Практическое задание

Экспериментальный анализ различных методов сортировки

по предмету «Алгоритмы и анализ сложности»

**СОДЕРЖАНИЕ**

1. Постановка задачи
2. Реализация различных методов сортировки
3. Сравнение сортировок между собой на различных входных данных
   1. Описание методики сравнения
   2. Результаты сравнения и анализ причин, которые привели к таким результатам
4. Выводы
5. Список литературы
6. Приложение
7. **Постановка задачи**

В этой работе будут рассмотрены различные методы сортировок на основе экспериментальных данных. Главная задача – проверить, как же согласуется теоретическая модель сложности с практической; определить аспекты, которые на практике могут быть решающими при выборе сортировок; оценить нюансы поведения сортировок на различных типах входных данных; выявить, если это возможно, какие сортировки с худшим временем работы в теории, будут не такими плохими в применении на практике.

1. **Реализация различных методов сортировки**

Для решения поставленной задачи на языке C++ были реализованы следующие методы сортировок:

* простые схемы:
  + пузырьком bubble\_sort
  + вставками insertion\_sort
  + сортировка посредством выбора choices\_sort
* сортировка Шелла (с различными длинами промежутков) Shell\_sort
* быстрая сортировка quick\_sort
* сортировка слиянием merge\_sort
* сортировка кучей heap\_sort
* поразрядная сортировка radix\_sort

Процедуры были реализованы для различных входных типов данных:

* массива цифр от 0 до 9 однобайтового типа данных char
* массива целых чисел типа int
* массива строк типа string
* массива структуры для записи дат date

Программный код методов сортировки, а также структуры date описаны в приложении.

1. **Сравнение сортировок между собой на различных входных данных**
   1. **Описание методики сравнения**

Приводится анализ и сравнение сложности сортировок в теоретической модели. Также проходит практический анализ методов: сравнивается среднее время работы процедур на массивах разных типов, различной длины и по-разному порожденных массивах. Затем проводится сопоставление теоретических моделей сложности с результатами на практике.

* 1. **Результаты сравнения и анализ причин, которые привели к таким результатам**

**Теоретическая модель сложности алгоритмов** представлена в таблице 1.

|  |  |  |
| --- | --- | --- |
| Сортировка | Сложность (в среднем) | График |
| Пузырьком | O(n2) |  |
| Вставками | O(n2) |
| Выбором | O(n2) |
| Шелла | O(n log2n) |  |
| Быстрая | O(n log n) |  |
| Слиянием | O(n log n) |
| Кучей | O(n log n) |
| Поразрядная | O(n) |  |

Таблица 1. Теоретическая модель сложности алгоритмов

Основываясь на теоретических данных, можно отметить, что простые сортировки пузырьком, вставками и выбором являются самыми неудачными, и, при соответствии теории, на практике с массивами больших размеров время работы будет резко возрастать.

Следующие методы сортировки – Шелла, быстрая сортировка, слиянием и кучей – являются более удачными и на практике должны показать хорошие результаты даже при работе с большими данными.

Поразрядная сортировка, теоретически, является самой лучшей.

**Сложность алгоритмов на практике.**

В таблицах 2-5 указано время работы алгоритмов при входных данных – случайно сгенерированном массиве с определённым типом и количеством элементов. Проводились по 50 тестов для каждого случая (кроме тех, которые занимали очень много времени) и вычислялось среднее значение.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Тип массива | | Случайно сгенерированный | | | | |
| Тип элементов массива | | сhar (1 байт) | | | | |
| Количество элементов | | 50 | 500 | 5000 | 50000 | 500000 |
| Среднее время работы (в секундах) сортировки | Пузырьком | 10-6 | 10-4 | 0,07 | 10,09 | 824,527 ~ 14 мин |
| Вставками | 10-6 | 10-4 | 0,019 | 2,213 | 177,148 ~ 3 мин |
| Выбором | 10-6 | 10-4 | 0,026 | 3,03 | 245,586 ~ 4 мин |
| Шелла | 10-6 | 10-4 | 0,01 | 1,356 | 98,076 ~ 1,6 мин |
| Быстрая | 10-6 | 10-5 | 10-4 | 0,005 | 0,04 |
| Слиянием | 10-5 | 10-4 | 0,002 | 0,032 | 0,243 |
| Кучей | 10-6 | 10-4 | 0,001 | 0,021 | 0,212 |
| Поразрядная | 10-6 | 10-5 | 10-5 | 10-4 | 0,004 |

Таблица 2. Время работы алгоритмов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Тип массива | | Случайно сгенерированный | | | | |
| Тип элементов массива | | int (4 байта) | | | | |
| Количество элементов | | 50 | 500 | 5000 | 50000 | 500000 |
| Среднее время работы (в секундах) сортировки | Пузырьком | 10-6 | 10-4 | 0,07 | 10,734 | 986,851 ~ 16 мин |
| Вставками | 10-6 | 10-4 | 0,02 | 2,639 | 224,705 ~ 4 мин |
| Выбором | 10-6 | 10-4 | 0,054 | 7,802 | 361,773 ~ 6 мин |
| Шелла | 10-6 | 10-4 | 0,011 | 1,645 | 136,51 ~ 2 мин |
| Быстрая | 10-6 | 10-5 | 10-4 | 0,008 | 0,075 |
| Слиянием | 10-5 | 10-4 | 0,003 | 0,037 | 0,328 |
| Кучей | 10-6 | 10-4 | 0,002 | 0,028 | 0,308 |
| Поразрядная | 10-5 | 10-5 | 10-4 | 0,007 | 0,058 |

Таблица 3. Время работы алгоритмов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Тип массива | | Случайно сгенерированный | | | | |
| Тип элементов массива | | string (28 байт) | | | | |
| Количество элементов | | 50 | 500 | 5000 | 50000 | 500000 |
| Среднее время работы (в секундах) сортировки | Пузырьком | 0,001 | 0,12 | 28,2 | 1206,3 ~ 20 мин | 3 суток 3 ч |
| Вставками | 10-5 | 10-4 | 0,014 | 1,342 | 4ч 44 мин |
| Выбором | 10-4 | 0,026 | 7,8 | 567,33 ~ 10 мин | 15ч 11мин |
| Шелла | 10-4 | 0,003 | 0,121 | 0,85 | 691,463 ~ 11 мин |
| Быстрая | 10-4 | 0,003 | 0,08 | 0,463 | 10,524 |
| Слиянием | 10-4 | 0,014 | 0,362 | 2,305 | 30,203 |
| Кучей | 10-4 | 0,015 | 0,532 | 3,441 | 57,703 |
| Поразрядная | 0,326 | 3,407 | 18,817 | 26,7 | 119,685 ~ 2 мин |

Таблица 4. Время работы алгоритмов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Тип массива | | Случайно сгенерированный | | | | |
| Тип элементов массива | | struct date (12 байт) | | | | |
| Количество элементов | | 50 | 500 | 5000 | 50000 | 500000 |
| Среднее время работы (в секундах) сортировки | Пузырьком | 10-5 | 0,012 | 2,235 | 176,981 ~ 3 мин | 10514,4 ~ 2ч 55 мин |
| Вставками | 10-5 | 0,006 | 1,03 | 71,294 ~  1 мин | 4362,8 ~ 1ч 4 мин |
| Выбором | 10-5 | 0,004 | 0,7 | 57,627 ~  1 мин | 3619,42 ~ 1ч |
| Шелла | 10-5 | 0,002 | 0,347 | 29,37 | 2472,9 ~ 41 мин |
| Быстрая | 10-5 | 10-4 | 0,009 | 0,105 | 1,371 |
| Слиянием | 10-5 | 10-4 | 0,016 | 0,191 | 2,0 |
| Кучей | 10-5 | 10-4 | 0,016 | 0,168 | 1,87 |
| Поразрядная | 10-4 | 0,002 | 0,017 | 0,15 | 1,263 |

Таблица 5. Время работы алгоритмов

Во-первых, из данных таблиц можно сделать вывод о том, что динамика возрастания времени работы алгоритмов не зависит от типа данных. Для всех случаев она подчиняется единому закону.

Гистограмма 1.

Как видно из данных, представленных в таблицах и на гистограмме 1, простые сортировки пузырьком, вставками и выбором действительно не только проигрывают другим сортировкам по времени работы с большими массивами, но и в некотором случае, например, при сортировке массива типа string, становятся совсем непригодными для работы.

Гистограмма 2.

Сложные алгоритмы показали свою работоспособность с любыми типами данных даже на большом количестве элементов в массиве. Как и было сказано в теоретическом описании, сортировка Шелла из сложных – самая медленная.

Гистограмма 3.

Рассматривая гистограмму 3 сложности самых лучших алгоритмов, приходим к выводу, что сортировки слиянием кучей занимают почти одинаковое время. Оно имеют хорошую работоспособность, но всё ещё не являются самыми быстрыми.

Гистограмма 4.

Среди двух лидеров, для числового типа, поразрядная сортировка стала самой лучшей.

Таким образом, теоретическая модель сложности алгоритмов совпадает с практической.

Однако немного иначе дело обстоит с сортировкой массива строкового типа.

Во-первых, простые алгоритмы на больших массивах справились со своей задачей кране плохо. В случае пузырькового алгоритма отсчет пошел на сутки. Хотя, это соответствует теоретической модели.

По вышеуказанной причине, следующая гистограмма представляет данные таблицы 3 только до 50000 элементов.

Гистограмма 5.

Из диаграммы видно, что в этом случае из простых алгоритмов выделяется алгоритм вставки. Рассмотрим его вместе со сложными алгоритмами.

Гистограмма 5.

Как видно из гистограммы 5, самая быстрая сортировка для числового типа – поразрядная – оказалась отнюдь не самой быстрой для строкового типа. Далее мы разберем, что привело к такому исходу.

Гистограмма 6.

Сортировки слиянием и кучей, как с числовых типом, работают почти одинаково хорошо, но есть алгоритмы лучше.

Гистограмма 7.

Удивительно, что при работе со строковым типом простая сортировка вставками и сортировка Шелла стали одними их лидеров. Тем не менее, самой быстрой остается сортировка, которая так и называется – быстрая.

**Лидеры**

Алгоритмы сортировок пузырьком, вставками и выбором просты для понимания и написания кода, но совершенно неработоспособны на больших данных.

Сортировки слиянием и кучей работают почти одинаково хорошо с любыми типами данных и даже при большом количестве элементов массива остаются работоспособными и затрачивают времени меньше минуты.

Для работы с однобайтовым типом char и числовыми типами int и struct date самыми быстрыми являются быстрая и поразрядная сортировки.

Для типа string самой лучшей стала быстрая сортировка.

Таким образом, несмотря на то, что в теоретической модели поразрядная сортировка должна работать лучше всех, общим лидером для всех типов данных стала быстрая сортировка.

**Почему поразрядная сортировка не самая лучшая для работы со строковым типом?**

Поразрядная сортировка всё ещё остается самой быстрой для числовых типов, но она не так хорошо работает со строковым типом. Дело в том, что существуют два варианта этой сортировки: LSD – Least Significant Digit (перебор от младших разрядов к старшим) и MSD – Most Significant Digit (сначала анализируются старшие разряды). В первом варианте происходит сортировка по определённому разряду между всеми числами в массиве (рис. 1).

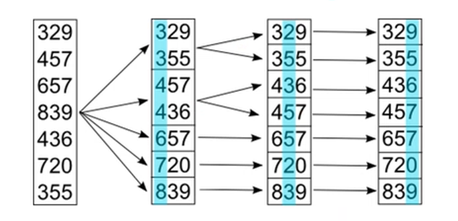


Рис. 1. LSD

Во втором случае сортировка по каждому разряду состоит из сортировки только набора элементов массива, предшествующий разряд которых имеет одинаковый символ (рис. 2).

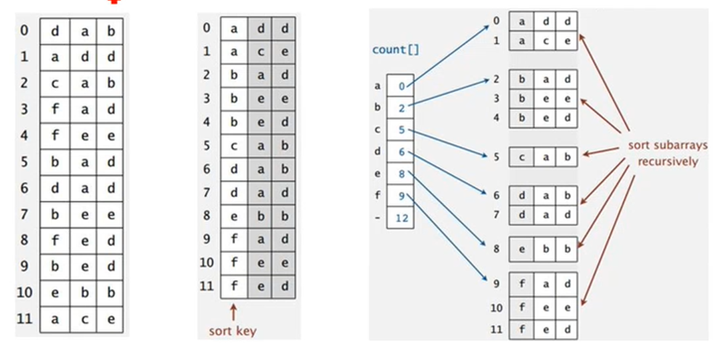


Рис. 2. MSD

Время на сортировку нескольких отдельных частей уходит больше, чем на сортировку целой части, так как алгоритм работает со счетчиками и временными массивами.

**Почему простая сортировка вставками стала одним из лидеров для строкового типа?**

Это связано с тем, что внутренний цикл алгоритма имеет условие:

a[j - 1] > a[j] ,

то есть цикл перейдёт к новому шагу, если просматриваемый элемент массива имеет больший вес, чем предыдущий. Для наших тестов строковый тип имеет диапазон от «a» до «z», а также пробел с весом 0, то есть строку можно представить в виде 27-тиричной системы счисления. Для такой системы вероятность, что некоторое число меньше данного, может быть больше, чем для 10-тиричной системе. Поэтому цикл в данном переходит к новому шагу быстрее.

Таким образом, простая сортировка вставками может оказаться эффективной для сортировки типов в большой системе счисления.

1. **Выводы**

Из проведенного анализа можно подвести итоги о том, что:

* теоретическая модель сложности алгоритмов совпадает с тем, что мы имеем на практике;
* несмотря на вышесказанное, некоторые алгоритмы могут работать лучше или хуже с определенными типами данных, например, строковым типом;
* простые алгоритмы не подходят для работы с большими данными;
* лучший универсальный метод – быстрая сортировка.

1. **Список литературы**

1. Васильев А.В. Алгоритмы и анализ сложности. Лекции. / ИВМиИТ КПФУ; 2021.

2. Know Thy Complexities! [Электронный ресурс]. —2021. — URL: https://www.bigocheatsheet.com (дата обращения 30.04.2021).

3. C++ reference [Электронный ресурс]. —2021. — URL: https://en.cppreference.com/w (дата обращения 26.04.202).

4. Geeks for geeks. A computer science portal for geeks. [Электронный ресурс]. —2021. — URL: https://www.geeksforgeeks.org (дата обращения 24.04.2020).

1. **Приложение**

date.cpp

#pragma once

#include <iostream>

using namespace std;

struct date

{

int d;

int m;

int y;

date& operator= (const date& d)

{

this->d = d.d;

this->m = d.m;

this->y = d.y;

return \*this;

}

friend bool operator> (const date& d1, const date& d2)

{

if (d1.y == d2.y && d1.m == d2.m && d1.d <= d2.d)

return false;

if (d1.y == d2.y && d1.m <= d2.m)

return false;

if (d1.y <= d2.y)

return false;

return true;

}

friend bool operator< (const date& d1, const date& d2)

{

if (d1.y == d2.y && d1.m == d2.m && d1.d >= d2.d)

return false;

if (d1.y == d2.y && d1.m >= d2.m)

return false;

if (d1.y >= d2.y)

return false;

return true;

}

friend ostream& operator<< (ostream& out, const date& d)

{

out << d.d << "." << d.m << "." << d.y;

return out;

}

};

Sorts.cpp

#include <iostream>

#include <string>

#include<ctime>

#include <queue>

#include "date.cpp"

#include <chrono>

using namespace std;

void bubble\_sort(char\* a, int n)

{

char t;

for (int k = n - 1; k > 0; k--)

for (int i = 0; i < n - 1; i++)

if (a[i] > a[i + 1])

{

t = a[i];

a[i] = a[i + 1];

a[i + 1] = t;

}

}

void bubble\_sort(int\* a, int n)

{

int t;

for (int k = n - 1; k > 0; k--)

for (int i = 0; i < n - 1; i++)

if (a[i] > a[i + 1])

{

t = a[i];

a[i] = a[i + 1];

a[i + 1] = t;

}

}

void bubble\_sort(string\* a, int n)

{

string t;

for (int k = n - 1; k > 0; k--)

{

for (int i = 0; i < n - 1; i++)

if (a[i] > a[i + 1])

{

t = a[i];

a[i] = a[i + 1];

a[i + 1] = t;

}

}

}

void bubble\_sort(date\* a, int n)

{

date t;

for (int k = n - 1; k > 0; k--)

{

for (int i = 0; i < n - 1; i++)

if (a[i] > a[i + 1])

{

t = a[i];

a[i] = a[i + 1];

a[i + 1] = t;

}

}

}

void insertion\_sort(char\* a, int n)

{

char t;

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

for (int j = i; j > 0 && a[j - 1] > a[j]; j--)

{

t = a[j - 1];

a[j - 1] = a[j];

a[j] = t;

}

}

void insertion\_sort(int\* a, int n)

{

int t;

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

for (int j = i; j > 0 && a[j - 1] > a[j]; j--)

{

t = a[j - 1];

a[j - 1] = a[j];

a[j] = t;

}

}

void insertion\_sort(string\* a, int n)

{

string t;

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

{

for (int j = i; j > 0 && a[j - 1] > a[j]; j--)

{

t = a[j - 1];

a[j - 1] = a[j];

a[j] = t;

}

}

}

void insertion\_sort(date\* a, int n)

{

date t;

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

for (int j = i; j > 0 && a[j - 1] > a[j]; j--)

{

t = a[j - 1];

a[j - 1] = a[j];

a[j] = t;

}

}

void choices\_sort(char\* a, int n)

{

char t;

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

for (int j = i + 1; j < n; j++)

if (a[i] > a[j])

{

t = a[i];

a[i] = a[j];

a[j] = t;

}

}

void choices\_sort(int\* a, int n)

{

int t;

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

for (int j = i + 1; j < n; j++)

if (a[i] > a[j])

{

t = a[i];

a[i] = a[j];

a[j] = t;

}

}

void choices\_sort(string\* a, int n)

{

string t;

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

{

for (int j = i + 1; j < n; j++)

{

if (a[i] > a[j])

{

t = a[i];

a[i] = a[j];

a[j] = t;

}

}

}

}

void choices\_sort(date\* a, int n)

{

date t;

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

for (int j = i + 1; j < n; j++)

if (a[i] > a[j])

{

t = a[i];

a[i] = a[j];

a[j] = t;

}

}

void Shell\_sort(char\* a, int n)

{

int i, j, k, increment;

char temp;

int swp = 0, comp = 0; increment = n / 2;

while (increment > 0)

{

for (i = 0;i < increment;i++)

{

for (j = 0;j < n;j += increment)

{

temp = a[j];

for (k = j - increment;k >= 0 && temp < a[k];k -= increment)

{

comp++;

swp++;

a[k + increment] = a[k];

}

a[k + increment] = temp;

swp++;

}

}

comp++;

if (increment / 2 != 0)

increment = increment / 2;

else if (increment == 1)

increment = 0;

else

increment = 1;

}

}

void Shell\_sort(int\* a, int n)

{

int i, j, k, increment;

int temp;

int swp = 0, comp = 0; increment = n / 2;

while (increment > 0)

{

for (i = 0;i < increment;i++)

{

for (j = 0;j < n;j += increment)

{

temp = a[j];

for (k = j - increment;k >= 0 && temp < a[k];k -= increment)

{

comp++;

swp++;

a[k + increment] = a[k];

}

a[k + increment] = temp;

swp++;

}

}

comp++;

if (increment / 2 != 0)

increment = increment / 2;

else if (increment == 1)

increment = 0;

else

increment = 1;

}

}

void Shell\_sort(string\* a, int n)

{

int i, j, k, increment;

string temp;

int swp = 0, comp = 0; increment = n / 2;

while (increment > 0)

{

for (i = 0;i < increment;i++)

{

for (j = 0;j < n;j += increment)

{

temp = a[j];

for (k = j - increment;k >= 0 && temp < a[k];k -= increment)

{

comp++;

swp++;

a[k + increment] = a[k];

}

a[k + increment] = temp;

swp++;

}

}

comp++;

if (increment / 2 != 0)

increment = increment / 2;

else if (increment == 1)

increment = 0;

else

increment = 1;

}

}

void Shell\_sort(date\* a, int n)

{

int i, j, k, increment;

date temp;

int swp = 0, comp = 0; increment = n / 2;

while (increment > 0)

{

for (i = 0;i < increment;i++)

{

for (j = 0;j < n;j += increment)

{

temp = a[j];

for (k = j - increment;k >= 0 && temp < a[k];k -= increment)

{

comp++;

swp++;

a[k + increment] = a[k];

}

a[k + increment] = temp;

swp++;

}

}

comp++;

if (increment / 2 != 0)

increment = increment / 2;

else if (increment == 1)

increment = 0;

else

increment = 1;

}

}

void quick\_sort(char\* a, int first, int last)

{

char mid, t;

int f = first, l = last;

mid = a[(f + l) / 2];

do

{

while (a[f] < mid)

f++;

while (a[l] > mid)

l--;

if (f <= l)

{

t = a[f];

a[f] = a[l];

a[l] = t;

f++;

l--;

}

} while (f < l);

if (first < l)

quick\_sort(a, first, l);

if (f < last)

quick\_sort(a, f, last);

}

void quick\_sort(int\* a, int first, int last)

{

int mid, t;

int f = first, l = last;

mid = a[(f + l) / 2];

do

{

while (a[f] < mid)

f++;

while (a[l] > mid)

l--;

if (f <= l)

{

t = a[f];

a[f] = a[l];

a[l] = t;

f++;

l--;

}

} while (f < l);

if (first < l)

quick\_sort(a, first, l);

if (f < last)

quick\_sort(a, f, last);

}

void quick\_sort(string\* a, int first, int last)

{

string mid, t;

int f = first, l = last;

mid = a[(f + l) / 2];

do

{

while (a[f] < mid)

f++;

while (a[l] > mid)

l--;

if (f <= l)

{

t = a[f];

a[f] = a[l];

a[l] = t;

f++;

l--;

}

} while (f < l);

if (first < l)

quick\_sort(a, first, l);

if (f < last)

quick\_sort(a, f, last);

}

void quick\_sort(date\* a, int first, int last)

{

date mid, t;

int f = first, l = last;

mid = a[(f + l) / 2];

do

{

while (a[f] < mid)

f++;

while (a[l] > mid)

l--;

if (f <= l)

{

t = a[f];

a[f] = a[l];

a[l] = t;

f++;

l--;

}

} while (f < l);

if (first < l)

quick\_sort(a, first, l);

if (f < last)

quick\_sort(a, f, last);

}

template <class Type>

void merge(Type\* a, int l, int m, int r)

{

int n1 = m - l + 1;

int n2 = r - m;

Type\* L = new Type[n1];

Type\* R = new Type[n2];

for (int i = 0; i < n1; i++)

L[i] = a[l + i];

for (int j = 0; j < n2; j++)

R[j] = a[m + 1 + j];

int i = 0;

int j = 0;

int k = l;

while (i < n1 && j < n2)

{

if (R[j] > L[i])

{

a[k] = L[i];

i++;

}

else

{

a[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

a[k] = L[i];

i++;

k++;

}

while (j < n2)

{

a[k] = R[j];

j++;

k++;

}

}

void merge\_sort(char\* a, int l, int r)

{

if (l < r)

{

int m = (l + r) / 2;

merge\_sort(a, l, m);

merge\_sort(a, m + 1, r);

merge(a, l, m, r);

}

}

void merge\_sort(int\* a, int l, int r)

{

if (l < r)

{

int m = (l + r) / 2;

merge\_sort(a, l, m);

merge\_sort(a, m + 1, r);

merge(a, l, m, r);

}

}

void merge\_sort(string\* a, int l, int r)

{

if (l < r)

{

int m = (l + r) / 2;

merge\_sort(a, l, m);

merge\_sort(a, m + 1, r);

merge(a, l, m, r);

}

}

void merge\_sort(date\* a, int l, int r)

{

if (l < r)

{

int m = (l + r) / 2;

merge\_sort(a, l, m);

merge\_sort(a, m + 1, r);

merge(a, l, m, r);

}

}

template <class Type>

void heapify(Type\* a, int k, int i)

{

Type t;

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < k && a[l] > a[largest])

largest = l;

if (r < k && a[r] > a[largest])

largest = r;

if (largest != i)

{

t = a[i];

a[i] = a[largest];

a[largest] = t;

heapify(a, k, largest);

}

}

void heap\_sort(char\* a, int n)

{

char t;

for (int i = n / 2 - 1; i >= 0; i--)

heapify(a, n, i);

for (int i = n - 1; i > 0; i--)

{

t = a[0];

a[0] = a[i];

a[i] = t;

heapify(a, i, 0);

}

}

void heap\_sort(int\* a, int n)

{

int t;

for (int i = n / 2 - 1; i >= 0; i--)

heapify(a, n, i);

for (int i = n - 1; i > 0; i--)

{

t = a[0];

a[0] = a[i];

a[i] = t;

heapify(a, i, 0);

}

}

void heap\_sort(string\* a, int n)

{

string t;

for (int i = n / 2 - 1; i >= 0; i--)

heapify(a, n, i);

for (int i = n - 1; i > 0; i--)

{

t = a[0];

a[0] = a[i];

a[i] = t;

heapify(a, i, 0);

}

}

void heap\_sort(date\* a, int n)

{

date t;

for (int i = n / 2 - 1; i >= 0; i--)

heapify(a, n, i);

for (int i = n - 1; i > 0; i--)

{

t = a[0];

a[0] = a[i];

a[i] = t;

heapify(a, i, 0);

}

}

template <class Type>

Type get\_max(Type\* a, int n)

{

Type max = a[0];

for (int i = 1; i < n; i++)

if (a[i] > max)

max = a[i];

return max;

}

template <class Type>

int get\_max\_size(Type\* a, int n)

{

int max = a[0].length();

for (int i = 1; i < n; i++)

{

if (a[i].length() > max)

max = a[i].length();

}

return max;

}

void count\_sort(char\* a, int n)

{

char\* output = new char[n];

int i;

int count[10] = { 0 };

int x = 0;

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

{

count[(int)(a[i] - '0')]++;

}

for (i = 1; i < 10; i++)

count[i] += count[i - 1];

for (i = n - 1; i >= 0; i--)

{

output[--count[(int)(a[i] - '0')]] = a[i];

}

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

{

a[i] = output[i];

}

delete[] output;

}

void radix\_sort(char\* a, int n)

{

count\_sort(a, n);

}

void count\_sort(int\* a, int n, int exp)

{

int\* output = new int[n];

int i, count[10] = { 0 };

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

count[(a[i] / exp) % 10]++;

for (i = 1; i < 10; i++)

count[i] += count[i - 1];

for (i = n - 1; i >= 0; i--)

{

output[--count[(a[i] / exp) % 10]] = a[i];

}

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

a[i] = output[i];

delete[] output;

}

void radix\_sort(int\* a, int n)

{

int m = get\_max(a, n);

for (int exp = 1; m / exp > 0; exp \*= 10)

count\_sort(a, n, exp);

}

void count\_sort(string\* a, int exp, int b, int e)

{

string\* output = new string[e - b];

int i;

int count[27] = { 0 };

int x = 0;

for (i = b; i < e; i++)

{

if (a[i].length() < exp + 1)

x = 0;

else

x = (int)(a[i][exp] - 'a') + 1;

count[x]++;

}

if (b < e)

for (i = 1; i < 27; i++)

count[i] += count[i - 1];

for (i = e - 1; i >= b; i--)

{

if (a[i].length() < exp + 1)

x = 0;

else

x = (int)(a[i][exp] - 'a') + 1;

output[--count[x]] = a[i];

}

x = 0;

for (i = b; i < e; i++, x++)

{

a[i] = output[x];

}

delete[] output;

}

void radix\_sort(string\* a, int n)

{

int m = get\_max\_size(a, n);

int b = 0, e = n;

int\* next = new int[27];

int i, j = 27;

int x;

for (int exp = 0; exp < m;)

{

if (b < e - 1)

count\_sort(a, exp, b, e);

if (j == 27)

{

for (i = 0; i < 27; i++)

next[i] = 0;

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

{

if (a[i].length() < exp + 1)

x = 0;

else

x = (int)(a[i][exp] - 'a') + 1;

next[x]++;

}

exp++;

j = 0;

e = next[j++];

}

do

{

b = e; e += next[j++];

} while (b >= e - 1 && j < 27);

}

}

void count\_sort\_date\_year(date\* a, int n, int exp)

{

date\* output = new date[n];

int i, count[10] = { 0 };

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

count[(a[i].y / exp) % 10]++;

for (i = 1; i < 10; i++)

count[i] += count[i - 1];

for (i = n - 1; i >= 0; i--)

{

output[--count[(a[i].y / exp) % 10]] = a[i];

}

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

a[i] = output[i];

delete[] output;

}

void count\_sort\_date\_month(date\* a, int exp, int b, int e)

{

date\* output = new date[e - b];

int i, count[10] = { 0 };

for (i = b; i < e; i++)

count[(a[i].m / exp) % 10]++;

for (i = 1; i < 10; i++)

count[i] += count[i - 1];

for (i = e - 1; i >= b; i--)

{

output[--count[(a[i].m / exp) % 10]] = a[i];

}

int x = 0;

for (i = b; i < e; i++, x++)

{

a[i] = output[x];

}

delete[] output;

}

void count\_sort\_date\_day(date\* a, int exp, int b, int e)

{

date\* output = new date[e - b];

int i, count[10] = { 0 };

for (i = b; i < e; i++)

count[(a[i].d / exp) % 10]++;

for (i = 1; i < 10; i++)

count[i] += count[i - 1];

for (i = e - 1; i >= b; i--)

{

output[--count[(a[i].d / exp) % 10]] = a[i];

}

int x = 0;

for (i = b; i < e; i++, x++)

{

a[i] = output[x];

}

delete[] output;

}

void radix\_sort(date\* a, int n)

{

int exp;

for (exp = 1; exp < 1000; exp \*= 10)

{

count\_sort\_date\_year(a, n, exp);

}

queue<int> count;

for (int i = 0, y = -1; i < n; i++)

{

if (a[i].y > y)

{

count.push(1);

y = a[i].y;

}

else

count.back()++;

}

int b, e = 0;

for (exp = 1; exp < 100;)

{

do

{

b = e; e += count.front();

count.push(count.front());

count.pop();

} while (b >= e - 1 && e < n);

if (b < e - 1)

count\_sort\_date\_month(a, exp, b, e);

if (e >= n)

{

exp \*= 10;

e = 0;

}

}

queue<int> c;

count = c;

for (int i = 0, y = -1, m = -1; i < n; i++)

{

if (a[i].y > y)

{

m = -1;

y = a[i].y;

}

if (a[i].m > m)

{

count.push(1);

m = a[i].m;

}

else

count.back()++;

}

e = 0;

for (exp = 1; exp < 100;)

{

do

{

b = e; e += count.front();

count.push(count.front());

count.pop();

} while (b >= e - 1 && e < n);

if (b < e - 1)

count\_sort\_date\_day(a, exp, b, e);

if (e >= n)

{

exp \*= 10;

e = 0;

}

}

}