Visualisasi Metode Pengurutan

Taufik Fuadi Abidin
taufik.abidin@ndsu.nodak.edu
http://www.cs.ndsu.nodak.edu/~abidin

Pendahuluan

Mata kuliah Struktur Data merupakan salah satu mata kuliah yang diajarkan pada banyak program studi ilmu komputer. Mata kuliah ini mempelajari struktur data dan algoritma, diantaranya array, linked list, stack, queue, table hash, heap, metode pengurutan, metode pencarian, binary tree dan banyak lagi.

Salah satu cara termudah memahami materi mata kuliah ini adalah dengan cara memvisualisasikan materinya secara langsung, misalnya algoritma pengurutan. Tanpa visualisasi, algoritma pengurutan yang dipelajari harus dibayangkan oleh masing-masing orang. Hal ini tentunya tidaklah mudah, paling tidak didukung oleh beberapa alasan berikut: pertama, seringnya terjadi pertukaran data dari suatu posisi ke posisi lain selama proses pengurutan berlangsung. Kedua, sulitnya membayangkan dan mengingat posisi data yang berpindah dan data yang tidak berpindah. Ketiga, pertukaran dan perpindahan data tergantung kepada metode pengurutuan yang digunakan.

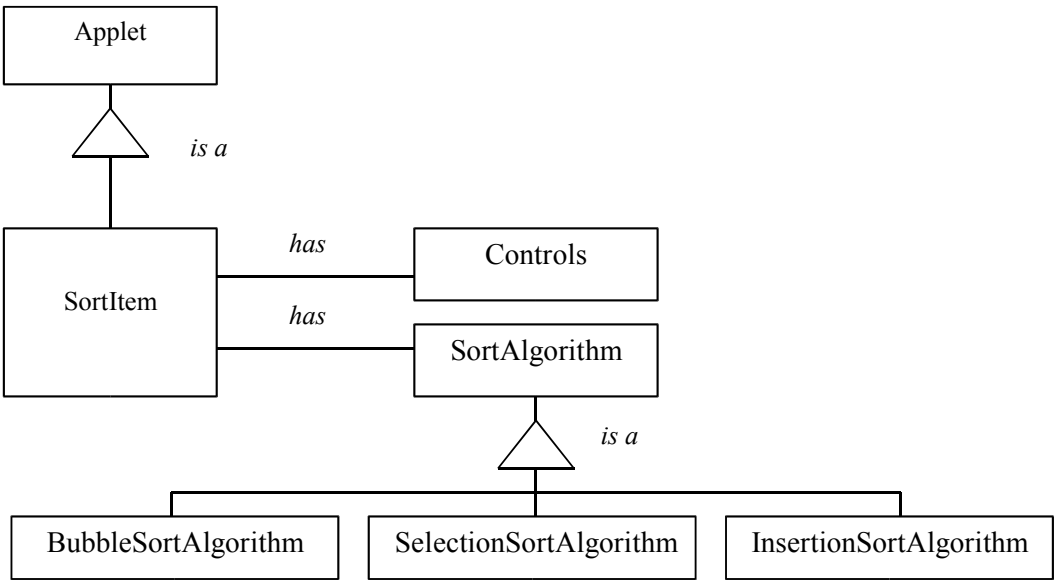
Tulisan ini membahas program visualisasi metode pengurutan dasar: *bubble sort*, *selection sort* dan *insertion sort*. Tujuannya adalah memperlihatkan perubahan posisi data dan menghitung secara tepat jumlah perbandingan dan pertukaran data selama proses pengurutan berlangsung. Implementasi program visualisasi ini menggunakan bahasa pemrograman Java. Program ini merupakan aplikasi *applet* yang terdiri dari 6 class meliputi: *SortItem*, *Control*, *SortAlgoritm*, *BubbleSortAlgorithm*, *SelectionSortAlgorithm* dan *InsertionSortAlgorithm*. Khusus class *SortItem* dan *SortAlgorithm* dikembangkan dan dimodifikasi dari program aslinya yang ditulis oleh James Gosling dari Sun Microsystems pada tahun 1997.

Metode Pengurutan

Secara umum, metode pengurutan dapat dikelompokkan dalam 2 kategori yaitu metode pengurutan sederhana (*elementary sorting methods*) dan metode pengurutan lanjut (*advanced sorting methods*). Metode pengurutan sederhana meliputi *bubble sort*, *selection sort* dan *insertion sort*. Sedangkan metode pengurutan lanjut diantaranya *shell sort*, *quick sort*, *merge sort* dan *radix sort*. Metode *bubble sort* merupakan metode pengurutan yang menggunakan konsep gelembung. Perbandingan data dilakukan dari posisi pertama atau posisi terakhir bergeser satu persatu sampai semua data dibandingkan [3], [4]. Metode *selection sort* merupakan perbaikan dari metode bubble sort dengan mengurangi jumlah perbandingan [4]. Selection sort merupakan metode pengurutan yang mencari nilai data terbesar atau terkecil dan kemudian menempatkannya pada posisi yang sebenarnya, dimulai dari data diposisi 0 hingga data diposisi N-1. Sedangkan metode *insertion sort* adalah

metode pengurutan yang biasa dipakai oleh pemain kartu dalam mengurutkan kartunya, yaitu menyisipkan kartu dengan nilai yang lebih kecil ke posisi sebelum kartu pembandingnya.

Hirarki Class



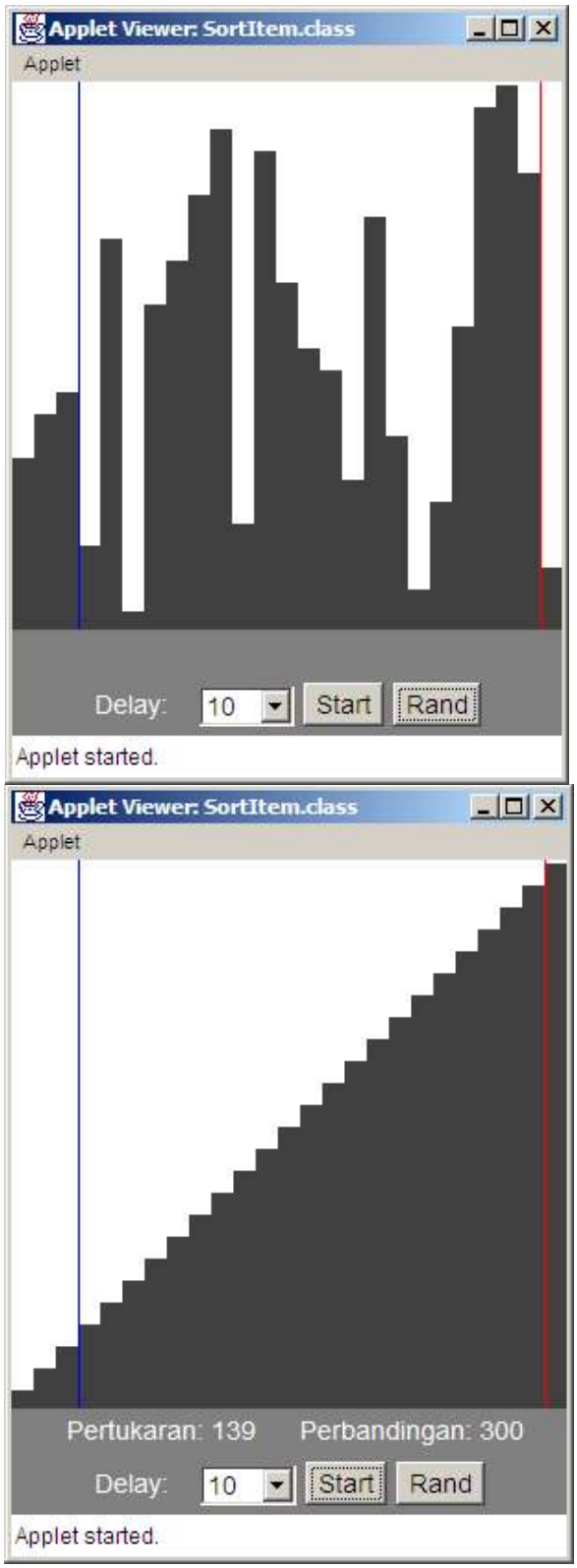
Gambar 1. Hirarki class program visualisasi.

Class *SortItem* menurunkan sifat class *applet* yang memiliki tombol-tombol pengatur visualisasi yang diimplementasi di class *Controls*. *SortItem* memiliki algoritma pengurutan (*SortAlgorithm*) yang merupakan *abstract class* yang kemudian diturunkan sifatnya oleh class *BubbleSortAlgorithm*, *SelectionSortAlgorithm*, dan *InsertionSortAlgorithm*.

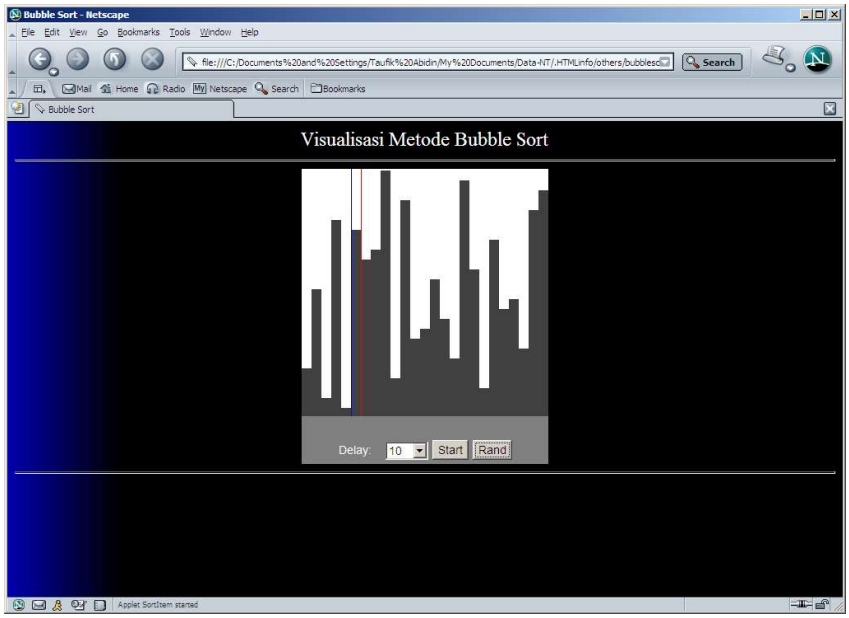
Metode Pengacakan (*Scramble*)
Dalam visualisasi ini, data direpresentasikan dalam bentuk diagram batang sebanyak 25 buah. Tinggi dan posisi diagram batang diperoleh secara acak menggunakan metode *scramble* yang ditulis dalam kelas *SortItem* sebagai berikut:

```
private void scramble(){
  int item[] = new int[25];
  double f = width / (double) item.length;
              for (int i = item.length; --i >= 0;){
                    item[i] = (int)(i * f) + 10;
                                }
                     int seed = item.length-1;
              for (int i = item.length; --i >= 0;){
                int j = (int)(seed * Math.random());
                          int temp = item[i];
                           item[i] = item[j];
                             item[j] = temp;
                                }
                              arr = item;
}
```

Nilai dari variabel *f* merupakan lebar dari diagram batang. Perulangan *for…loop* yang pertama memberikan nilai kepada array *item*, tetapi belum teracak. Sedangkan pada perulangan *for…loop* yang kedua, nilai *j* diacak dan kemudian data diposisi *i* ditukar dengan data diposisi *j*. Gambar 2 berikut memperlihatkan tampilan data setelah diacak dan setelah terurut yang dijalankan menggunakan *appletviewer command*, dan gambar 3 bila program dijalankan melalui browser.



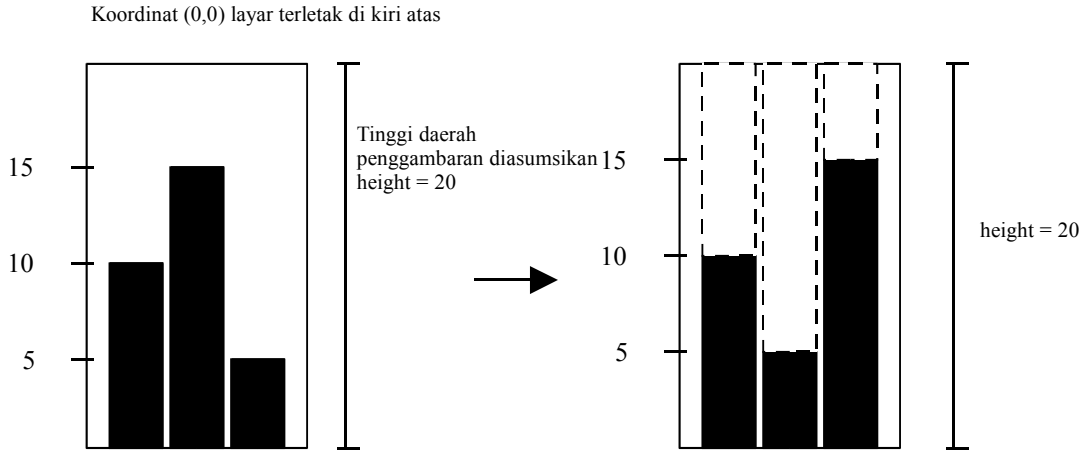
Gambar 1. Tampilan setelah diacak dan setelah terurut.



Gambar 2. Program dijalankan melalui browser.

1. Teknik Penggambaran

Teknik penggambaran merupakan kunci utama dalam program visualisasi ini. Teknik penggambaran yang tidak benar mengakibatkan visualisasi menjadi tidak halus. Sebagai contoh, bila terdapat 3 buah data yang digambarkan dalam bentuk diagram batang masing-masing bernilai 10, 15, dan 5, yang kemudian selama proses pengurutan mengalami pertukaran menjadi 10, 5, dan 15, maka cara penggambaran perubahan tersebut dapat diilustrasikan sebagai berikut:



Gambar 3. Teknik penggambaran perubahan data dalam program visualisasi.

Dari ilustrasi di atas dapat dilihat bahwa penggambaran dilakukan dalam 2 tahap. Tahap pertama adalah menggambar diagram batang dengan lebar *wbar* setinggi *height – nilai data*. Sebagai contoh, bila *nilai data = 5*, maka *20 – 5 = 15*. Warna dari diagram batang ini adalah sama dengan warna latar belakang. Penggambaran dimulai dari koordinat *y = 0* dengan *dx* merupakan koordinat *x* dari diagram batang yang berubah sesuai urutan dari diagram batang. Sintaksnya dalam Java adalah:

```
g.fillRect(dx, 0, wbar, height-arr[i]);
```

Tahap kedua adalah menggambar diagram batang warna hitam dengan lebar *wbar* setinggi *nilai data* dengan koordinat *y = height – nilai data*. Sebagai contoh, bila *nilai data = 5*, maka *y = 15*. Sintaknya dalam Java adalah:

```
g.fillRect(dx, height-arr[i], wbar, arr[i]);
```

Berikut kode lengkap dari metode *paint* untuk menggambarkan perubahan data.

```
public void paint(Graphics g){
                        int item[] = arr;
                          int nbar = 25;
                  int wbar = (int)(width / nbar);
                       int dx = width - wbar;
                    g.setColor(getBackground());
            for (int i = item.length; --i >= 0; dx -= wbar){
                 g.fillRect(dx, 0, wbar, height-arr[i]);
                                        }
                                dx = width - wbar;
                             g.setColor(Color.darkGray);
            for (int i = item.length; --i >= 0; dx -= wbar){
               g.fillRect(dx, height-arr[i], wbar, arr[i]);
                                        }
                                  if (h1 >= 0){
                               g.setColor(Color.red);
                                  dx = h1 * wbar;
                           g.drawLine(dx, 0, dx, height-1);
                                        }
                                  if (h2 >= 0){
      g.setColor(Color.blue);
      dx = h2 * wbar;
      g.drawLine(dx, 0, dx, height-1);
    }
  }
```

Hasil Pengamatan

Metode Bubble Sort

Perhatikan bahwa jumlah pertukaran dan perbandingan data tidaklah tetap, perbedaan tergantung pada keadaan awal data setelah diacak. Tabel berikut memperlihatkan tabulasi hasil visualisasi menggunakan metode *bubble sort.*

Tabel 1. Tabulasi hasil visualisasi dengan metode *bubble sort*

Pengacakan	Jumlah Total	
	Pertukaran Data	Perbandingan Data
1	155	279
2	127	272
3	137	245
4	142	297
5	151	294
6	138	285
7	178	300
8	165	272
9	156	285
10	175	300
Rata-Rata	**152**	**283**

Jumlah rata-rata pertukaran untuk 10 kali pengacakan adalah 152 kali, sedangkan rata-rata jumlah perbandingan adalah sebanyak 283 kali. Tetapi bila data pada awalnya terurut, maka untuk 25 jumlah data, jumlah pertukaran data sebanyak 0 kali dengan jumlah perbandingan sebanyak 24 kali.

Metode Selection Sort

Jumlah pertukaran data dengan metode ini tidak selalu tetap namun tergantung pada keadaan awal data. Sedangkan jumlah perbandingan data selalu tetap. Tabel 2 berikut memperlihatkan tabulasi hasil visualisasi menggunakan metode *selection sort*.

Tabel 2. Tabulasi hasil visualisasi dengan metode *selection sort*

Pengacakan	**Jumlah Total**	
	Pertukaran Data	**Perbandingan Data**
1	23	325
2	24	325
3	21	325
4	22	325
5	23	325
6	22	325
7	21	325
8	20	325
9	23	325
10	22	325
Rata-Rata	**22**	**325**

Terlihat bahwa jumlah pertukaran data turun drastis dibandingkan pengurutan menggunakan metode *bubble sort*, karena memang metode *selection sort* diciptakan untuk memperbaiki metode *bubble sort* mengurangi jumlah perbandingan [4]. Jumlah perbandingan data selalu tetap dan tidak tergantung pada keadaan awal data. Bila data awal dalam keadaan sudah terurut, maka untuk jumlah data yang sama, total pertukaran data sebanyak 0 kali dengan jumlah perbandingan tetap 325 kali.

Metode Insertion Sort

Hasil pengamatan menunjukkan bahwa dengan jumlah data dan jumlah pengacakan yang sama, metode *insertion sort* bekerja lebih cepat dibandingkan dengan metode *bubble sort* dan *selection sort* (untuk mencatat waktu secara lebih tepat, mungkin program perlu dimodifikasi sehingga waktu proses pengurutan dapat dihitung secara tepat). Jumlah pertukaran data dengan metode ini juga tergantung pada keadaan awal data yaitu rata-rata sebanyak 136 kali (dari 10 kali pengamatan), dan jumlah perbandingan data selalu 300 kali. Berikut tabulasi hasil visualisasi menggunakan metode *insertion sort*.

Tabel 3. Tabulasi hasil visualisasi dengan metode *insertion sort*

Pengacakan	**Jumlah Total**	
	Pertukaran Data	**Perbandingan Data**
1	149	300
2	138	300
3	168	300
4	157	300
5	102	300
6	106	300
7	122	300
8	123	300

9	145	300
10	148	300
Rata-Rata	**136**	**300**

Penutup

Hasil pengamatan menunjukkan bahwa dengan program visualisasi, proses pengurutan dapat diilustrasikan secara cepat, mudah dan berulang-ulang. Program visualisasi juga memperlihatkan secara jelas bagaimana proses pertukaran data terjadi, total perbandingan dan jumlah pertukaran data yang diperlukan. Selamat mencoba dan jangan lupa, kembangkan untuk hal-hal lain yang menarik.

Referensi

1. Flanigan, David, *Java in a Nutshell: A Desktop Quick Reference*, 2nd edition, O'Reilly & Associates Inc. California, 1997.
2. Savitch, Walter, *Java: An Introduction to Computer Science & Programming*, Prentice Hall Inc, New Jersey, 1999.
3. Sedgewich, Robert, *Algorithms in C*, 3rd edition, Addison-Wesley Publishing, USA, 1998.
4. Waite, Mitchell, *Data Structures and Algorithms in Java*, Waite Group Press, California, 1998.