LAPORAN TUGAS

Desain dan Analisis Algoritma

"Tugas Sorting"

NAMA :

RIDHO NUR ROHMAN WIJAYA

06111840000065

DEPARTEMEN MATEMATIKA

FAKULTAS SAINS DAN ANALITIKA DATA

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

SURABAYA

2021

A. PERMASALAHAN

Pada tugas ini akan ditampilkan penerapan beberapa algoritma yang ada untuk menjawab permasalahan yang diberikan.

1. Deskripsi Masalah

Masalah yang diberikan adalah

a) Nomor 1

Buat algoritma Merge Sort dan lakukan analisis beserta contoh dengan data.

b) Nomor 2

Buat algoritma Heap Sort dan lakukan analisis beserta contoh dengan data.

c) Nomor 3

Implementasikan algoritma Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Merge Sort, dan Heap Sort. Bandingkan dalam grafik untuk data dengan ukuran

$$n = 1000, 10000, 100000, 1000000, 10000000$$

2. Jawaban Masalah

Jawaban dari permasalahan yang ada adalah

a) Nomor 1

Input : Array tak terurut A dengan panjang array n.

Output : Array terurut A dengan elemen terurut naik.

Algoritmanya adalah

```
1.  MergeSort(A,m,n){
2.  if (m!=n)
3.       mid = (m+n)/2
4.       MergeSort(A,m,mid)
5.       MergeSort(A,mid+1,n)
6.       Merge(A,m,mid,mid+1,n)
7.  }
```

```
8.  Merge(A,aLeft,bRight,bLeft,bRight){
9.  size = bRight - aLeft + 1
10. temp = new Integer[size]
11. tIndex = 0
12. aIndex = aleft
13. bIndex = bLeft
14. while aIndex<=aRight && bIndex<=bRight
15.       if A[tIndex]<=A[aIndex]
16.             temp[tIndex] = A[aIndex]
17.             aIndex++
18.       else
19.             temp[tIndex] = A[bIndex]
20.             bIndex++
21.       tIndex++
22. while aIndex<=aRight
23.       temp[tIndex] = A[aIndex]
24.       aIndex++
25.       tIndex++
26. while bIndex<=bRight
27.       temp[tIndex] = A[bIndex]
28.       bIndex++
29.       tIndex++
30. for (i=0; i<size; i++)
31.       A[aLeft+i] = temp[i]
32. }
```

Untuk menghitung banyaknya kerja, pekerjaan utama yang ditinjau adalah operator perbandingan. Untuk array dengan panjang *n*, proses *merge* paling buruk membutuhkan *n* kali (*Step 6*) perbandingan. Karena proses *sorting* dilakukan dengan *sorting* bagian kiri (*Step 4*) dan kanan (*Step 5*), atau disebut proses *devide* maka banyaknya kerja adalah

$$
\begin{aligned}
W(n) &= n + W\left(\frac{n}{2}\right) + W\left(\frac{n}{2}\right) \\
&= n + 2W\left(\frac{n}{2}\right) \\
&= n + 2\left[\frac{n}{2} + 2W\left(\frac{n}{4}\right)\right] \\
&= 2n + 4W\left(\frac{n}{4}\right) \\
&= 2n + 4\left[\frac{n}{4} + 2W\left(\frac{n}{8}\right)\right] \\
&= 3n + 8W\left(\frac{n}{8}\right) \\
&\vdots
\end{aligned}
$$

diperoleh

$$W(n) = kn + 2^k W\left(\frac{n}{2^k}\right)$$

Tinjau saat $n = 1$ maka $W(1) = 1$, sehigga untuk

$$\frac{n}{2^k} = 1 \Longrightarrow k = \log_2 n$$

Oleh karena itu didapat

$$
\begin{aligned}
W(n) &= n\log_2 n + 2^{\log_2 n} W(1) \\
&= n\log_2 n + n \\
&= n(\log_2 n + 1)
\end{aligned}
$$

Jadi banyaknya kerja yang dilakukan pada algoritma *Merge Sort* adalah

$$W(n) = O(n \log n)$$

Contoh penggunaan Merge Sort pada array $A = [10,\ 0,\ 3,\ 16,\ 18,\ 12,\ 15]$

- Visualisasi *Merge Sort*

Pertama dilakukan proses *devide* untuk memecah array *A* sampai terbentuk 1 array seperti langkah berikut

$$[10, 0, 3, 16, 18, 12, 15]$$

$$[10, 0, 3, 16] \quad [18, 12, 15]$$

$$[10, 0] \quad [3, 16] \quad [18, 12] \quad [15]$$

$$[10] \quad [0] \quad [3] \quad [16] \quad [18] \quad [12] \quad [15]$$

Setelah itu dilagukan proses *merge* dengan terurut naik

$$[0,10] \quad [3,16] \quad [12,18] \quad [15]$$

$$[0,3,10,16] \quad [12,15,18]$$

$$[0,3,10,12,15,16,18]$$

Jadi terbentuk array terurut $A = [0,\ 3,\ 10,\ 12,\ 15,\ 16,\ 18]$.

- Visualisasi *Merge Sort* dalam bentuk array

Visualisasi dalam bentuk array akan sedikit berbeda dengan visualisasi dasarnya, karena proses program dilakukan dari kiri ke kanan. Proses sorting tersebut adalah

$$A = [10,\ 0,\ 3,\ 16,\ 18,\ 12,\ 15]$$
$$A = [0,\ 10,\ 3,\ 16,\ 18,\ 12,\ 15]$$
$$A = [0,\ 10,\ 3,\ 16,\ 18,\ 12,\ 15]$$
$$A = [0,\ 3,\ 10,\ 16,\ 18,\ 12,\ 15]$$
$$A = [0,\ 3,\ 10,\ 16,\ 12,\ 18,\ 15]$$
$$A = [0,\ 3,\ 10,\ 16,\ 12,\ 15,\ 18]$$
$$A = [0,\ 3,\ 10,\ 12,\ 16,\ 16,\ 18]$$

Jadi terbentuk array terurut $A = [0,\ 3,\ 10,\ 12,\ 15,\ 16,\ 18]$.

b) Nomor 2

Input : Array tak terurut A dengan panjang array n.

Output : Array terurut A dengan elemen terurut naik.

Algoritmanya adalah

```
1.  HeapSort(A,n){
2.  for (i=n/2-1; i>=0; i--)
3.       Heapify(A,n,i)
4.  for (i=n-1; i>=0; i--)
5.       Heapify(A,i,0)
6.  }
7.
8.  Heapify(A,n,i){
9.  largest = i
10. left = 2*i + 1
```

```
11.  right = 2*i + 2
12.  if left<n && A[left]>A[largest]
13.       largest = left
14.  if right<n && A[right]>A[largest]
15.       largest = right
16.  if largest != i
17.       temp = A[i]
18.       A[i] = A[largest]
19.       A[largest] = temp
20.       Heapify(A,n,largest)
21.  }
```

Untuk menghitung banyaknya kerja, pekerjaan utama yang ditinjau adalah operator perbandingan. Pada proses *Heapify* (*Step 8*), tinjau untuk *tree* dengan n node, maka maksimal node pada *subtree* adalah $\frac{2n}{3}$. Sehingga banyak kerja pada proses *Heapify* adalah

$$\begin{aligned} H(n) &= 1 + H\left(\frac{2n}{3}\right) \\ &= 2 + H\left(\frac{2^2 n}{3^2}\right) \\ &= 3 + H\left(\frac{2^3 n}{3^3}\right) \\ &\vdots \end{aligned}$$

diperoleh

$$H(n) = k + H\left(\left(\frac{2}{3}\right)^k n\right)$$

Tinjau saat $n = 1$ maka $H(1) = 1$, sehigga untuk

$$\left(\frac{2}{3}\right)^k n = 1 \Longrightarrow k = \log_{2/3} n$$

Oleh karena itu didapat

$$\begin{aligned} H(n) &= \log_{2/3} n + H(1) \\ &= \log_{2/3} n + 1 \\ &= O(\log n) \end{aligned}$$

Pada proses pembentukan *tree* (*Step 2*), ketika kedalaman *tree* adalah h maka banyak nodenya adalah $\left\lceil \frac{n}{2^{h+1}} \right\rceil$. Setiap node akan melakukan proses *Heapify*, sehigga banyak kerja pada saat pembentukan *tree* adalah

$$T(n) = \sum_{h=0}^{\log n} \left\lceil \frac{n}{2^{h+1}} \right\rceil H(n) = \sum_{h=0}^{\log n} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(\log n) = O(n)$$

Pada proses pembentukan *tree* dengan angka terbesar (*Step 4*) ada di bagian atas memiliki banyak kerja

$$P(n) = (n-1)H(n) = (n-1)O(\log n) = O(n \log n)$$

Jadi banyak kerja pada algoritma *Heap Sort* adalah

$$W(n) \ = T(n) + P(n) = O(n) + O(n \log n) = O(n \log n)$$

Contoh penggunaan Heap Sort pada array $A = [10,\ 0,\ 3,\ 16,\ 18,\ 12,\ 15]$

- Visualisasi *Heap Sort* dalam bentuk *tree*

  Pertama dilakukan proses pembentukan *tree* dari array A seperti berikut

  [10]

  [0] [3]

  [16] [18] [12] [15]

  Setelah itu dilagukan proses *heapify* dengan tahapan-tahapan sebagai berikut

  [10]

  [0] [15]

  [16] [18] [12] [3]

  [10]

  [18] [15]

  [16] [0] [12] [3]

  [18]

  [10] [15]

  [16] [0] [12] [3]

[18]

[16] [15]

[10] [0] [12] [3]

[3]

[16] [15]

[10] [0] [12] [18]

[16]

[3] [15]

[10] [0] [12] [18]

[16]

[10] [15]

[3] [0] [12] [18]

[12]

[10] [15]

[3] [0] [16] [18]

[15]

[10] [12]

[3] [0] [16] [18]

[0]

[10] [12]

[3] [15] [16] [18]

[12]

[10] [0]

[3] [15] [16] [18]

[3]

[10] [0]

[12] [15] [16] [18]

[10]

[3] [0]

[12] [15] [16] [18]

[0]

[3] [10]

[12] [15] [16] [18]

[3]

[0] [10]

[12] [15] [16] [18]

[0]

[3] [10]

[12] [15] [16] [18]

Bentuk kembali dalam array sehingga diperoleh array terurutnya adalah $A = [0,\ 3,\ 10,\ 12,\ 15,\ 16,\ 18]$.

- Visualisasi *Heap Sort* dalam bentuk array

  Proses sorting tersebut adalah

$$A = [10,\ 0,\ 3,\ 16,\ 18,\ 12,\ 15]$$
$$A = [10,\ 0,\ 15,\ 16,\ 18,\ 12,\ 3]$$
$$A = [10,\ 18,\ 15,\ 16,\ 0,\ 12,\ 3]$$
$$A = [18,\ 10,\ 15,\ 16,\ 0,\ 12,\ 3]$$
$$A = [18,\ 16,\ 15,\ 10,\ 0,\ 12,\ 3]$$
$$A = [3,\ 16,\ 15,\ 10,\ 0,\ 12,\ 18]$$
$$A = [16,\ 3,\ 15,\ 10,\ 0,\ 12,\ 18]$$
$$A = [16,\ 10,\ 15,\ 3,\ 0,\ 12,\ 18]$$

$$A = [12, \quad 10, \quad 15, \quad 3, \quad 0, \quad 16, \quad 18]$$
$$A = [15, \quad 10, \quad 12, \quad 3, \quad 0, \quad 16, \quad 18]$$
$$A = [0, \quad 10, \quad 12, \quad 3, \quad 15, \quad 16, \quad 18]$$
$$A = [12, \quad 10, \quad 0, \quad 3, \quad 15, \quad 16, \quad 18]$$
$$A = [3, \quad 10, \quad 0, \quad 12, \quad 15, \quad 16, \quad 18]$$
$$A = [10, \quad 3, \quad 0, \quad 12, \quad 15, \quad 16, \quad 18]$$
$$A = [0, \quad 3, \quad 10, \quad 12, \quad 15, \quad 16, \quad 18]$$
$$A = [3, \quad 0, \quad 10, \quad 12, \quad 15, \quad 16, \quad 18]$$
$$A = [0, \quad 3, \quad 10, \quad 12, \quad 15, \quad 16, \quad 18]$$

Jadi terbentuk array terurut $A = [0,\ 3,\ 10,\ 12,\ 15,\ 16,\ 18]$.

c) Nomor 3

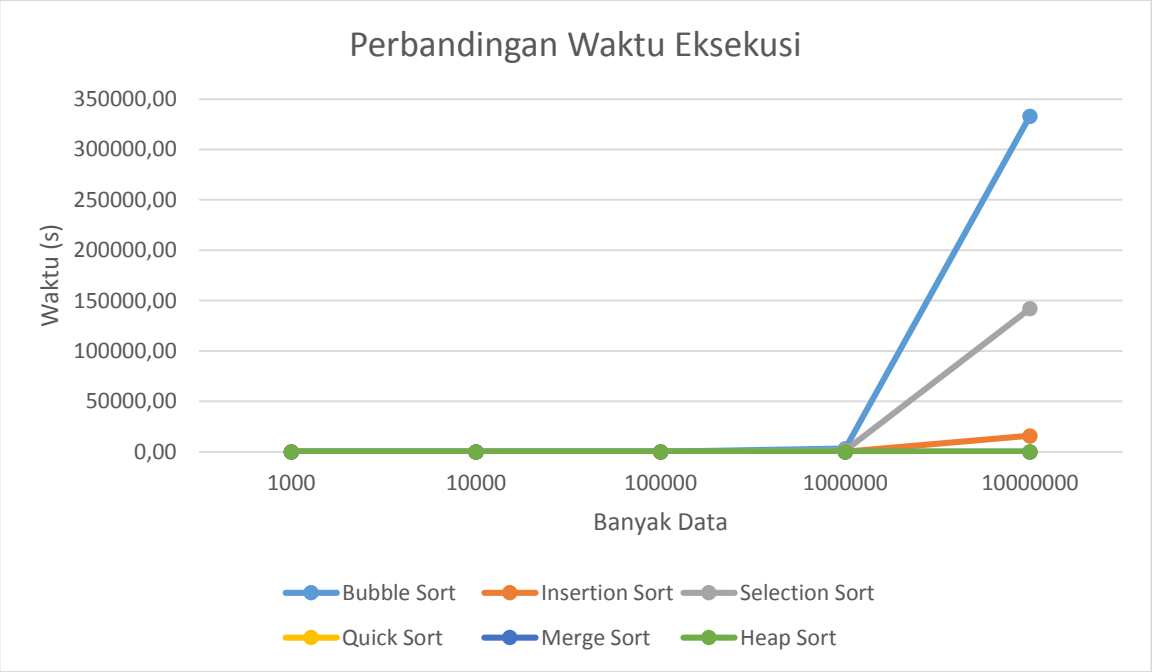
Setelah proses running program didapat data sebagai berikut

Tabel 3.1 Perbandingan waktu eksekusi (nanoseconds)

	Banyak Data				
Metode	1000	10000	100000	1000000	10000000
Bubble Sort	25257871	297560869	29662705864	3131597936944	333064706363578
Insertion Sort	4876745	28148934	1498020120	158404979554	15929062319208
Selection Sort	7386736	140331917	13655961686	1402318804093	142058706453121
Quick Sort	1477436	11863221	26327908	204841583	1857634656
Merge Sort	1364989	12833750	32045332	315609600	2800221236
Heap Sort	1154373	28734822	26041881	345124861	4631513259

Keterangan: waktu eksekusi Bubble Sort, Insertion Sort, dan Selection Sort adalah hasil dari prediksi, sebab jika diperkirakan waktu yang dibutuhkan Bubble Sort adalah 85 jam, Insertion Sort adalah 4 jam, Selection Sort adalah 40 jam.

Serta diperoleh grafik sebagai berikut



Kesimpulan: Metode paling lama waktu eksekusinya adalah Bubble Sort, sedangkan metode paling cepat adalah Quick Sort. Secara keseluruhan, source code program dari setiap algoritma akan ada pada bab Source Code dan hasil training semua algoritma akan ditampilkan pada bab Running Program.

B. SOURCE CODE

Program penyelesaian masalah-masalah yang ada dapat di implementasikan pada source code berikut ini:

a) Bubble Sort

BubbleSortCode.java

```
1.  /*
2.   * To change this license header, choose License Headers in
     Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  package TugasSorting;
7.
8.  /**
9.   *
10.  * @author OWNER
11.  */
12. public class BubbleSortCode {
13.
14.     public static void BubbleSort(int[] A, int n) {
15.         for (int i = 0; i < n - 1; i++) {
16.             for (int j = 0; j < n - 1 - i; j++) {
17.                 if (A[j] > A[j + 1]) {
18.                     int temp = A[j];
19.                     A[j] = A[j + 1];
20.                     A[j + 1] = temp;
21.                 }
22.             }
23.         }
24.     }
25. }
```

b) Insertion Sort

InsertionSortCode.java

```
1.  /*
2.   * To change this license header, choose License Headers in
     Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  package TugasSorting;
7.
8.  /**
9.   *
10.  * @author OWNER
11.  */
12. public class InsertionSortCode {
13.
14.     public static void InsertionSort(int[] A, int n) {
```

```
15.         int key, j;
16.         for (int i = 1; i < n; i++) {
17.             key = A[i];
18.             j = i - 1;
19.             while (j >= 0 && A[j] > key) {
20.                 A[j + 1] = A[j];
21.                 j--;
22.             }
23.             A[j + 1] = key;
24.         }
25.     }
26. }
```

c) Selection Sort

SelectionSortCode.java

```
1.  /*
2.   * To change this license header, choose License Headers in
    Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  package TugasSorting;
7.
8.  /**
9.   *
10.  * @author OWNER
11.  */
12. public class SelectionSortCode {
13.
14.     public static void SelectionSort(int[] A, int n) {
15.         for (int i = 0; i < n - 1; i++) {
16.             int index = i;
17.             for (int j = i + 1; j < n; j++) {
18.                 if (A[j] < A[index]) {
19.                     index = j;
20.                 }
21.             }
22.             int smallerNumber = A[index];
23.             A[index] = A[i];
24.             A[i] = smallerNumber;
25.         }
26.     }
27. }
```

d) Quick Sort

QuickSortCode.java

```
1.  /*
2.   * To change this license header, choose License Headers in
    Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  package TugasSorting;
7.
8.  /**
9.   *
10.  * @author OWNER
11.  */
12. public class QuickSortCode {
13.
14.     public static void QuickSort(int[] A, int left, int right) {
15.         if (left < right) {
16.             int iPivot = (left + right) / 2;
17.             int pivot = A[iPivot];
18.             int pLeft = left;
19.             int pRight = right;
20.             while (pLeft <= pRight) {
21.                 while (A[pLeft] < pivot) {
22.                     pLeft++;
23.                 }
24.                 while (A[pRight] > pivot) {
25.                     pRight--;
26.                 }
27.                 if (pLeft <= pRight) {
28.                     int temp = A[pLeft];
29.                     A[pLeft] = A[pRight];
30.                     A[pRight] = temp;
31.                     pLeft++;
32.                     pRight--;
33.                 }
34.             }
35.             QuickSort(A, left, pRight);
36.             QuickSort(A, pLeft, right);
37.         }
38.     }
39. }
```

e) Merge Sort

MergeSortCode.java

```
1.  /*
2.   * To change this license header, choose License Headers in
    Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  package TugasSorting;
```

```
7.
8.  /**
9.   *
10.  * @author OWNER
11.  */
12. public class MergeSortCode {
13.
14.     public static void MergeSort(int[] A, int m, int n) {
15.         int mid;
16.         if (m != n) {
17.             mid = (m + n) / 2;
18.             MergeSort(A, m, mid);
19.             MergeSort(A, mid + 1, n);
20.             Merge(A, m, mid, mid + 1, n);
21.         }
22.     }
23.
24.     private static void Merge(int[] A, int sub1_left, int
sub1_right, int sub2_left, int sub2_right) {
25.
26.         int[] temp;
27.         int sub_size, sub_index, sub1_left_index,
sub2_left_index;
28.
29.         sub_size = sub2_right - sub1_left + 1;
30.         temp = new int[sub_size];
31.         sub_index = 0;
32.         sub1_left_index = sub1_left;
33.         sub2_left_index = sub2_left;
34.
35.         while (sub1_left_index <= sub1_right && sub2_left_index
<= sub2_right) {
36.             if (A[sub1_left_index] < A[sub2_left_index]) {
37.                 temp[sub_index] = A[sub1_left_index];
38.                 sub1_left_index++;
39.             } else {
40.                 temp[sub_index] = A[sub2_left_index];
41.                 sub2_left_index++;
42.             }
43.             sub_index++;
44.         }
45.
46.         while (sub1_left_index <= sub1_right) {
47.             temp[sub_index] = A[sub1_left_index];
48.             sub1_left_index++;
49.             sub_index++;
50.         }
51.
52.         while (sub2_left_index <= sub2_right) {
53.             temp[sub_index] = A[sub2_left_index];
54.             sub2_left_index++;
55.             sub_index++;
56.         }
57.
```

```
58.             System.arraycopy(temp, 0, A, sub1_left, sub_size);
59.         }
60.     }
```

f) Heap Sort

HeapSortCode.java

```
1.  /*
2.   * To change this license header, choose License Headers in
     Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  package TugasSorting;
7.
8.  /**
9.   *
10.  * @author OWNER
11.  */
12. public class HeapSortCode {
13.
14.     public static void HeapSort(int A[], int n) {
15.         for (int i = n / 2 - 1; i >= 0; i--) {
16.             Heapify(A, n, i);
17.         }
18.
19.         for (int i = n - 1; i >= 0; i--) {
20.             int temp = A[0];
21.             A[0] = A[i];
22.             A[i] = temp;
23.
24.             Heapify(A, i, 0);
25.         }
26.     }
27.
28.     private static void Heapify(int A[], int n, int i) {
29.         int largest = i;
30.         int left = 2 * i + 1;
31.         int right = 2 * i + 2;
32.
33.         if (left < n && A[left] > A[largest]) {
34.             largest = left;
35.         }
36.
37.         if (right < n && A[right] > A[largest]) {
38.             largest = right;
39.         }
40.
41.         if (largest != i) {
42.             int temp = A[i];
43.             A[i] = A[largest];
44.             A[largest] = temp;
45.
```

```
46.                 Heapify(A, n, largest);
47.             }
48.         }
49.     }
```

g) Main program untuk eksekusi Merge Sort dan Heap Sort

mainExecution.java

```
1.  /*
2.   * To change this license header, choose License Headers in
     Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  package TugasSorting;
7.
8.  import java.util.Arrays;
9.  import java.util.Random;
10.
11. /**
12.  *
13.  * @author OWNER
14.  */
15. public class mainExecution {
16.
17.     static Random inran = new Random();
18.
19.     public static void main(String[] args) {
20.         int n;
21.         int[] arr, A, B;
22.
23.         n = 7;
24.         arr = new int[n];
25.         arr = GenerateArray(arr, n);
26.
27.         A = arr.clone();
28.         B = arr.clone();
29.
30.         System.out.println("Array A");
31.         System.out.println(Arrays.toString(A));
32.         System.out.println("Array A setelah diurutkan dengan
            metode Merge Sort");
33.         MergeSortCode.MergeSort(A, 0, n - 1);
34.         System.out.println(Arrays.toString(A));
35.
36.         System.out.println();
37.
38.         System.out.println("Array B");
39.         System.out.println(Arrays.toString(B));
40.         System.out.println("Array B setelah diurutkan dengan
            metode Heap Sort");
41.         HeapSortCode.HeapSort(B, n);
42.         System.out.println(Arrays.toString(B));
43.     }
```

```
44.
45.     public static int[] GenerateArray(int[] A, int n) {
46.         for (int i = 0; i < n; i++) {
47.             A[i] = inran.nextInt(3 * n);
48.         }
49.
50.         return A;
51.     }
52. }
```

h) Main program untuk cek perbandingan waktu eksekusi

mainComparison.java

```
1.  /*
2.   * To change this license header, choose License Headers in
Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  package TugasSorting;
7.
8.  import java.util.Random;
9.  import java.util.Scanner;
10.
11. /**
12.  *
13.  * @author OWNER
14.  */
15. public class mainComparison {
16.
17.     static Scanner input = new Scanner(System.in);
18.     static Random inran = new Random();
19.
20.     public static void main(String[] args) {
21.         int n;
22.         int[] arr, A, B, C, D, E, F;
23.
24.         System.out.print("Masukkan banyak data: ");
25.         n = input.nextInt();
26.         arr = new int[n];
27.         arr = GenerateArray(arr, n);
28.
29.         A = arr.clone();
30.         B = arr.clone();
31.         C = arr.clone();
32.         D = arr.clone();
33.         E = arr.clone();
34.         F = arr.clone();
35.
36.         /*
37.          Case 1 : Bubble Sort
38.          Case 2 : Insertion Sort
39.          Case 3 : Selection Sort
```

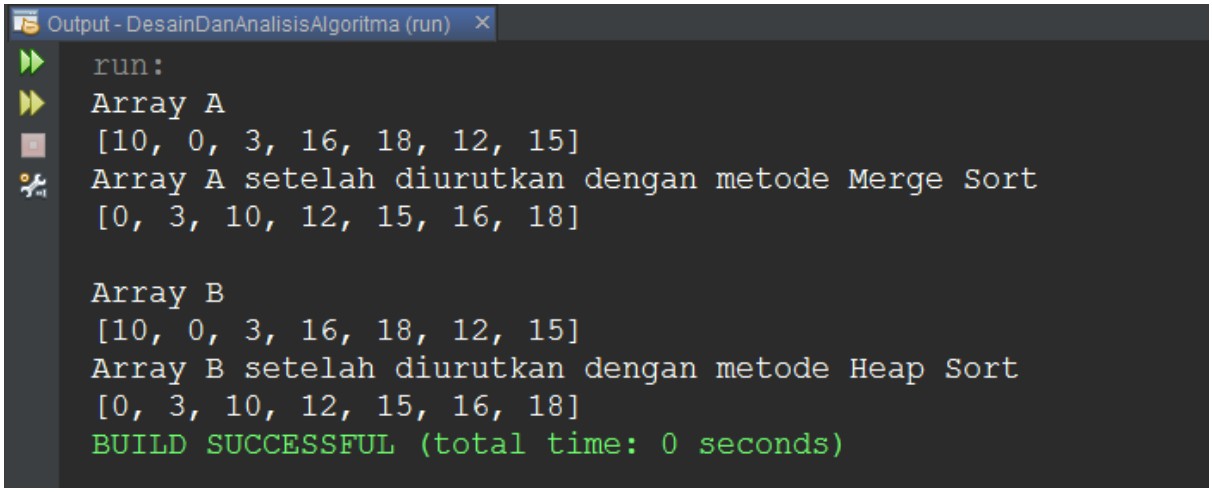
```
40.          Case 4 : Quick Sort
41.          Case 5 : Merge Sort
42.          Case 6 : Heap Sort
43.          */
44.
45.         System.out.println("-----------------------------------
------");
46.         System.out.println("Waktu eksekusi untuk " + n + "
data");
47.         System.out.println("-----------------------------------
------");
48.         System.out.println("Metode\t\t\tWaktu(ns)");
49.
50.         System.out.println("Bubble Sort\t\t" + Sort(A, n, 1));
51.         System.out.println("Insertion Sort\t\t" + Sort(B, n,
2));
52.         System.out.println("Selection Sort\t\t" + Sort(C, n,
3));
53.         System.out.println("Quick Sort\t\t" + Sort(D, n, 4));
54.         System.out.println("Merge Sort\t\t" + Sort(E, n, 5));
55.         System.out.println("Heap Sort\t\t" + Sort(F, n, 6));
56.
57.         System.out.println("-----------------------------------
------");
58.     }
59.
60.     public static int[] GenerateArray(int[] A, int n) {
61.         for (int i = 0; i < n; i++) {
62.             A[i] = inran.nextInt(3 * n);
63.         }
64.
65.         return A;
66.     }
67.
68.     public static long Sort(int[] A, int n, int choice) {
69.         long startTime = 0;
70.         long endTime = 0;
71.
72.         switch (choice) {
73.             case 1:
74.                 startTime = System.nanoTime();
75.                 BubbleSortCode.BubbleSort(A, n);
76.                 endTime = System.nanoTime();
77.                 break;
78.             case 2:
79.                 startTime = System.nanoTime();
80.                 InsertionSortCode.InsertionSort(A, n);
81.                 endTime = System.nanoTime();
82.                 break;
83.             case 3:
84.                 startTime = System.nanoTime();
85.                 SelectionSortCode.SelectionSort(A, n);
86.                 endTime = System.nanoTime();
87.                 break;
```

```
88.                 case 4:
89.                     startTime = System.nanoTime();
90.                     QuickSortCode.QuickSort(A, 0, n - 1);
91.                     endTime = System.nanoTime();
92.                     break;
93.                 case 5:
94.                     startTime = System.nanoTime();
95.                     MergeSortCode.MergeSort(A, 0, n - 1);
96.                     endTime = System.nanoTime();
97.                     break;
98.                 case 6:
99.                     startTime = System.nanoTime();
100.                    HeapSortCode.HeapSort(A, n);
101.                    endTime = System.nanoTime();
102.                    break;
103.                default:
104.                    System.out.println("Tidak ada pilihan metode
sorting");
105.            }
106.
107.            return endTime - startTime;
108.    }
109. }
```

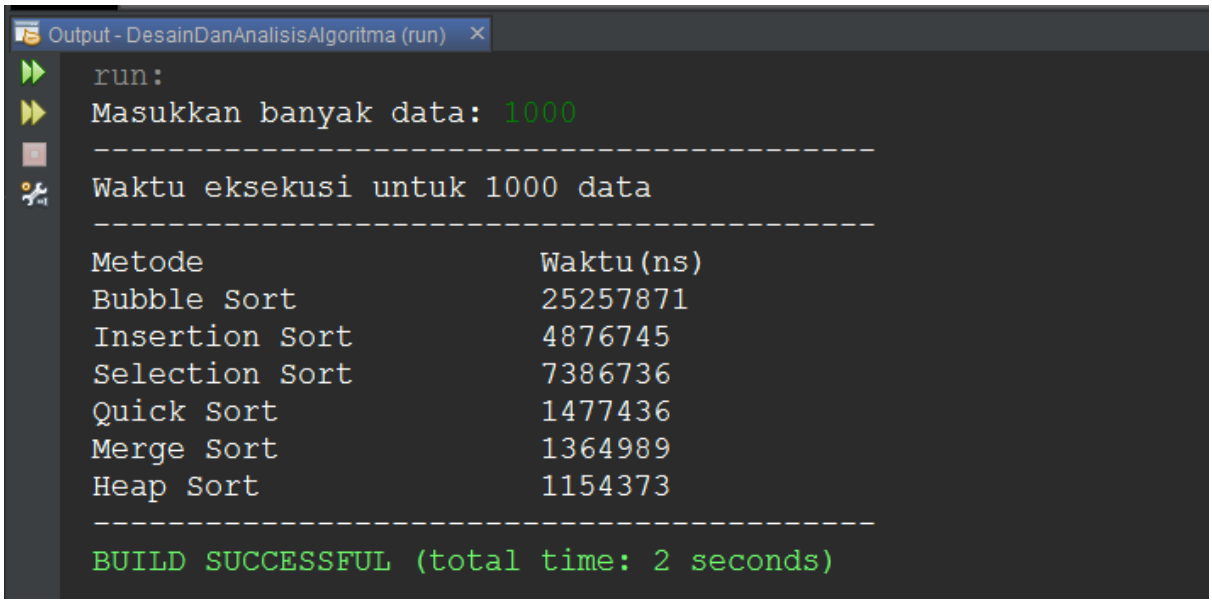
C. RUNNING PROGRAM

Beberapa hasil outputan dari program tersebut adalah:

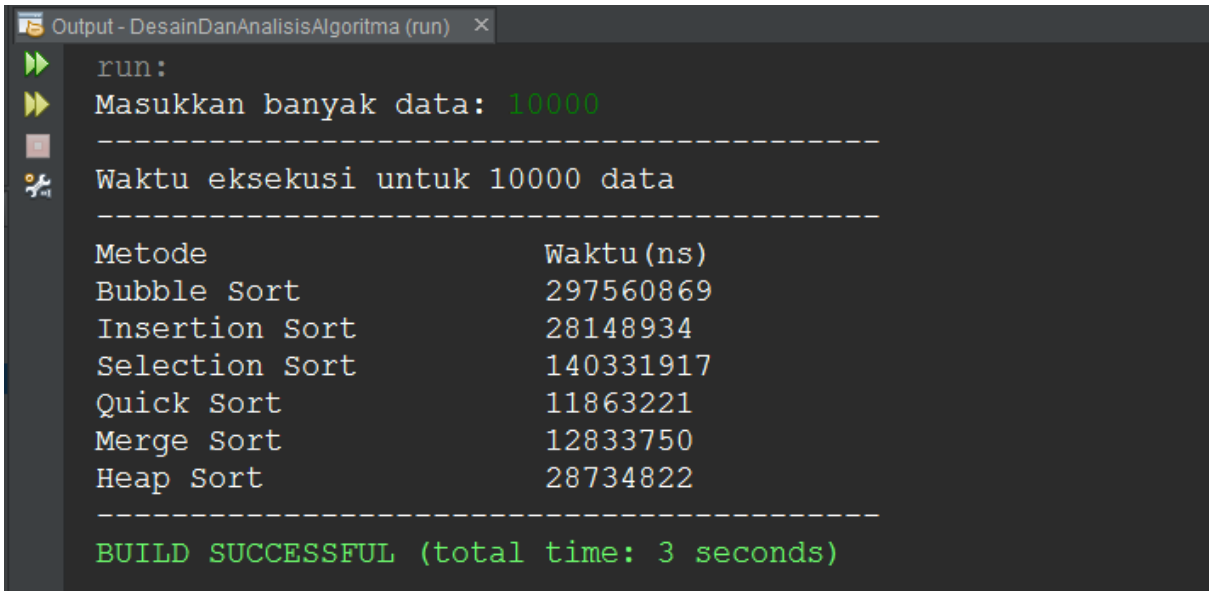
a) Hasil keluaran sorting dengan metode Merge Sort dan Heap Sort



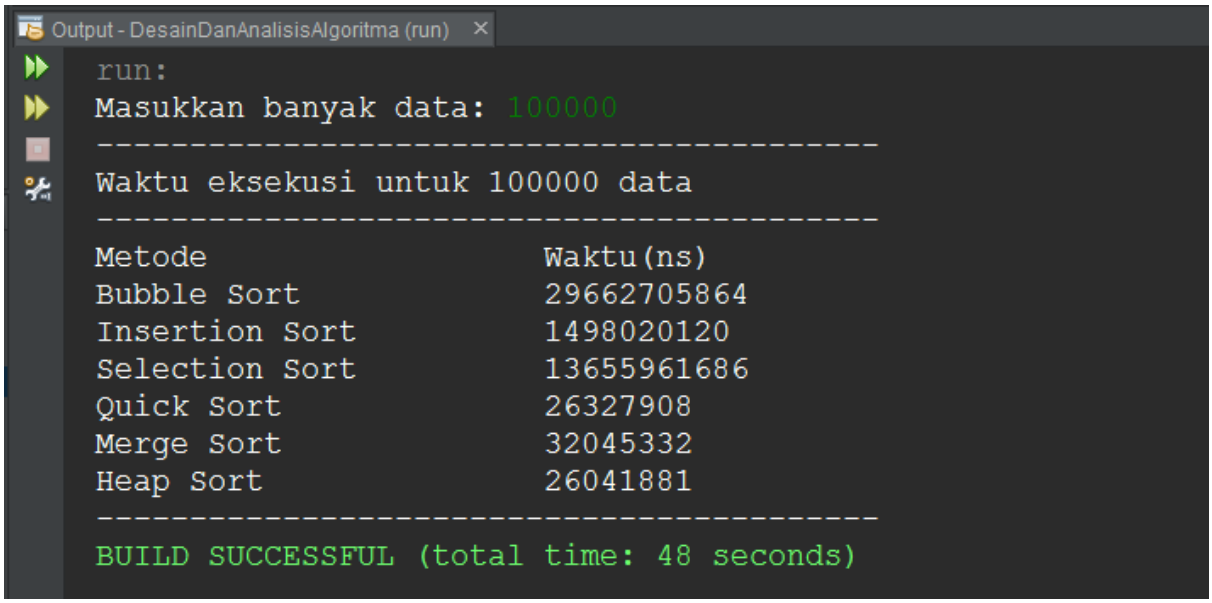
b) Perbandingan waktu eksekusi untuk 1000 data



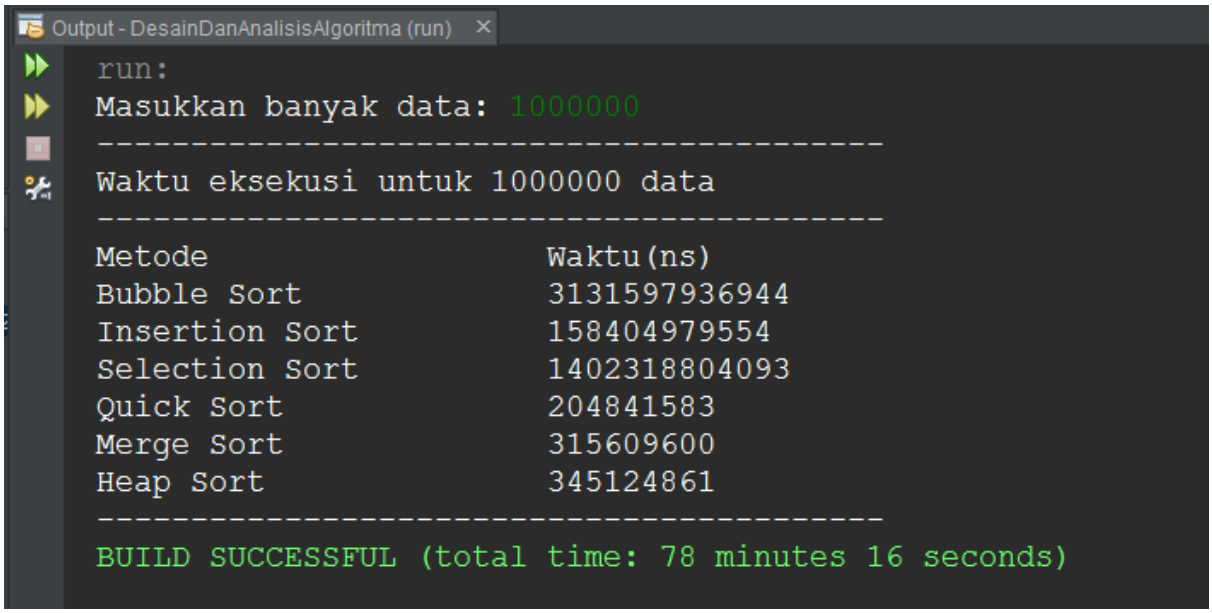
c) Perbandingan waktu eksekusi untuk 10000 data



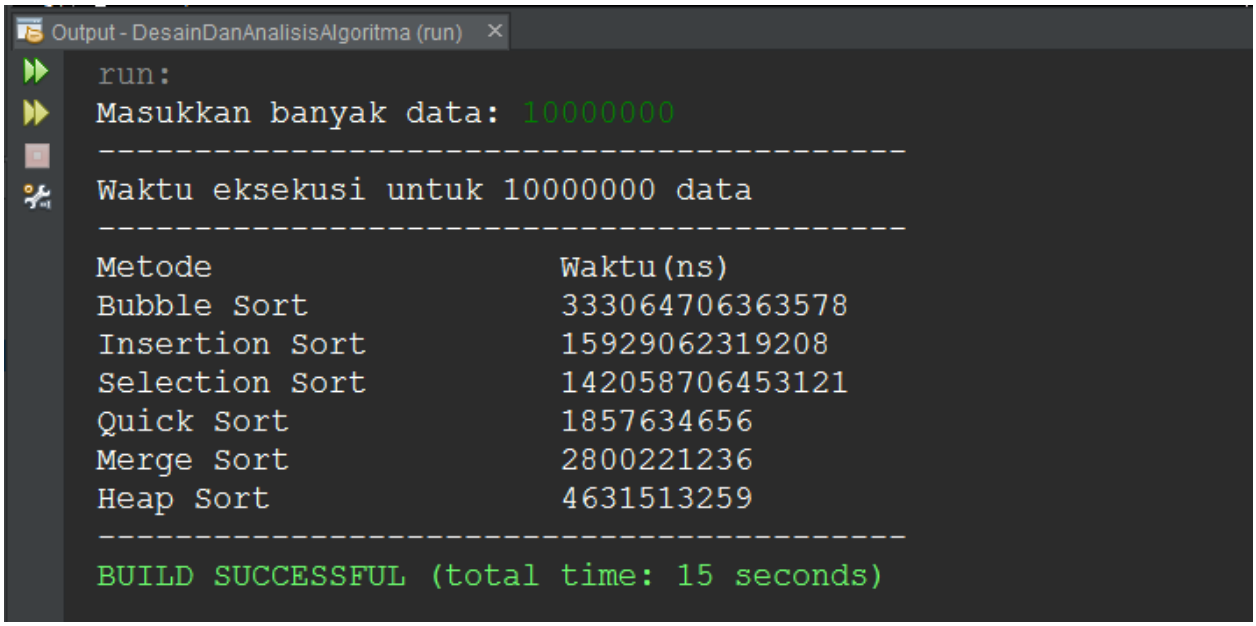
d) Perbandingan waktu eksekusi untuk 100000 data



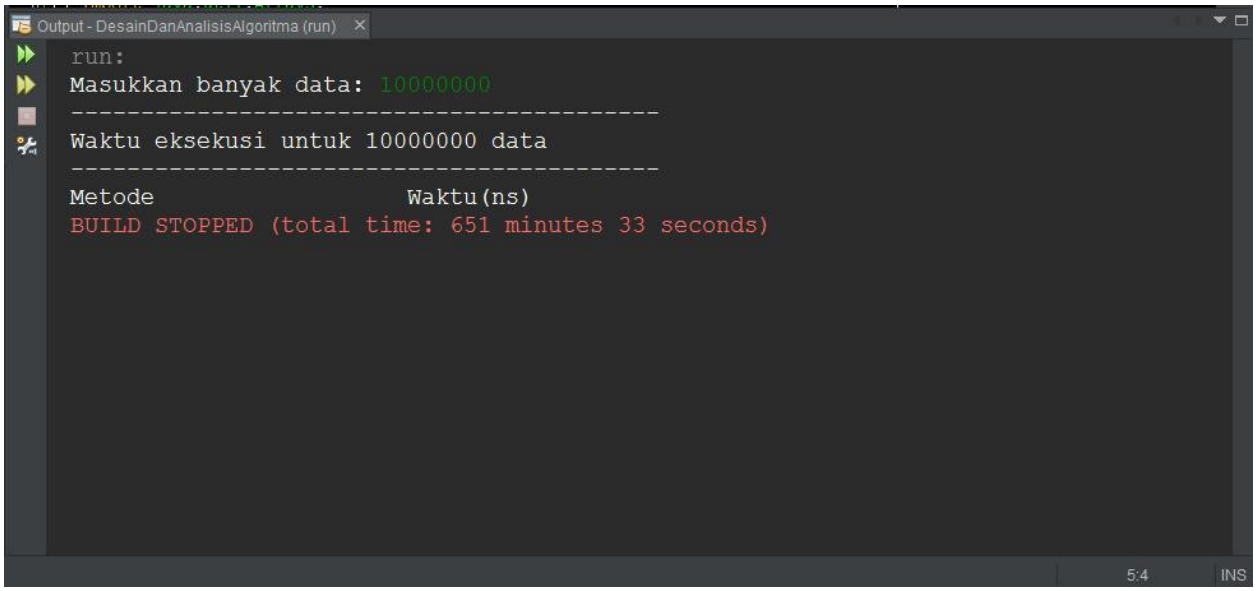
e) Perbandingan waktu eksekusi untuk 1000000 data



f) Perbandingan waktu eksekusi untuk 10000000 data



g) Bukti waktu metode pengurutan dengan kompleksitas $O(n^2)$ sangat lama


Output - DesainDanAnalisisAlgoritma (run)
run:
Masukkan banyak data: 10000000

Waktu eksekusi untuk 10000000 data

Metode Waktu(ns)
BUILD STOPPED (total time: 651 minutes 33 seconds)
5:4
INS