Perbandingan Kecepatan Gabungan Algoritma Utama *Quick Sort* dan *Merge Sort* dengan Algoritma Tambahan *Insertion Sort*, *Bubble Sort* dan *Selection Sort*

Muhammad Ezar Al Rivan[#1]

[#]*Teknik Informatika, STMIK Global Informatika MDP*
Jalan Rajawali No.14 Palembang
[1]meedzhar@mdp.ac.id

Abstract — **Data ordering should be done before data processing. The sorting algorithm has its own strengths and weaknesses. By taking strengths of each algorithm we could combine the algorithms to give better performance. Quick Sort and Merge Sort are algorithms that divide the data into parts and each part is divided again into sub-section until one element. In this experiment data is divided into parts that have size not more than threshold. Threshold used to determine which sorting algorithm will be used prime algorithm or additional algorithm. The step to divide data into one element is skipped because threshold condition has been satisfied. This step is replaced by Insertion Sort, Bubble Sort and Selection Sort. Then each part is joined with other as usual. Data size and data type may affect excecution time so this experiment use 5 data sizes and 3 types of data. The sorting algorithms that dominate in experiments are Merge-Insertion Sort and Merge-Selection Sort.**

Keywords— **Combine, Merge-Insertion Sort, Merge-Selection Sort, Replacement, Sorting Algorithm**

I. PENDAHULUAN

Kemajuan teknologi telah memberikan pengaruh terhadap perkembangan data. Data menjadi lebih besar dan bervariasi. Dengan meningkatnya jumlah data maka pengolahan data menjadi lebih kompleks. Sebelum dilakukan pengolahan data, ada proses-proses yang dilakukan diantaranya yaitu pengurutan. Algoritma pengurutan memiliki keunggulan dan kelemahan masing-masing yang tergantung oleh jumlah data. Algoritma *Quick Sort* dan *Merge Sort* merupakan algoritma yang memiliki keunggulan dalam melakukan pengurutan dengan jumlah data yang besar. Algoritma *Insertion Sort*, *Bubble Sort* dan *Selection Sort* memiliki keunggulan dalam mengurutkan data dengan jumlah kecil. *Quick Sort* dan *Merge Sort* menggunakan konsep *divide and conquer* yaitu dengan membagi-bagi data ke dalam sub-sub bagian kemudian dibagi lagi sampai bagian terkecil kemudian diurutkan per sub-sub bagian kecil. Proses pengurutan sub-sub bagian kecil ini akan diganti dengan menggunakan algoritma *Insertion Sort*, *Bubble Sort* dan *Selection Sort*.

Pada penelitian [1], perbandingan dilakukan terhadap algoritma *Merge Sort* dengan *Insertion Sort*. Hasil dari penelitian tersebut bahwa untuk data dengan jumlah data kurang dari 100 *Insertion Sort* merupakan algoritma yang lebih cepat dibandingkan dengan *Merge Sort*. Namun untuk jumlah data lebih dari 100 maka *Merge Sort* algoritma yang lebih cepat.

Pada penelitian [2], perbandingan juga dilakukan terhadap algoritma *Merge Sort* dan *Insertion Sort* namun perbedaan dengan penelitian [1] yaitu jumlah data yang digunakan dalam eksperimen. Hasil penelitian menunjukkan bahwa *Merge Sort* lebih cepat dibandingkan *Insertion Sort* untuk data dalam jumlah besar.

Pada penelitian [3], perbandingan dilakukan terhadap algoritma *Quick Sort* dan *Insertion Sort*. Hasil dari penelitian tersebut bahwa untuk data dengan jumlah data kurang dari 100 *Insertion Sort* merupakan algoritma yang lebih cepat dibandingkan dengan *Quick Sort*. Namun untuk jumlah data lebih dari 100 maka *Quick Sort* algoritma yang lebih cepat.

Penelitian lain [4] membandingkan lima algoritma yaitu *Quick Sort, Merge Sort, Selection Sort, Bubble Sort* dan *Insertion Sort*. Hasil penelitian *Quick Sort* merupakan algoritma yang cepat jika data yang digunakan 1000 namun apabila data hanya 100 maka *Insertion Sort* merupakan algoritma yang cepat.

Pada penelitian [5], perbandingan dilakukan terhadap lima algoritma *Bubble Sort, Selection Sort, Insertion Sort, Merge Sort* dan *Quick Sort*. Hasil dari penelitian *Quick Sort* merupakan algoritma yang paling cepat. Untuk data dengan skala kecil *Insertion Sort* dan *Selection Sort* yang digunakan. Untuk data memiliki pola dan aturan tertentu *Bubble Sort* dan *Insertion Sort* yang digunakan. Untuk data skala besar

Quick Sort dan *Merge Sort* yang digunakan. Pada penelitian [6] , enam algoritma yang dibandingkan yaitu *Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Bubble Sort* dan *Comparison Sort*. *Quick Sort* merupakan algoritma yang tercepat.

Berbagai penelitian dilakukan untuk meningkatkan kecepatan algoritma pengurutan, seperti yang dilakukan pada [7]–[9]. Pada penelitian [7] peningkatan dilakukan dengan cara mengoptimasi *Bubble Sort* dan *Selection Sort.* Optimasi dilakukan dengan mengurangi jumlah pertukaran data yang terjadi selama proses pengurutan. Penelitian [8] meningkatkan kecepatan algoritma *Bubble Sort* dan *Selection Sort*. Optimasi dilakukan dengan cara menghindari proses perbandingan data dengan data yang lain selama proses pengurutan. Penelitian [9] melakukan optimasi terhadap algoritma *Insertion Sort*. Optimasi dilakukan dengan cara mengurangi pencarian posisi data yang tepat dan mengurangi proses pertukaran data ke posisi yang tepat.

Pada penelitian [10] , algoritma *Quick Sort* dan *Merge Sort* digabungkan dengan *Insertion Sort*. Hasil penelitian tersebut menunjukkan bahwa hasil penggabungan memiliki waktu yang lebih cepat. *Merge-Insertion Sort* lebih cepat dibandingkan dengan *Merge Sort*. *Quick-Insertion Sort* juga lebih cepat dibandingan dengan *Quick Sort*.

Penelitian yang dilakukan bertujuan untuk mendapatkan algoritma yang memiliki waktu yang lebih cepat dalam mengurutkan data. Pada penelitian [10], algoritma yang dihasilkan memiliki waktu yang lebih cepat dibandingkan dengan algoritma biasa. Penelitian ini menggunakan algoritma lain yaitu *Bubble Sort* dan *Selection Sort.* Berdasarkan penelitian yang dilakukan [5], untuk pengurutan data dalam skala kecil dapat menggunakan *Insertion Sort*, *Bubble Sort* dan *Selection Sort* sehingga *Bubble Sort* dan *Selection Sort* dapat menjadi pembanding terhadap penelitian [10]. Selain itu pada penelitian [10] dilakukan untuk jenis data acak sehingga perlu diukur juga terhadap jenis data yang lain seperti terurut terbalik dan hampir terurut. Dengan melakukan penelitian ini dapat menambah algoritma yang dapat menjadi pilihan alternatif yang dapat disesuaikan dengan jenis data seperti untuk data acak, data terurut terbalik dan data hampir terurut.

II. TINJAUAN PUSTAKA

Terdapat banyak jenis algoritma pengurutan. Algoritma pengurutan yang populer [11] diantaranya yaitu *Bubble Sort*, *Insertion Sort*, *Selection Sort*, *Merge Sort* dan *Quick Sort*. Algoritma pengurutan memiliki kelemahan dan kelebihan masing-masing.

Terdapat dua kelas algoritma pengurutan berdasarkan kompleksitas [12]. Kelas algoritma pengurutan yang pertama yaitu algoritma yang memiliki kompleksitas $O(n^2)$. Algoritma yang memiliki kompleksitas $O(n^2)$ yaitu *Bubble Sort*, *Insertion Sort*, *Selection Sort*, *Gnome Sort*, *Cocktail Sort* dan *Radix Sort*. Kelas algoritma berikutnya yaitu algoritma yang memiliki kompleksitas $O(n \log n)$ yaitu *Heap Sort*, *Quick Sort* dan *Merge Sort*.

A. *Bubble Sort*

Sesuai dengan nama algoritma, prinsip kerja algoritma ini seperti gelembung. Bagian yang ringan akan mengapung dan bagian yang berat akan tenggelam. Algoritma *Bubble Sort* dimulai pada elemen paling kiri. Data kemudian dibandingkan dengan data pada data disebelah kanan. Apabila data lebih besar dari data di sebelah kanan maka ditukar. Proses ini akan diulang sampai data habis. Algoritma *Bubble Sort* memiliki kelebihan sederhana dan mudah diimplementasikan. Kelemahan algoritma ini yaitu tidak efisien. Algoritma ini memiliki kompleksitas waktu *Average Case* dan *Worst Case* yaitu $O(n^2)$ [5]. Algoritma *Bubble Sort* dapat dilihat pada pada Algoritma 1. Berikut adalah algoritma *Bubble Sort* dalam bahasa Java.

```
public void bubbleSort(int A[]){
  int n = A.length;
  int temp = 0;
       for(int i=0;i<n;i++){
           for(int j=1;j<(n-i);j++){
                   if(A[j-1] > A[j]){
                        temp = A[j-1];
                        A[j-1] = A[j];
                        A[j] = temp;
                   }
             }
       }
}
```

Algoritma 1. Algoritma *Bubble Sort*

B. *Insertion Sort*

Algoritma *Insertion Sort* merupakan algoritma yang mengurutkan dengan cara mencari posisi elemen seharusnya dalam data. Elemen dibandingkan dengan elemen lainya dengan mencari nilai yang lebih kecil. Pencarian posisi dilakukan sampai tidak ditemukan lagi elemen lain yang lebih kecil dari elemen yang dibandingkan. Algoritma *Insertion Sort* memiliki kompleksitas waktu *Average Case* dan *Worst Case* yaitu $O(n^2)$ [5]. Algoritma *Insertion Sort* dapat dilihat pada pada Algoritma 2. Berikut adalah algoritma *Insertion Sort* dalam bahasa Java.

```
public void insertionSort(int A[]){
  int key;
  int i;
  for(int j=1;j < A.length; j++){
       key = A[j];
       i = j-1;
       while(i >= 0 && A[i] > key){
          A[i+1] = A[i];
          i = i - 1;
       }
       A[i+1] = key;
  }
}
```

Algoritma 2. Algoritma *Insertion Sort*

C. Selection Sort

Algoritma *Selection Sort* merupakan algoritma yang mengurutkan dengan cara mencari nilai terkecil terlebih dahulu dan kemudian meletakkan pada posisi awal. Proses dilakukan lagi terhadap data yang tersisa. Pada awalnya dilakukan pencarian nilai terkecil dari seluruh elemen. Nilai terkecil kemudian diletakkan di posisi awal dengan melakukan penukaran. Proses diulang lagi dengan mencari nilai terkecil tanpa melibatkan nilai terkecil sebelumnya. Proses dilakukan sampai data tidak tersisa. Insertion Sort memiliki kompleksitas waktu yaitu $O(n^2)$ [5]. Algoritma *Selection Sort* dapat dilihat pada pada Algoritma 3. Berikut adalah algoritma *Selection Sort* dalam bahasa Java.

```
public void selectionSort(int A[]){
  for(int i=0;i<A.length-1;i++){
       int index = i;
       for(int j=i+1;j<A.length;j++) {
             if(A[j] < A[index]){
                   index = j;
             }
       }
       int smallerNumber= A[index];
       A[index] = A[i];
       A[i] = smallerNumber;
  }
}
```

Algoritma 3. Algoritma *Selection Sort*

D. Merge Sort

Algoritma *Merge Sort* menggunakan konsep *divide and conquer*. Algoritma *Merge Sort* merupakan algoritma yang melakukan pengurutan dengan melakukan pembagian data menjadi bagian kecil. Kemudian bagian-bagian kecil ini dibagi lagi menjadi sub-bagian kecil sampai didapatkan satu elemen. Pengurutan dilakukan bersamaan dengan penggabungan. Satu elemen digabungkan dengan elemen lain dengan langsung diurutkan. Gabungan elemen ini kemudian digabungkan lagi dengan gabungan elemen yang lain. Kompleksitas waktu *Average Case* dan *Worst Case* yaitu $O(n \log n)$ [5]. Algoritma *Merge Sort* dapat dilihat pada pada Algoritma 4. Berikut adalah algoritma *Merge Sort* dalam bahasa Java.

```
public void mergeSort(int A[], int p,
int r){
  if(p < r){
       int q = (p+r)/2;
       mergeSort(A, p, q);
       mergeSort(A, q+1, r);
       merge(A, p, q, r);
  }
}

private void merge(int A[],int p, int
q,int r){
  int n1 = q - p + 2;
  int n2 = r - q + 1;
  createArray(n1, n2);
  for(int i = p; i <= q ; i++){
       L[i-p] = A[i];
  }
  L[L.length-1]= Integer.MAX_VALUE;



  for(int j = q+1; j <= r ; j++){
       R[j-(q+1)] = A[j];
  }
  R[R.length-1] = Integer.MAX_VALUE;
  int i = 0;
  int j = 0;
  int l;

  for(l = p; l <= r; l++){
       if(L[i] <= R[j]){
          A[l] = L[i];
          i++;
       }
       else{
          A[l] = R[j];
          j++;
       }
 }
}
```

Algoritma 4. Algoritma *Merge Sort*

E. Quick Sort

Sama halnya dengan *Merge Sort*, *Quick Sort* juga menggunakan konsep *divide and conquer*. Algoritma *Quick Sort* membagi data ke bagian yang kecil seperti pada algoritma Merge Sort tetapi perbedaannya terletak pada *pivot* yang digunakan pada *Quick Sort*. *Pivot* digunakan sebagai acuan yang digunakan untuk melakukan pengurutan.

Data yang berada di sebelah kiri *pivot* merupakan data yang lebih kecil dari *pivot* sedangkan sebelah kanan *pivot* untuk data yang lebih besar dari *pivot*. Kompleksitas waktu *Average Case* dan *Worst Case* yaitu masing-masing $O(n \log n)$ dan $O(n^2)$ [5]. Algoritma *Quick Sort* dapat dilihat pada pada Algoritma 5. Berikut adalah algoritma *Quick Sort* dalam bahasa Java.

```
private void exchange(int A[], int
i , int j){
  int temp = A[i];
  A[i] = A[j];
  A[j] = temp;
}

public void quickSort(int A[], int p,
int r) {
  if(p < r){
       int q = partition(A, p, r);
       quickSort(A, p, q - 1);
       quickSort(A, q+1, r);
  }
}

private int partition(int A[],int p,
int r){
  int i = p;
  for(int j = p; j <= r ; j++){
       if(A[j] < A[i]){
          exchange(A, i, j);
          i++;
          exchange(A, i, j);
      }
  }
  return i;
}
```

Algoritma 5. Algoritma *Quick Sort*

Pada Tabel I dapat dilihat perbandingan kompleksitas masing-masing algoritma [5].

TABEL I
PERBANDINGAN KOMPLEKSITAS ALGORITMA PENGURUTAN

Algoritma	***Average Case***	***Worst Case***
Bubble Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n^2)$

III. METODE YANG DIUSULKAN

Pada penelitian yang dilakukan algoritma-algoritma pengurutan akan dilakukan penggabungan. Proses penggabungan dilakukan dengan cara memasukkan algoritma tambahan ke dalam algoritma utama. Pada algoritma utama data dibagi-bagi menjadi bagian-bagian kecil. Pada bagian kecil inilah algoritma tambahan yang digunakan untuk mengurutkan data. Agar dapat dibedakan kapan menggunakan algoritma utama dan kapan menggunakan algoritma tambahan maka diperlukan nilai batas. Apabila bagian-bagian kecil ini memiliki banyak data masih diatas nilai batas maka yang digunakan adalah algoritma utama namun jika lebih kecil dari nilai batas maka algoritma tambahan yang digunakan. Algoritma utama yang digunakan yaitu *Merge Sort* dan *Quick Sort*. Algoritma tambahan yaitu *Insertion Sort*, *Bubble Sort* dan *Selection Sort*. Dengan menggabungkan algoritma utama dan tambahan maka akan didapatkan 6 kandidat yang diujikan yaitu, *Merge-Insertion Sort* (MIS), *Merge-Bubble Sort* (MBS), *Merge-Selection Sort* (MSS), *Quick-Insertion Sort* (QIS), *Quick-Bubble Sort* (QSS) dan *Quick-Selection Sort* (QSS).

Enam kandidat akan diujikan dengan 5 varian jumlah data yaitu 100, 1000, 10000, 100000 dan 1000000 data. Nilai batas sebagai penentu antara algoritma utama dan algoritma tambahan juga digunakan dalam pengujian. Nilai batas merupakan nilai yang digunakan untuk menentukan kapan menggunakan algoritma utama dan algoritma tambahan. Ketika ukuran data yang dibagi hasil algoritma utama lebih dari nilai batas maka algoritma utama digunakan namun apabila ukuran data kurang dari batas maka algoritma tambahan yang digunakan. Nilai batas yang digunakan yaitu 8, 16, 32 dan 64. Jenis data yang diujikan terdapat 3 jenis yaitu acak, terurut-terbalik dan hampir terurut.

Untuk melakukan pengujian maka ada perubahan yang dilakukan pada algoritma utama dan tambahan. Perubahan yang dilakukan yaitu pada penentuan nilai batas dan ukuran data.

Berikut adalah algoritma tambahan yang dilakukan perubahan.

1. *Insertion Sort*

```
public void insertionSort(int A[],
int p, int r){
  int key;
  int i;

  for(int j = p + 1 ; j <= r; j++){
       key = A[j];
       i = j;
       while(i>p&& A[i-1] >= key){
             A[i] = A[i-1];
             --i;
       }
       A[i] = key;
  }
}
```

Algoritma 6. Algoritma *Insertion Sort* yang dimodifikasi

Algoritma 6 merupakan algoritma *Insertion Sort* yang telah dimodifikasi. Perubahan dengan menambahkan nilai p dan r. Nilai p dan r masing-masing merupakan indeks yang menunjukkan awal dan akhir bagian data. Nilai p digunakan untuk menunjuk indeks dimana permulaan bagian data yang akan diurutkan. Nilai r digunakan untuk menunjuk indeks akhir data yang diurutkan. *Insertion Sort* digunakan ketika ukuran data hasil pembagian *Quick Sort* atau *Merge Sort* berada dibawah nilai batas.

2. *Bubble Sort*

```
public void bubbleSort(int A[], int
p, int r){
     int temp = 0;
     for(int i=p;i<r;i++){
      for(int j=p+1;j<(r+1);j++){
       if(A[j-1] > A[j]){
             temp = A[j-1];
             A[j-1] = A[j];
             A[j] = temp;
       }
      }
     }
}
```

Algoritma 7. Algoritma *Bubble Sort* yang dimodifikasi

Algoritma 7 merupakan algoritma *Bubble Sort* yang telah dimodifikasi. Perubahan *Bubble Sort* dengan menambahkan nilai p dan r. Nilai p dan r merupakan indeks awal dan akhir dari bagian data yang diperoleh dari hasil pembagian *Merge Sort* dan *Quick Sort*. Ketika panjang bagian data kurang dari nilai batas maka *Bubble Sort* yang digunakan untuk mengurutkan bagian data.

3. *Selection Sort*

```
public void selectionSort(int A[],int
p,int r){
      for(int i=p;i<r;i++){
       int index = i;
       for(int j= i+1;j<r+1;j++){
             if(A[j] < A[index]){
                   index = j;
             }
       }
     int smallerNumber = A[index];
     A[index] = A[i];
     A[i] = smallerNumber;
  }
}
```

Algoritma 8. Algoritma *Selection Sort* yang dimodifikasi

Algoritma 8 merupakan algoritma *Selection Sort* yang telah dimodifikasi. Perubahan *Selection Sort* dengan menambahkan nilai p dan r. Nilai p dan r merupakan indeks awal dan akhir dari bagian data yang diperoleh dari hasil pembagian *Merge Sort* dan *Quick Sort*. Ketika panjang bagian data kurang dari nilai batas maka *Selection Sort* yang digunakan untuk mengurutkan bagian data.

Berikut adalah algoritma utama yang dilakukan perubahan.

1. *Merge Sort*

```
public void mergeSort(int A[], int
p, int r){
     if (p < r){
      if((r - p + 1) > batas){
            int q = (p+r)/2;
            mergeSort(A, p, q);
            mergeSort(A,q+1, r);
            merge(A, p, q, r);
      }
      else{
         //insertionSort(A,p,r);
         //bubbleSort(A, p, r);
         //selectionSort(A, p,r);
      }
     }
}
```

Algoritma 9. Algoritma *Merge Sort* yang dimodifikasi

Algoritma 9 merupakan algoritma *Merge Sort* yang telah dimodifikasi. Perubahan algoritma *Merge Sort* dilakukan dengan menambahkan kondisi. Apabila ukuran data lebih besar dari batas maka dilakukan *Merge Sort* seperti biasa. Namun jika ukuran data kurang dari batas maka dilakukan pengurutan menggunakan *Insertion Sort, Bubble Sort* atau *Selection Sort*. Nilai p dan r digunakan untuk penunjuk awal dan akhir dari data.

2. *Quick Sort*

```
public  void  quickSort(int  A[],
int p, int r) {
     if (p < r){
      if((r - p + 1) > batas){
      int q = partition(A, p, r);
      quickSort(A, p, q - 1);
      quickSort(A, q+1, r);
     }
     else{
        //insertionSort(A,p,r);
        //bubbleSort(A, p, r);
        //selectionSort(A,p,r);
     }
    }
}
```

Algoritma 10. Algoritma *Quick Sort* yang dimodifikasi

Sama seperti *Merge Sort,* perubahan dilakukan dengan menambahkan kondisi. Apabila ukuran data lebih besar dari batas maka dilakukan pengurutan dengan menggunakan *Quick Sort* biasa. Namun apabila ukuran data lebih kecil maka pengurutan dilakukan dengan *Insertion Sort, Bubble Sort* atau *Selection Sort*. Algoritma 10 merupakan algoritma *Quick Sort* yang telah dimodifikasi.

Data yang digunakan dalam eksperimen ini diperoleh dari hasil pengacakan. Dalam eksperimen ini angka yang diacak merupakan angka dengan kemunculan 1 kali. Pengacakan data dilakukan dengan menggunakan program *generator*. Sebelum mengukur waktu pengurutan terlebih dahulu menjalankan program *generator*. Data yang diacak disimpan pada *file* agar perbandingan yang dilakukan *fair*. Untuk memenuhi jenis data terurut terbalik maka data acak yang disimpan akan diurutkan dengan cara terbalik lalu disimpan pada *file* dilakukan dengan menggunakan program. Untuk memenuhi jenis data hampir terurut juga menggunakan program namun tidak semua data yang diurutkan. Sama seperti jenis data yang lain, jenis hampir terurut juga disimpan pada *file*. Eksperimen dilakukan dengan menggunakan *processor* i5-4210U dan RAM 8 GB. IDE yang digunakan yaitu Netbeans 8 dengan JDK 8.

IV. HASIL DAN PEMBAHASAN

Hasil eksperimen dapat dilihat pada tabel berikut.

TABEL II
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 100 DAN JENIS DATA *RANDOM* (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	108624	107342	111617	95367
MIS	63292	47897	48752	62865
MBS	71846	75695	105631	173628
MSS	64576	53884	62438	83820
QS	176194	146258	111190	113756
QIS	176194	156521	107341	87241
QBS	169779	213827	150535	174056
QSS	168923	131717	109908	178760

Pada Tabel II menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 100 data *random*. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge-Insertion Sort* (MIS) dengan nilai batas 16.

TABEL III
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 1000 DAN JENIS DATA *RANDOM* (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	1060583	1016535	924589	3977187
MIS	635923	558089	623948	2770774
MBS	762081	870704	1457019	3901065
MSS	664575	639772	750533	1690518
QS	881396	869849	759087	1119172
QIS	911760	1044760	753955	1280398
QBS	883535	1170491	1247040	3319882
QSS	771916	800142	972059	1558373

Pada Tabel III menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 1000 data *random*. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge Insertion Sort* (MIS) dengan nilai batas 16.

TABEL IV
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 10000 DAN JENIS DATA *RANDOM* (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	5398283	5330714	5272126	5120308
MIS	4912895	4183744	6239052	8311893
MBS	4205982	5200707	8642898	13014383
MSS	5413679	5228077	4619524	10426645
QS	5568062	5414107	5562075	5605695
QIS	5882388	6015389	7142258	7429215
QBS	5388447	7075545	7238480	14303761
QSS	6490513	4554521	8113462	10357365

Pada Tabel IV menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 10000 data *random*. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge-Insertion Sort* (MIS) dengan nilai batas 16.

TABEL V
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 100000 DAN JENIS DATA *RANDOM* (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	34878221	47607358	33434032	35165178
MIS	32784425	44154903	41642433	33410938
MBS	35645006	52450546	53623175	39808223
MSS	35957622	46392392	46733232	35829325
QS	37991119	37430891	32415786	20340105
QIS	29842589	30326266	27960909	23644164
QBS	25276949	34466818	39351916	37199958
QSS	23405105	31983000	25785431	34086205

Pada Tabel V menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 100000 data *random*. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Quick Sort* (QS) dengan nilai batas 64.

TABEL VI
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 1000000 DAN JENIS DATA *RANDOM* (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	2,56E+08	2,51E+08	2,46E+08	2,51E+08
MIS	1,89E+08	1,68E+08	1,65E+08	1,59E+08
MBS	2,06E+08	2,22E+08	2,47E+08	3,24E+08
MSS	1,71E+08	1,73E+08	2,01E+08	2,06E+08
QS	1,55E+08	1,68E+08	1,55E+08	1,57E+08
QIS	1,51E+08	1,64E+08	1,44E+08	1,41E+08
QBS	1,57E+08	1,66E+08	1,86E+08	3,69E+08
QSS	1,52E+08	1,55E+08	1,77E+08	2,88E+08

Pada Tabel VI menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 1000000 data *random*. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Quick-Insertion Sort* (QIS) dengan nilai batas 64.

TABEL VII
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 100 DAN JENIS DATA TERURUT TERBALIK (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	101354	159515	94512	102637
MIS	97505	90235	67142	96650
MBS	192873	89807	115039	193300
MSS	94939	57733	61155	101354
QS	540983	803135	533285	559800
QIS	490520	501211	523877	435780
QBS	505060	537990	518317	524305
QSS	461867	503349	480684	478546

Pada Tabel VII menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 100 data terurut terbalik. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge-Selection Sort* (MSS) dengan nilai batas 16.

TABEL VIII
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 1000 DAN JENIS DATA TERURUT TERBALIK (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	8,9E+05	8,8E+05	9,7E+05	1,2E+06
MIS	6,2E+05	6,1E+05	8,8E+05	1,2E+06
MBS	7,6E+05	9,0E+05	1,6E+06	2,4E+06
MSS	6,0E+05	5,8E+05	8,3E+05	9,9E+05
QS	1,1E+07	1,0E+07	1,3E+07	1,1E+07
QIS	1,2E+07	1,2E+07	1,1E+07	1,1E+07
QBS	1,3E+07	1,0E+07	1,1E+07	1,1E+07
QSS	1,2E+07	1,2E+07	1,2E+07	1,0E+07

Pada Tabel VIII menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 1000 data terurut terbalik. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge-Selection Sort* (MSS) dengan nilai batas 16.

TABEL IX
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 10000 DAN JENIS DATA TERURUT TERBALIK (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	5443187	5210115	4529717	5012111
MIS	5178897	5331569	5852880	9585021
MBS	6870698	6017100	8065136	8162642
MSS	5989729	6852736	5618526	5961504
QS	2,65E+08	2,60E+08	2,42E+08	2,40E+08
QIS	2,28E+08	2,25E+08	2,24E+08	2,25E+08
QBS	2,13E+08	2,28E+08	2,18E+08	2,32E+08
QSS	2,10E+08	2,14E+08	2,17E+08	2,21E+08

Pada Tabel IX menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 10000 data terurut terbalik. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge Sort* (MS) dengan nilai batas32.

TABEL X
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 100000 DAN JENIS DATA TERURUT TERBALIK (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	33244581	34675513	25678517	28052427
MIS	42103872	42362176	37652844	35508157
MBS	33428472	37202096	36295468	34353917
MSS	30209517	33988700	45182130	31261974
QS	1,97E+10	2,00E+10	2,15E+10	2,22E+10
QIS	2,01E+10	2,06E+10	2,15E+10	2,11E+10
QBS	1,98E+10	2,09E+10	2,13E+10	2,11E+10
QSS	2,00E+10	2,11E+10	2,14E+10	2,13E+10

Pada Tabel X menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 100000 data terurut terbalik. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge Sort* (MS) dengan nilai batas 32.

TABEL XI
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 1000000 DAN JENIS DATA TERURUT TERBALIK (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	1,61E+08	1,66E+08	1,59E+08	1,71E+08
MIS	9,58E+07	1,02E+08	1,00E+08	1,09E+08
MBS	1,14E+08	1,06E+08	1,24E+08	1,74E+08
MSS	8,99E+07	1,09E+08	1,30E+08	1,93E+08
QS	2,14E+12	2,13E+12	2,06E+12	2,06E+12
QIS	2,01E+12	2,01E+12	1,97E+12	1,95E+12
QBS	2,02E+12	2,03E+12	1,96E+12	1,95E+12
QSS	2,02E+12	2,00E+12	1,97E+12	1,95E+12

Pada Tabel XI menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 1000000 data terurut terbalik. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge-Selection Sort* (MSS) dengan nilai batas 8.

TABEL XII
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 100 DAN JENIS DATA HAMPIR TERURUT (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	203563	93228	108624	106058
MIS	120171	44049	33784	31218
MBS	135138	67141	104775	146257
MSS	119744	53457	62010	80399
QS	117178	73129	82537	73556
QIS	98361	108197	65858	54312
QBS	74412	76551	96222	149252
QSS	71418	82110	101782	98361

Pada Tabel XII menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 100 data hampir terurut. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge-Insertion Sort* (MIS) dengan nilai batas 64.

TABEL XIII
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 1000 DAN JENIS DATA HAMPIR TERURUT (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	855309	889949	947254	851032
MIS	528581	434069	431931	389594
MBS	905344	811688	1309051	1862435
MSS	659015	577333	724019	954525
QS	2153241	1559229	1533569	2406412
QIS	2709191	2255023	2509049	2500924
QBS	2539413	2219954	2612114	2527010
QSS	2415820	2390161	2678828	2401707

Pada Tabel XIII menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 1000 data hampir terurut. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge-Insertion Sort* (MIS) dengan nilai batas 64.

TABEL XIV
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 10000 DAN JENIS DATA HAMPIR TERURUT (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	4715746	5100208	4883387	3401564
MIS	4813252	4580608	4511756	6111611
MBS	4405269	4957372	8051024	8230212
MSS	3127865	3316889	6289943	8136983
QS	50547911	36470806	48309994	36911291
QIS	19574603	20419649	20361488	18774889
QBS	24670963	18790713	27717146	24750079
QSS	28051572	26358488	27305742	21665406

Pada Tabel XIV menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 10000 data hampir terurut. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge-Selection Sort* (MSS) dengan nilai batas 8.

TABEL XV
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 100000 DAN JENIS DATA HAMPIR TERURUT (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	27609377	34107160	27774452	31139238
MIS	37401383	23002254	25705031	29551356
MBS	42971583	51862093	45728244	29717714
MSS	21280090	37989835	48487044	30093623
QS	1,10E+09	9,90E+08	1,22E+09	9,50E+08
QIS	1,68E+09	1,24E+09	1,19E+09	1,20E+09
QBS	1,28E+09	1,27E+09	1,18E+09	1,20E+09
QSS	1,03E+09	1,46E+09	1,29E+09	1,42E+09

Pada Tabel XV menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 100000 data hampir terurut. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge-Selection Sort* (MSS) dengan nilai batas 8.

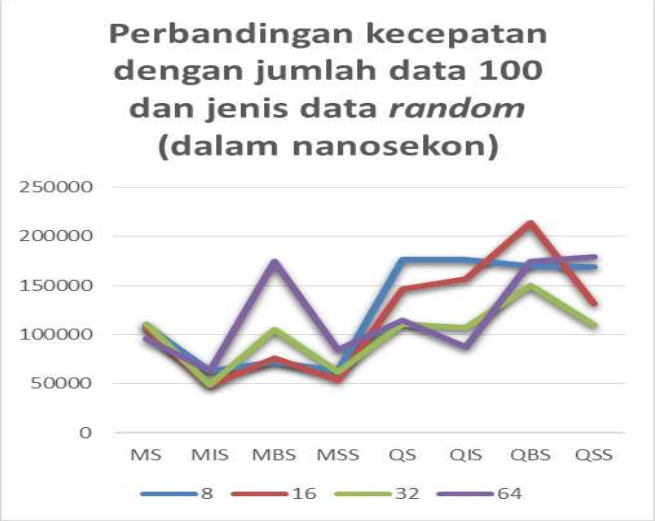
TABEL XVI
PERBANDINGAN KECEPATAN KANDIDAT ALGORITMA DENGAN JUMLAH DATA 1000000 DAN JENIS DATA HAMPIR TERURUT (DALAM NANOSEKON)

Algoritma	Batas			
	8	16	32	64
MS	1,69E+08	1,73E+08	2,12E+08	1,73E+08
MIS	1,06E+08	1,16E+08	8,64E+07	8,13E+07
MBS	1,39E+08	1,49E+08	1,37E+08	1,61E+08
MSS	1,11E+08	1,04E+08	1,14E+08	1,11E+08
QS	9,27E+10	9,00E+10	9,00E+10	9,16E+10
QIS	1,23E+11	1,18E+11	1,23E+11	1,19E+11
QBS	1,20E+11	1,17E+11	1,21E+11	1,19E+11
QSS	9,31E+10	1,20E+11	1,31E+11	1,33E+11

Pada Tabel XVI menunjukkan perbandingan waktu dari algoritma utama dan gabungan algoritma. Perbandingan dilakukan dengan membandingkan algoritma dan nilai batas terhadap 1000000 data hampir terurut. Pada bagian yang berwarna kuning merupakan algoritma yang memiliki waktu tercepat. Algoritma tercepat yaitu *Merge-Insertion Sort* (MIS) dengan nilai batas 64.

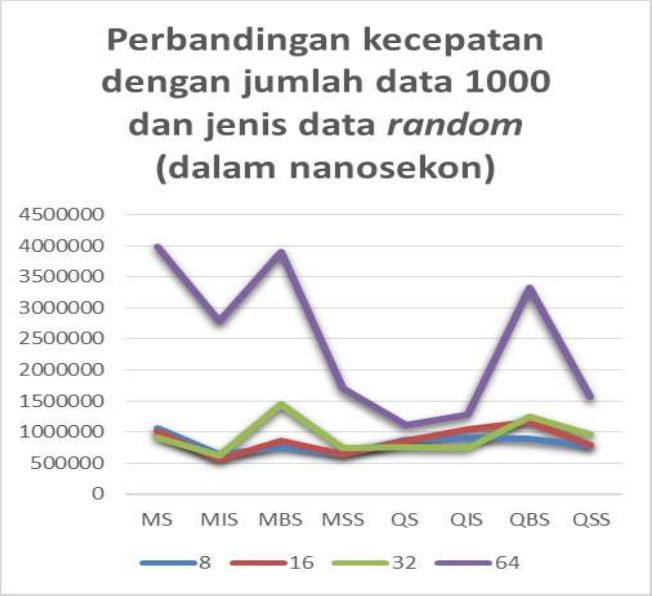
Dari masing-masing tabel kemudian dibuat grafik agar perbandingan dapat dilihat dengan mudah. Grafik dibuat sesuai dengan tabel yaitu dibuat dengan membandingkan kandidat algoritma dan nilai batas terhadap waktu.

Gambar 1 menunjukkan perbandingan kecepatan untuk ukuran data 100 dan jenis data *random*. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge-Insertion Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 16.



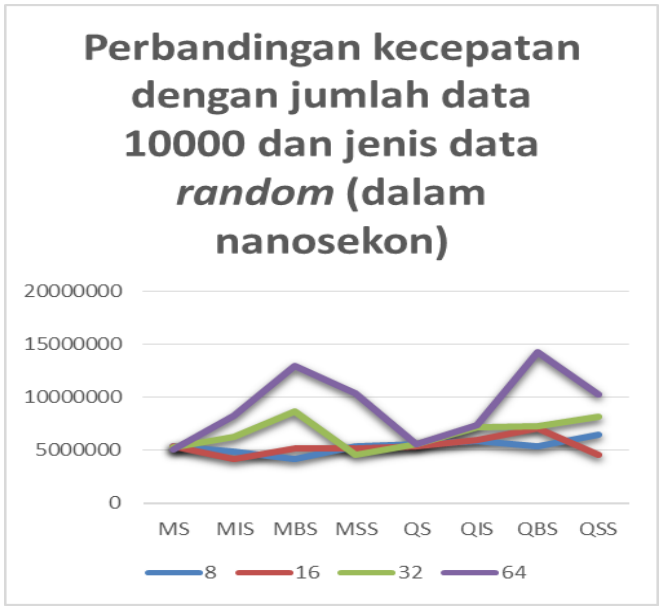
Gambar 1.Perbandingan kecepatan kandidat algoritma dengan jumlah data 100 dan jenis data *random*.

Gambar 2 menunjukkan perbandingan kecepatan untuk ukuran data 1000 dan jenis data *random*. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge-Insertion Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 16.



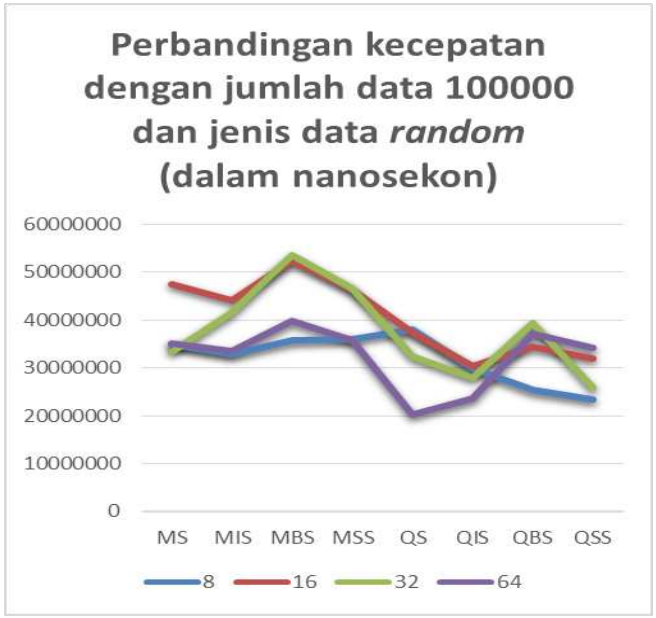
Gambar 2. Perbandingan kecepatan kandidat algoritma dengan jumlah data 1000 dan jenis data *random*.

Gambar 3 menunjukkan perbandingan kecepatan untuk ukuran data 10000 dan jenis data *random*. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge-Insertion Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 16.



Gambar 3. Perbandingan kecepatan kandidat algoritma dengan jumlah data 10000 dan jenis data *random*.

Gambar 4 menunjukkan perbandingan kecepatan untuk ukuran data 100000 dan jenis data *random*. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Quick Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 64.



Gambar 4. Perbandingan kecepatan kandidat algoritma dengan jumlah data 100000 dan jenis data *random*.

Gambar 5 menunjukkan perbandingan kecepatan untuk ukuran data 1000000 dan jenis data *random*. Algoritma

e-ISSN : 2443-2229

yang memiliki waktu paling cepat secara keseluruhan yaitu *Quick-Insertion Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 64.

Gambar 5. Perbandingan kecepatan kandidat algoritma dengan jumlah data 1000000 dan jenis data *random*.

Gambar 6 menunjukkan perbandingan kecepatan untuk ukuran data 100 dan jenis data terurut terbalik. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge-Selection Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 16.

Gambar 6. Perbandingan kecepatan kandidat algoritma dengan jumlah data 100 dan jenis data terurut terbalik.

Gambar 7 menunjukkan perbandingan kecepatan untuk ukuran data 1000 dan jenis data terurut terbalik. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge-Selection Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 16.

Gambar 7. Perbandingan kecepatan kandidat algoritma dengan jumlah data 1000 dan jenis data terurut terbalik.

Gambar 8 menunjukkan perbandingan kecepatan untuk ukuran data 10000 dan jenis data terurut terbalik. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 32.

Gambar 8. Perbandingan kecepatan kandidat algoritma dengan jumlah data 10000 dan jenis data terurut terbalik.

Gambar 9 menunjukkan perbandingan kecepatan untuk ukuran data 100000 dan jenis data terurut terbalik. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 32.

Gambar 9. Perbandingan kecepatan kandidat algoritma dengan jumlah data 100000 dan jenis data terurut terbalik.

Gambar 10 menunjukkan perbandingan kecepatan untuk ukuran data 1000000 dan jenis data terurut terbalik. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge-Selection Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 8.

Gambar 10. Perbandingan kecepatan kandidat algoritma dengan jumlah data 1000000 dan jenis data terurut terbalik.

Gambar 11 menunjukkan perbandingan kecepatan untuk ukuran data 100 dan jenis data hampir terurut. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge-Insertion Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 64.

Gambar 11. Perbandingan kecepatan kandidat algoritma dengan jumlah data 100 dan jenis data hampir terurut..

Gambar 12 menunjukkan perbandingan kecepatan untuk ukuran data 1000 dan jenis data hampir terurut. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge-Insertion Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 64.

Gambar 12. Perbandingan kecepatan kandidat algoritma dengan jumlah data 1000 dan jenis data hampir terurut.

Gambar 13 menunjukkan perbandingan kecepatan untuk ukuran data 10000 dan jenis data hampir terurut. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge-Selection Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 8.

Gambar 13. Perbandingan kecepatan kandidat algoritma dengan jumlah data 10000 dan jenis data hampir terurut..

Gambar 14 menunjukkan perbandingan kecepatan untuk ukuran data 100000 dan jenis data hampir terurut. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge-Selection Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 8.

Gambar 14. Perbandingan kecepatan kandidat algoritma dengan jumlah data 100000 dan jenis data hampir terurut..

Gambar 15 menunjukkan perbandingan kecepatan untuk ukuran data 1000000 dan jenis data hampir terurut. Algoritma yang memiliki waktu paling cepat secara keseluruhan yaitu *Merge-Insertion Sort*. Untuk nilai batas yang memberikan waktu tercepat yaitu 64.

Gambar 15. Perbandingan kecepatan kandidat algoritma dengan jumlah data 1000000 dan jenis data hampir terurut.

Pada Tabel XVII dapat dilihat algoritma-algoritma yang memiliki waktu paling cepat untuk masing-masing ukuran data dan jenis data.

TABEL XVII
KANDIDAT ALGORITMA TERCEPAT TERHADAP JENIS DATA DAN UKURAN DATA

Ukuran	Jenis Data		
	Random	**Terurut terbalik**	**Hampir terurut**
100	MIS, batas 16	MIS, batas 16	MIS, batas 64
1000	MIS, batas 16	MSS, batas 16	MIS, batas 64
10000	MIS, batas 16	MS, batas 32	MSS, batas 8
100000	QS, batas 64	MS, batas 32	MSS, batas 8
1000000	QIS, batas 64	MSS, batas 8	MIS, batas 64

Penggabungan algoritma utama dan algoritma tambahan memberikan pengaruh jika dibandingkan dengan algoritma utama saja. Penggabungan algoritma utama dan tambahan dapat memberikan waktu yang lebih cepat. Penggabungan algoritma utama dan tambahan juga dapat memberikan waktu yang lebih lama.

Berdasarkan hasil eksperimen, walaupun *Quick Sort* dan *Merge Sort* memiliki cara kerja yang sama, *Merge Sort*

memiliki waktu yang lebih cepat dibandingkan dengan *Quick Sort*. Selisih waktu terbesar antara *Merge Sort* dan *Quick Sort* terjadi pada saat jenis data yang digunakan terurut terbalik dan hampir terurut.

Jika melihat pada Tabel XVII, algoritma yang mendominasi dalam eksperimen yaitu *Merge-Insertion Sort* kemudian diikuti *Merge-Selection Sort*. *Merge Sort* masih dapat dikatakan lebih baik pada kasus jenis data terurut terbalik dengan ukuran data 10000 sampai 100000. *Quick Sort* memiliki kelebihan pada kasus jenis data *random* dan ukuran data 100000. *Quick-Insertion Sort* dapat digunakan untuk kasus jenis data random dan ukuran data 1000000.

V. KESIMPULAN

Penggabungan algoritma utama *Quick Sort* dan *Merge Sort* dengan algoritma tambahan *Insertion Sort*, *Bubble Sort* dan *Selection Sort* dapat memberikan pengaruh terhadap waktu. Pengaruh yang diberikan oleh penggabungan ini dapat mempercepat dan juga dapat memperlambat. Selain penggabungan algoritma, nilai batas, jumlah data juga mempengaruhi waktu.

Penggabungan algoritma yang mendominasi dalam eksperimen yang dilakukan yaitu *Merge-Insertion Sort* dan *Merge-Selection Sort*. Hasil penggabungan algoritma dapat memberikan waktu yang lebih cepat dibandingkan dengan *Merge Sort*. Algoritma *Merge Sort* menghasilkan waktu yang lebih cepat dibandingkan dengan *Quick Sort*.

DAFTAR PUSTAKA

[1] R. Hibbler, “Merge sort,” *Dept. Comput. Sci. Florida Inst. Technol. Florida, USA.*, 2008.

[2] Arief Hendra Saptadi and D. W. Sari, “Analisis algoritma insertion sort, merge sort dan implementasinya dalam bahasa pemrograman c++,” vol. 8, pp. 1–8, 2012.

[3] R. Hibbler, “Quick sort,” *Dept. Comput. Sci. Florida Inst. Technol. Florida, USA.*, 2008.

[4] P. Sareen, “International Journal of Advanced Research in Computer Science and Software Engineering Comparison of Sorting Algorithms (On the Basis of Average Case),” *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 3, pp. 522–532, 2013.

[5] Y. Yang, P. Yu, and Y. Gan, “Experimental study on the five sort algorithms,” *2011 2nd Int. Conf. Mech. Autom. Control Eng. MACE 2011 - Proc.*, pp. 1314–1317, 2011.

[6] I. M. Alturani, A. Mahmoud, and I. Alturani, “Review on Sorting Algorithms A Comparative Study,” *Int. J. Comput. Sci. Secur.*, vol. 7, no. 3, pp. 120–126, 2013.

[7] P. Y. Padhye, “Circular Sort : A New Improved Version of Insertion Sort Algorithm,” *Sch. Comput. Math. Deakin Univ. Geelong, Victoria 3217*, pp. 1–10.

[8] M. Khairullah, “Enhancing Worst Sorting Algorithms,” *Int. J. Adv. Sci. Technol.*, vol. 56, pp. 13–26, 2013.

[9] J. Alnihoud and R. Mansi, “An enhancement of major sorting algorithms,” *Int. Arab J. Inf. Technol.*, vol. 7, no. 1, pp. 55–62, 2010.

[10] M. E. Al Rivan, “Perbandingan Performa Kombinasi Algoritma Pengurutan Quick-Insertion Sort dan Merge-Insertion Sort,” vol. 2, no. 1, pp. 6–10, 2016.

[11] C. Canaan, “Popular sorting algorithms,” no. April, pp. 62–71, 2011.

[12] D. T. V. D. Rao and B. Ramesh, “Experimental Based Selection of Best Sorting Algorithm,” vol. 2, no. 4, pp. 2908–2912, 2012.