Сортировка

TimSort

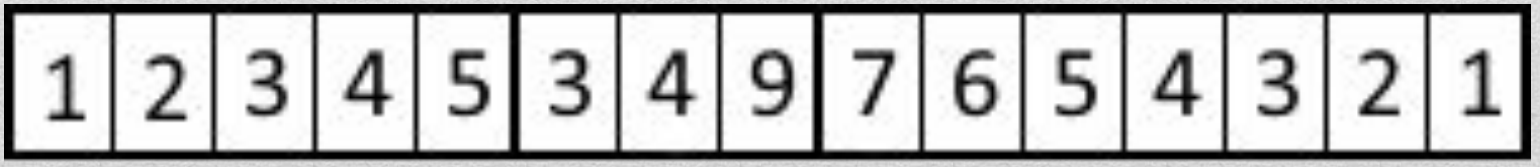
Общие сведения

Timsort — гибридный алгоритм сортировки, сочетающий сортировку вставками и сортировку слиянием, опубликованный в 2002 году Тимом Петерсоном

Описание алгоритма сортировки

Основная идея алгоритма в том, что в реальном мире сортируемые массивы данных часто содержат в себе упорядоченные подмассивы. На таких данных Timsort существенно быстрее многих алгоритмов сортировки

Понятие упорядоченного подмассива



Алгоритм

- По специальному алгоритму входной массив разделяется на подмассивы.
- Каждый подмассив сортируется сортировкой вставками.
- Отсортированные подмассивы собираются в единый массив с помощью модифицированной сортировки слиянием.

Используемые понятия:

- **N** — размер входного массива
- **run** — упорядоченный подмассив во входном массиве. Причём упорядоченный либо нестрого по возрастанию, либо строго по убыванию.
- **minrun** — это минимальный размер упорядоченного подмассива.

Шаг 0. Вычисление minrun.

- оно не должно быть слишком большим, поскольку к подмассиву размера minrun будет в дальнейшем применена сортировка вставками.
- Оно не должно быть слишком маленьким, поскольку чем меньше подмассив — тем больше итераций слияния подмассивов придётся выполнить на последнем шаге алгоритма
- Оптимальная величина для **N/minrun** это степень числа 2 (или близким к нему).

Шаг 1. Разбиение на подмассивы и их сортировка

- Указатель текущего элемента ставится в начало входного массива.
- Начиная с текущего элемента, в этом массиве идёт поиск упорядоченного подмассива **run**.
- Если размер текущего **run'a** меньше чем **minrun** — выбираются следующие за найденным **run**-ом элементы в количестве **minrun** — **size(run)**.
- К данному подмассиву применяется сортировка вставками.
- Указатель текущего элемента ставится на следующий за подмассивом элемент.
- Если конец входного массива не достигнут — переход к пункту 2, иначе — конец данного шага

Шаг 2. Слияние

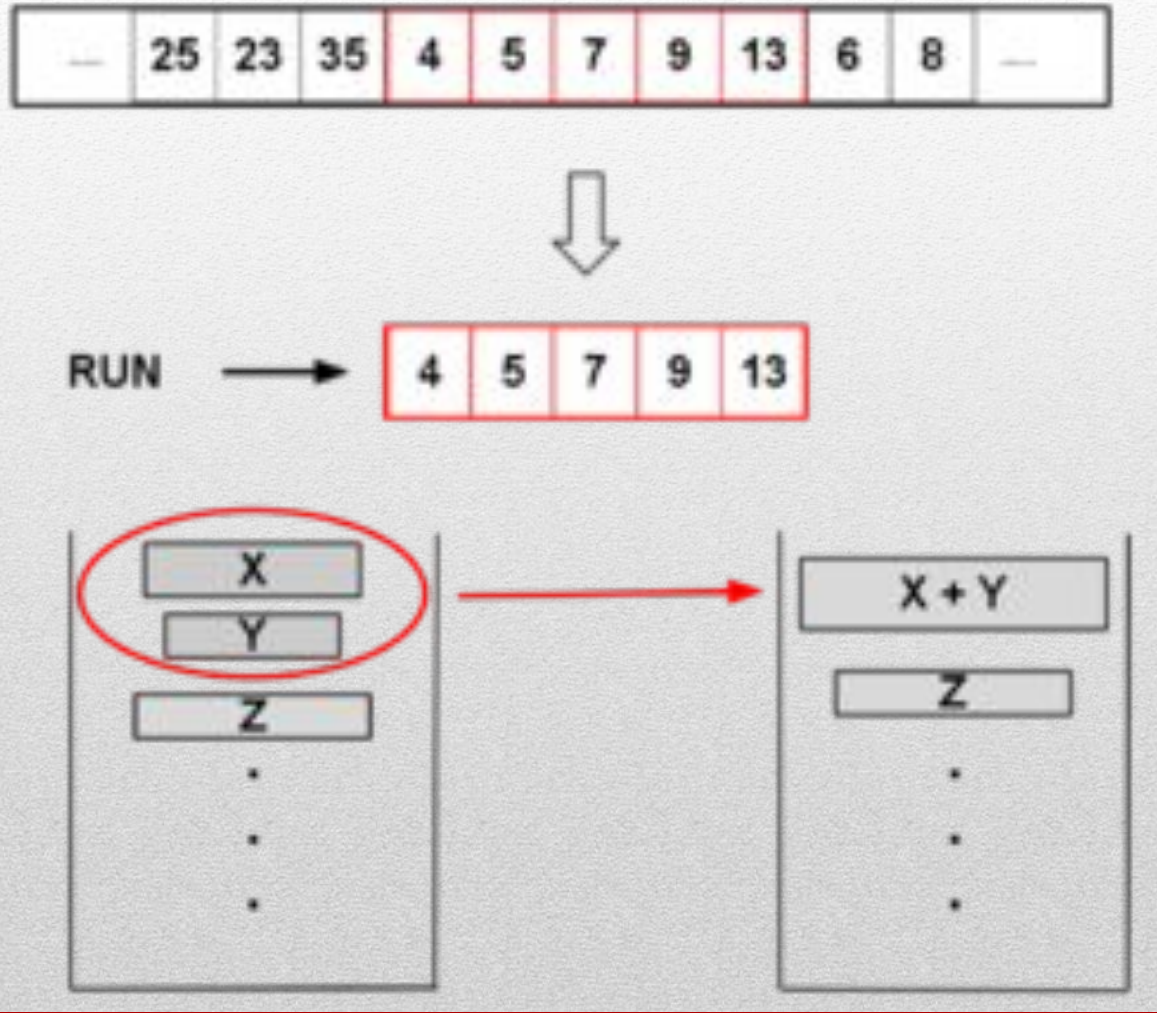
- Объединять подмассивы примерно равного размера
- Сохранить стабильность алгоритма — то есть не делать бессмысленных перестановок.

Алгоритм:

- Создается пустой стек пар <индекс начала подмассива>-<размер подмассива>.
- Берется первый упорядоченный подмассив.
- В стек добавляется пара данных <индекс начала>-<размер> для текущего подмассива.
- Определяется, нужно ли выполнять процедуру слияния текущего подмассива с предыдущими. Для этого проверяется выполнение двух правил (пусть **X**, **Y** и **Z** — размеры трёх верхних в стеке подмассивов):

$$X > Y + Z \qquad Y > Z$$

Шаг 2.1 Слияние



Шаг 2.2 Слияние

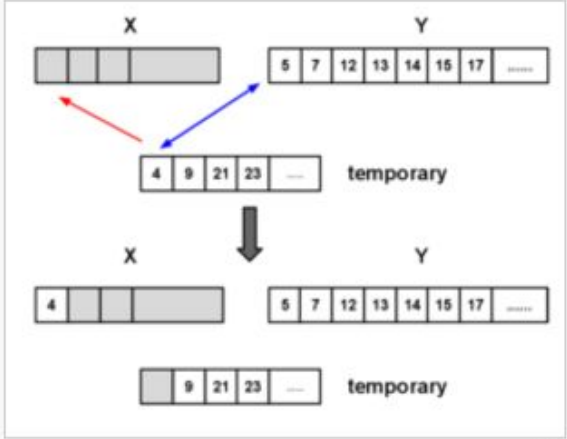
- Если одно из правил нарушается — массив **Y** сливается с меньшим из массивов **X** и **Z**. Повторяется до выполнения обоих правил или полного упорядочивания данных.
- Если еще остались не рассмотренные подмассивы — берется следующий и переходим к пункту 2. Иначе — конец

В идеальном случае:

есть подмассивы размера 128, 64, 32, 16, 8, 4, 2, 2. В этом случае никаких слияний не будет выполнятся пока не встретятся 2 последних подмассива, после чего будут выполнены 7 идеально сбалансированных слияний

Процедура слияния подмассивов

- Создается временный массив в размере меньшего из соединяемых подмассивов.
- Меньший из подмассивов копируется во временный массив
- Указатели текущей позиции ставятся на первые элементы большего и временного массива.
- На каждом следующем шаге рассматривается значение текущих элементов в большем и временном массивах, берется меньший из них и копируется в новый отсортированный массив. Указатель текущего элемента перемещается в массиве, из которого был взят элемент.
- Пункт 4 повторяется, пока один из массивов не закончится.
- Все элементы оставшегося массива добавляются в конец нового массива.



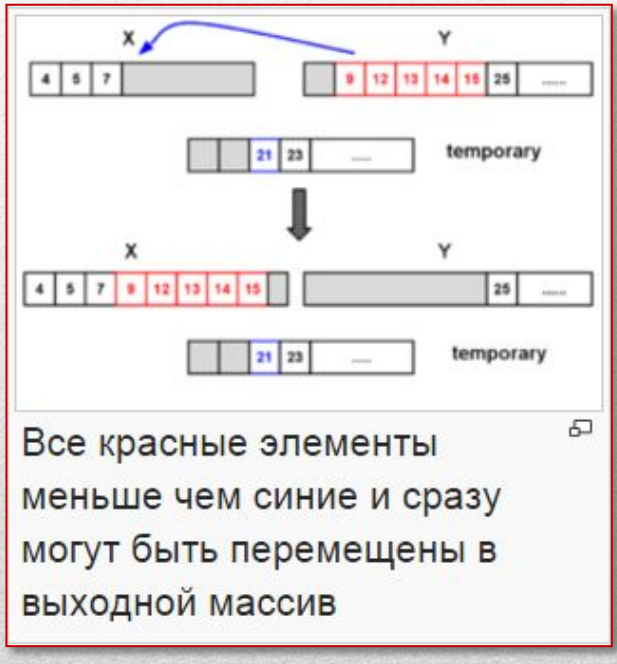
Элемент временного массива(выделенный голубой стрелкой) сравнивается с наименьшим элементом большого массива и меньший из них перемещается в новый отсортированный массив (как показано красной стрелкой).

Модификация процедуры слияния подмассивов (Galloping Mode)

A = {1, 2, 3,..., 9999, 10000}

B = { 20000, 20001, ..., 29999, 30000}

- Начинается процедура слияния, как было показано выше.
- На каждой операции копирования элемента из временного или большего подмассива в результирующий запоминается, из какого именно подмассива был элемент.



Все красные элементы меньше чем синие и сразу могут быть перемещены в выходной массив

Модификация процедуры слияния подмассивов (Galloping Mode)

- Если уже некоторое количество элементов (в данной реализации алгоритма это число равно 7) было взято из одного и того же массива — предполагается, что и дальше нам придётся брать данные из него. Чтобы подтвердить эту идею, алгоритм переходит в режим «галопа», то есть перемещается по массиву-претенденту на поставку следующей большой порции данных бинарным поиском (массив упорядочен) текущего элемента из второго соединяемого массива.
- В момент, когда данные из текущего массива-поставщика больше не подходят (или был достигнут конец массива), данные копируются целиком.

Пример использования Galloping Mode

A = {1, 2, 3,..., 9999, 10000}
B = { 20000, 20001, ..., 29999, 30000}

Первые 7 итераций сравниваются числа 1, 2, 3, 4, 5, 6 и 7 из массива A с числом 20000, так как 20000 больше — элементы массива A копируются в результирующий. Начиная со следующей итерации алгоритм переходит в режим «галопа»: сравнивает с числом 20000 последовательно элементы 8, 10, 14, 22, 38, n+2^i, …, 10000 массива A. (~log2 N сравнений). После того как конец массива A достигнут и известно, что он весь меньше B, нужные данные из массива A копируются в результирующий

Оценка

TimSort является одной из самых быстрых и удобных сортировок, в лучшем случае работая за время n, в худшем за n*log(n)

Если же сравнивать TimSort с QuickSort, то первая почти всегда будет работать быстрее, исключая те случаи, когда нам дан совсем небольшой массив без единого упорядоченного подмассива, в таком случае QuickSort сработает эффективнее

Name	Best	Average	Worst	Memory	Stable
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	Depends
Merge sort	$n \log n$	$n \log n$	$n \log n$	Depends	Yes
In-place Merge sort	—	—	$n (\log n)^2$	1	Yes
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No
Insertion sort	n	n^2	n^2	1	Yes
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No
Selection sort	n^2	n^2	n^2	1	Depends
Timsort	n	$n \log n$	$n \log n$	n	Yes
Shell sort	n	$n(\log n)^2$	$O(n\log^2 n)$	1	No
Bubble sort	n	n^2	n^2	1	Yes
Binary tree sort	n	$n \log n$	$n \log n$	n	Yes
Cycle sort	—	n^2	n^2	1	No
Library sort	—	$n \log n$	n^2	n	Yes
Patience sorting	—	—	$n \log n$	n	No
Smoothsort	n	$n \log n$	$n \log n$	1	No
Strand sort	n	n^2	n^2	n	Yes
Tournament sort	—	$n \log n$	$n \log n$		
Cocktail sort	n	n^2	n^2	1	Yes
Comb sort	—	—	n^2	1	No
Gnome sort	n	n^2	n^2	1	Yes
Bogosort	n	$n \cdot n!$	$n \cdot n! \to \infty$	1	No

Спасибо за внимание!