cout << " " << num;
            }
            std::cout << "\n";
            counter++;
            blocks.push(Y);
        }
    }
  }
  return blocks.top();
}
```

файл Makefile:

```
all: main Tests clean
main: Cut.o Merge.o Blocks.o main.o
	g++ Cut.o Merge.o Blocks.o main.o -o main
Tests: Cut.o Merge.o Blocks.o Tests.o
	g++ Cut.o Merge.o Blocks.o Tests.o -lgtest -pthread -o Tests
Cut.o: Cut.cpp Cut.h
	g++ -c Cut.cpp
Merge.o: Merge.cpp Merge.h Cut.h
	g++ -c Merge.cpp
Blocks.o: Blocks.cpp Merge.h Blocks.h
	g++ -c Blocks.cpp
main.o: main.cpp Blocks.h
	g++ -c main.cpp
Tests.o: Tests.cpp Blocks.h
	g++ -c Tests.cpp
clean:
	rm -rf *.o
```