Pembahasan Sofita Main Saham

Deskripsi Singkat

Sofita mengurutkan seluruh saham yang ada dari yang memiliki **harga termurah hingga harga termahal**. Apabila terdapat saham yang memiliki harga yang sama, maka Sofita akan mengurutkan saham-saham tersebut berdasarkan **nomor seri dari yang terkecil hingga terbesar**. Setelah diurutkan, **saham yang paling menguntungkan adalah saham yang berada di tengah urutan tersebut**.

Sehingga, apabila saat ini terdapat Y saham, maka:

- Jika Y merupakan bilangan ganjil, maka saham paling menguntungkan adalah saham yang memiliki nilai saham ke-$\lceil \frac{Y}{2} \rceil$ jika diurutkan dari harga termurah hingga harga termahal.
- Jika Y merupakan bilangan genap, maka saham paling menguntungkan adalah saham yang memiliki nilai saham ke-$\frac{Y}{2} + 1$ jika diurutkan dari harga termurah hingga harga termahal.

Pemilik saham Warungpedia akan melakukan Q perintah. Terdapat 2 macam perintah, yaitu:

1. **TAMBAH [C]**

   Ditambahkan saham baru seharga C dengan nomor seri saham bernilai $K + 1$ (K adalah banyak saham sebelum penambahan saham tersebut).

2. **UBAH [X] [C]**

   Harga saham dengan nomor seri X diubah menjadi seharga C.

Baik ketika **saham selesai ditambah maupun diubah**, program akan menampilkan **nomor seri** milik saham yang **paling menguntungkan** dari seluruh harga saham tersebut.

DISCLAIMER

Solusi yang akan dijelaskan mengasumsikan bahwa **Anda sudah memiliki pemahaman mengenai Heap** (baik MaxHeap maupun MinHeap) dan metode-metode umumnya seperti **peek(), poll(), insert(), remove(), dan balancing**.

INISIASI AWAL

Permasalahan ini mengharuskan kita untuk bisa mendapatkan nilai median setiap kali *query* TAMBAH dan UBAH dilaksanakan. Karena jumlah *query* maksimal adalah 10^6, maka penggunaan struktur data array dan juga pemanggilan method $Collections.sort()$ setiap kali *query* TAMBAH dan UBAH dipanggil tentunya tidak akan efisien untuk *problem* ini.

Untuk bisa menyelesaikan permasalahan ini dengan efisien, kita bisa memanfaatkan struktur data MaxHeap dan juga MinHeap untuk mendapatkan nilai median dari saham. Caranya adalah dengan membuat 1 buah MaxHeap dan 1 buah MinHeap sebagai berikut:

1. `lowerHeap` → **MaxHeap** yang berisikan setengah saham dengan nilai **terkecil**.
2. `upperHeap` → **MinHeap** yang berisikan setengah saham dengan nilai **terbesar**.

Contohnya, ketika kita memiliki data berupa [1, 2, 3, 4, 5, 6], maka `lowerHeap` akan menyimpan [1, 2, 3] dan `upperHeap` akan menyimpan [4, 5, 6].

Agar ide di atas bisa tercapai, maka kita harus bisa menjaga keseimbangan antara kedua Heap (`lowerHeap` dan `upperHeap`). Kedua Heap dikatakan seimbang apabila keduanya memiliki jumlah saham yang sama atau jumlah saham salah satu heap "lebih banyak 1 buah" dibandingkan jumlah saham Heap lainnya. Atau secara matematis, keseimbangan dicapai ketika

$$|size(MinHeap) - size(MaxHeap)| \leq 1$$

dengan method `size()` menyatakan jumlah saham di Heap tersebut. **Penjelasan cara memasukkan data awal ke dalam lowerHeap maupun upperHeap dapat dilihat pada penjelasan *query* TAMBAH.**

Selanjutnya, jika kedua Heap memiliki jumlah saham yang sama, maka nilai median akan sama dengan nilai head dari `upperHeap`/`MinHeap`. Sementara itu, jika salah satu Heap memiliki jumlah saham yang lebih banyak 1 buah dari Heap lainnya, maka nilai median akan sama dengan nilai head dari Heap tersebut (Heap yang nilai sahamnya lebih banyak).

```
if (upperHeap.size >= lowerHeap.size) {
   out.println(upperHeap.peek().nomorSeri);
} else {
   out.println(lowerHeap.peek().nomorSeri);
}
```

Apabila kita berhasil menjaga keseimbangan dari jumlah data di `lowerHeap` dan `upperHeap`, maka untuk bisa mendapatkan nilai median dari saham, kompleksitas yang dibutuhkan hanyalah $O(1)$.

QUERY TAMBAH [C]

Query TAMBAH ini akan memiliki 1 buah parameter yaitu C yang merupakan harga dari saham baru yang akan ditambahkan ke daftar Saham Warung Pedia. Untuk menambahkan saham ini, kita hanya perlu melakukan operasi berikut:

1. **Tambahkan sahamBaru ke lowerHeap atau ke upperHeap**
   Apabila harga sahamBaru > harga dari *head* upperHeap, maka tambahkan sahamBaru ke upperHeap. Sementara jika lebih rendah, maka tambahkan sahamBaru ke lowerHeap.

```
Saham sahamBaru = new Saham(daftarSaham.size() + 1, in.nextInt());
Saham upperHeapHead = upperHeap.peek();
if (sahamBaru.harga > upperHeapHead.harga) {
   upperHeap.insert(sahamBaru);
} else {
   lowerHeap.insert(sahamBaru);
}
```

2. **Lakukan *Balancing* Antar Heap**
   Cek apakah keseimbangan lowerHeap dan upperHeap masih terjaga. Jika keseimbangan kedua Heap tidak terjaga, maka *balancing* perlu dilakukan. *Balancing* akan memindahkan *head* dari Heap yang jumlah sahamnya lebih banyak, ke Heap yang jumlah sahamnya lebih sedikit.

```
public static void balance() {
   if (lowerHeap.size - upperHeap.size >= 2) {
       upperHeap.insert(lowerHeap.poll());
   } else if (upperHeap.size - lowerHeap.size >= 2) {
       lowerHeap.insert(upperHeap.poll());
   }
}
```

Seluruh proses di atas memiliki kompleksitas paling besar $O(3\ log\ N) = O(log\ N)$, dengan N merupakan jumlah saham di dalam Heap

QUERY UBAH [X] [C]

Query UBAH akan menerima 2 parameter yaitu X yang merupakan nomor seri saham yang nilainya akan berubah dan C yang merupakan harga saham yang baru. Untuk melakukan *query* ini, kita hanya perlu melakukan beberapa operasi berikut :

1. **Remove saham yang ingin diubah dari Heapnya**
   Cek apakah posisi saham yang ingin kita *remove* berada di lowerHeap atau upperHeap. Untuk mengecek posisi saham, kita hanya perlu membandingkan saham dengan head dari lowerHeap. Apabila harga awal saham $\leq$ harga dari *head* lowerHeap, maka posisi saham berada di lowerHeap. Jika tidak, maka posisi Saham berada di upperHeap. Selanjutnya, *remove* saham dari Heap-nya. Perhatikan bahwa posisi saham tidak selalu berada di *head* dari Heap.

```
int nomorSeri = in.nextInt();
Saham sahamTarget = daftarSaham.get(nomorSeri);
Saham lowerHeapHead = lowerHeap.peek();

if (sahamTarget.harga <= lowerHeapHead.harga) {
   lowerHeap.remove(sahamTarget);
} else {
   upperHeap.remove(sahamTarget);
}
```

2. **Ubah harga saham menjadi harga yang baru**
   Cukup *set* saham baru pada attribute Class yang kita gunakan untuk memodelkan saham.

```
sahamTarget.harga = in.nextInt();
```

3. **Tambahkan saham ke Heap yang sesuai**
   Lakukan hal yang sama seperti menambahkan sahamBaru pada *query* TAMBAH.

4. **Lakukan *Balancing* Antar Heap**
   Panggil method balance() seperti di *query* TAMBAH untuk menyeimbangkan heap.

Seluruh proses di atas memiliki kompleksitas maksimal $O(4\ log\ N) = O(log\ N)$, dengan N merupakan jumlah saham di dalam Heap.