1 Сортировка слиянием. Сортировка слиянием с $\mathcal{O}(\log N)$ памяти (стабильный вариант за $\mathcal{O}(N \log^2 N)$).

1.1 Определение

Def. Сортировка слиянием (англ. *Merge sort*) — алгоритм, выполняющий сортировку массивы засчёт слияния отсортированных подмассивов . Алгоритм использует $\mathcal{O}(N)$ дополнительной памяти и работает за $\mathcal{O}(N \log N)$) времени.

1.2 Классический вариант за $\mathcal{O}(N)$ доп. памяти

1.2.1 Принцип работы:

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.

1.2.2 Процедура слияния(Merge):

У нас есть два отсортированных массива $\boldsymbol{a}$ и $\boldsymbol{b}$. Нам надо получить из них же отсортированный массив, но уже с размером $|\boldsymbol{a}|+|\boldsymbol{b}|$. Для этого можно применить процедуру слияния. Эта процедура заключается в том, что мы сравниваем элементы массивов (начиная с начала) и меньший из них записываем в финальный. И затем, в массиве у которого оказался меньший элемент, переходим к следующему элементу и сравниваем теперь его. В конце, если один из массивов закончился, мы просто дописываем в финальный другой массив. После мы наш финальный массив записываем заместо двух исходных и получаем отсортированный участок.

Псевдокод:

```
def merge(A, B, C : int[n]):
    i_a = i_b = 0
    while i_a < A and i_b < B:
        if A[i_a] ≤ B[i_b]:
            C[i_c] = A[i_a]
            i_a += 1, i_c += 1
        else:
            C[i_c] = B[i_b]
            i_b += 1, i_c += 1
    while i_a < |A|:
        C[i_c] = A[i_a]
        i_a += 1, i_c += 1
    while i_b < |B|:
        C[i_c] = B[i_b]
        i_b += 1, i_c += 1
```

1.2.3 Рекурсивный алгоритм:

```
def mergeSort(A : int[n]; left, right : int):
    if right − left ≤ 1
        return
    mid = (left + right) / 2
    mergeSort(A, left, mid)
    mergeSort(A, mid, right)
    merge(A, left, mid, right)
```

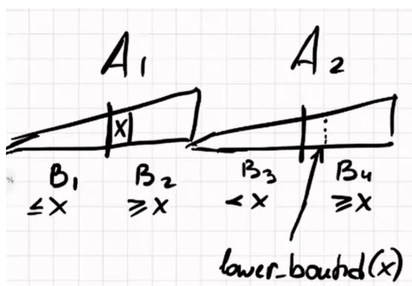
1.3 Стабильный вариант за $\mathcal{O}(N \log^2 N)$

1.3.1 Поменяется только процецура слияния(Merge → InplaceMerge):

```
def inplaceMerge(A₁, A₂ : int[n]):
    if |A₁| == 0 or |A₂| == 0:
        return
    if |A₁| == 1 and |A₂| == 1:
        if A₁[0] > A₂[0]:
            swap(A₁[0], A₂[0])
    m = |A₁|
    B₁ = A₁[0, ..., m/2)
    B₂ = A₁[m/2, ..., m)
    x = B₂[0]
    B₃ = A₂[m, ..., l_b(x))
    B₄ = A₂[l_b(x), ..., n)
    rotate(B₂, B₃)
    inplaceMerge(B₁, B₃)
    inplaceMerge(B₂, B₄)
```

Комментарий:

Если $|A_1| < |A_2|$, то ищём центральный элемент **x** уже в правом массиве, а в левом берём upper_bound(**x**).



Пояснительный рисуночек 1.