Сортировка Timsort

Для вопросов по курсу:

- Почта дисциплины - alg@moevm.pro

Префикс в теме письма [ALG_53XX]

Что такое Timsort?

Timsort — гибридный устойчивый алгоритм сортировки, сочетающий различные подходы.

На массивах данных, которые содержат упорядоченные подмассивы, данный алгоритм показывает себя намного лучше других сортировок.

Свойства TimSort:

- устойчивый;
- лучший случай: O(n);
- средний и худший: O(nlogn);
- дополнительная память: O(n);
- стек для слияния массивов: O(logn).

Зачем Timsort?

Классические сортировки:

- быстрая сортировка — O(nlogn) в среднем, неустойчива и плохо работает на некоторых данных;
- сортировка слиянием — O(nlogn), устойчива, но всегда делает много слияний;
- сортировка вставками — быстрая на маленьких или почти отсортированных массивах, но O(n^2) на больших.

Идея TimSort:

- использовать естественные упорядоченные фрагменты — runs;
- маленькие runs досортировывать вставками;
- крупные runs сливать по специальным правилам;
- на почти упорядоченных данных ускоряться с помощью галопа.

Идея алгоритма

- Шаг 0. Определение параметров сортировки (minrun)
- Шаг 1. Входной массив разделяется на подмассивы, которые сортируются сортировкой вставками*, сортировкой пузырьком или любой другой устойчивой сортировкой.
- Шаг 2. Отсортированные подмассивы объединяются в один массив с помощью модифицированного алгоритма слияния.

Шаг 0. Определение параметров сортировки (вычисление minrun)

- Число minrun определяется на основе n.
- Слишком большое → сортировка вставками неэффективна.
- Слишком маленькое → много итераций слияния.
- Оптимальная величина для n/minrun — степень двойки, т.к. слияние подмассивов наиболее эффективно работает на подмассивах примерно равного размера.

Шаг 0. Определение параметров сортировки (вычисление minrun)

- Автором алгоритма было выбрано оптимальное значение, которое принадлежит диапазону [32;65)
- Исключение: если n<64, тогда n=minrun и Timsort превращается в сортировку вставками.
- Берем старшие 6 бит числа n и добавляем единицу, если в оставшихся младших битах есть хотя бы один ненулевой.

```
int minRunLength(n):
  flag = 0          //
  while (n ⩾ 64)
    flag |= n & 1
    n >>= 1
  return n + flag
```

Подготовка к Шагу 1. Определение Run (естественные упорядоченные блоки)

- Run — максимальный подмассив, который уже находится в нужном порядке или в обратном порядке.
- Для обычной сортировки по возрастанию:
  - неубывающий run: $a_i \leq a_{i+1}$;
  - убывающий run: $a_i > a_{i+1}$
- Для убывающего run условие строгое (>), чтобы:
  - не менять порядок равных элементов (важно для устойчивости);
  - убывающий run развернуть и получить неубывающий.

Шаг 1. Разбиение на подмассивы

- Начиная с первого элемента, во входном массиве выделяются runs (упорядоченные подмассивы).
- Если длина очередного найденного run меньше minrun
  - расширить его до minrun (“добавить” недостающие элементы справа)
  - полученный run сортировать *сортировкой вставками**
- Длина run может быть больше minrun, если упорядоченность сохраняется дальше (= расширяем run “до упора”)
- Если получившийся подмассив убывающий → переставляем, чтобы стал неубывающий.
- На выходе: отсортированный подмассив размером ≥minrun.

Шаг 1. Разбиение на подмассивы. Пример 1

Пусть для простоты minrun = 3

a[i]	10	9	8	11	12	13	14	5	7	0
i	0	1	2	3	4	5	6	7	8	9

Шаг 1. Разбиение на подмассивы. Пример 2

Пусть для простоты minrun = 3

a[i]	0	1	2	3	4	5	6	7	8	9
i	0	1	2	3	4	5	6	7	8	9

Подготовка к Шагу 2. Стек runs

- В рамках шага 1 (разбиение на подмассивы) происходит формирование стека полученных подмассивов (runs):
  - хранится индекс начала run в массиве и длина run
  - при получении нового run
    - добавляем запись в стек
    - проверяем инварианты

Подготовка к Шагу 2. Стек runs. Пример

Пусть для простоты minrun = 3

a[i]	10	9	8	11	12	13	14	5	7	0
i	0	1	2	3	4	5	6	7	8	9

Подготовка к Шагу 2. Инварианты

- Инварианты — условия для значений стека runs (“метаданных” о run)
  - проверяются 3 верхние элемента стека
  - X, Y, Z — верхний (последний добавленный), второй и третий соответственно
- Условия касаются *размеров* run и требуют убывания размеров в стеке:
  1. Z > X + Y
  2. Y > X
- Если нарушается
  - условие 1 — подмассив Z слишком маленький, сливаем с Y (т.к. это соседний подмассив)
  - условие 2 — подмассив X слишком велик, сливаем 2 других
- Сохранение инвариантов (“исправление” нарушенных условий):
  - слить Y с меньшим из X и Z
- После выявлениях всех runs, сливаем оставшиеся в стеке по исходному алгоритму с доп. условием: если размер стека = 2 → слить Y и X



Подготовка к Шагу 2. Инварианты. Пример

Пусть для простоты minrun = 3

a[i]	10	9	8	11	12	13	14	5	7	0
i	0	1	2	3	4	5	6	7	8	9

Стек (индекс; длина):

- 9; **1** (вершина)
- 7; **2**
- 3; **4**
- 0; **3**

Подготовка к Шагу 2. Режим галопа

- Пусть есть два отсортированных подмассива:
  - A=[1,2,…,100]
  - B=[200,201,…,300]
- Обычное слияние сравнит много элементов, хотя очевидно, что весь A должен идти перед B.

Идея галопа:

- если из одного подмассива подряд взято много элементов → предполагаем, что так будет и дальше;
- с помощью модифицированного бинарного поиска находим, сколько элементов этого подмассива нужно перенести;
  - поиск O(logn) выигрывает против линейного сравнения O(n)

Подготовка к Шагу 2. Поиск границы при слиянии

- Задача (из идеи галопа): найти в отсортированном массиве индекс первого элемента, большего чем некоторое значение T.
- Например, при слиянии массивов A=[1,2,…,100] и B=[200,201,…,300], i раз подряд взяли элементы только из массива A
- Нужно найти, начиная с последней позиции (i), элемент с индексом j, до которого элементы массива A будут меньше элемента из B
- Тогда слияние A и B можно провести как: A[:i+1] (первые i шагов) + A[i+1:j] + слияние оставшейся части A[j:] и B
    - Подмассив A[i:j] помещается в результирующий сразу, без лишних сравнений с элементом из B

Подготовка к Шагу 2. Galloping Search

Дан отсортированный массив A и элемент T
Нужно найти подмассив A[0:j] такой, что A[j-1] <= T < A[j]

1. Быстро ищем элемент, для которого выполняется строгая часть условия: **T < A[k]**
   - “Быстро” — проверяем элементы с удвоением шага: A[2*i] (i=1,2,3…), т.е. сравниванием с A[1], A[3], A[5], A[9], …
   - Если нашли элемент, удовлетворяющий условию — запомнили текущий индекс k и предыдущий k_prev.
   - Если вышли за границы массива — k = размер массива (т.к. все элементы массива < T)
2. Если k = len(a), то искомый подмассив A[0:j] = A
3. Иначе осуществляем бинарный поиск T на найденном отрезке [k_prev; k]

Преимущество: полный бинарный поиск в массиве A всегда тратит log(len(A))

Подготовка к Шагу 2. Galloping Search. Пример

Пусть порог для галопа = 3

a[i]	0	2	10	14	15	16	17	18	19
i	0	1	3	4	5	6	7	8	9

b[i]	1	2	3	4	5	6	7	8	9
i	0	1	2	3	4	5	6	7	8

Шаг 2. Слияние массивов

Уточнение: слияние производится при нарушении инвариантов, которое происходит при добавлении нового run в стек, т.е. строго говоря, шаг 2 выполняется не *после* шага 1, а во время него при определенных условиях.

Объединяя рассмотренные ранее положения, получим следующую последовательность действий:

1. Смотрим последние 3 элемента в стеке (X,Y,Z) и проверяем инварианты
2. При нарушении инварианта соединяем Y с min(X,Z) и возвращаем в стек
    a. Активируем режим галопа, если выполняется условие при слиянии
3. Когда сливаем 2 последних подмассива в стеке, получаем отсортированный массив 😎😎😎

Сложность сортировки

	Время	Память	Устойчивая
Худший случай	n*log(n)	O(n)	Да
Средний случай	n*log(n)		
Лучший случай	n		

n - объем входных данных

Почему такая сложность?

Время:

В лучшем случае отсортированного массива - установим этот факт за O(n).

В остальных случаях при подборе хорошего размера подмассива логарифмическая сложность обуславливается применением алгоритма слияния.

На малых объемах данных (размере подмассива) сортировка вставками работает достаточно быстро (приемлемо).

Память:

Необходимо сохранить каждый элемент (n).

n - объем входных данных

Вопросы по курсу можно задавать:

Почта дисциплины - alg@moevm.pro
Иванов Дмитрий Владимирович - dmitry.ivanov@moevm.pro