BAB I

PENDAHULUAN

1.1 Latar belakang

Teknologi informasi telah berkembang sangat pesat pada masa ini. Pencarian informasi yang berjumlah kecil dan besar dalam waktu yang cepat, sangat dibutuhkan sebagai upaya efisiensi waktu. Pencarian sebuah dokumen akan lebih cepat apabila informasi-informasi mengenai dokumen yang akan dicari tersebut diurutkan terlebih dahulu daripada pencarian-pencarian dokumen tanpa pengurutan. Pengurutan data (*sorting*) merupakan suatu proses dimana suatu susunan data yang semula dalam kondisi acak dapat menjadi terurut, baik dari data terkecil sampai dengan data yang terbesar, atau sebaliknya dari data terbesar sampai dengan data terkecil (Wahyudi, 2009). Untuk dapat melakukan pengurutan data, sebuah algoritma pengurutan data (*sorting*) sangat penting dan dibutuhkan dalam pengolahan data informasi.

Namun sayangnya, setiap algoritma pengurutan data tersebut tidak lepas dari kekurangan-kekurangan yang ada seperti kompleksitas waktu pada *best case* (kasus terbaik), *average case* (kasus rata-rata), *worst case* (kasus terburuk). Dalam pengurutan data, kompleksitas berupa waktu adalah tolok ukur seberapa efektif suatu algoritma. Algoritma yang efektif adalah algoritma yang meminimumkan kebutuhan waktu dan ruang. Keefektifan algoritma diukur dari jumlah waktu dan ruang memori yang dibutuhkan untuk menjalankan algoritma tersebut (Nugraha, 2012).

Pada penelitian yang dilakukan oleh Ocampo (2008), telah dibandingkan kecepatan waktu eksekusi pada kompleksitas waktu rata-rata (*average case*) pada lima algoritma pengurutan data yaitu *Quick sort*, *Shell sort*, *Selection sort*, *Insertion sort* dan *Bubble sort*. Dari lima algoritma pengurutan data tersebut, waktu eksekusi *Quick sort* lebih cepat secara signifikan daripada algoritma yang lain pada ukuran data yang lebih besar, walaupun lebih lambat pada ukuran data yang kecil. Pada ukuran data yang lebih kecil, waktu eksekusi algoritma *Shell sort* lebih cepat dibandingkan algoritma *Quick sort* dan algoritma lainnya.

Berdasarkan hal tersebut di atas, maka dapat dilakukan perbaikan kompleksitas waktu pengurutan data dari kekurangan dan kelebihan masing-masing algoritma tersebut, sehingga proses pengurutan pada data kecil maupun data besar dapat dilakukan lebih

efektif dan efisien. Kekurangan yang terdapat pada algoritma pengurutan data dapat diminimalisasi untuk menghasilkan suatu algoritma yang lebih baik dari algoritma sebelumnya. Caranya dapat berupa perbaikan efisiensi dengan mengkombinasikan (*hybrid*) antara algoritma satu dengan algoritma yang lain. Oleh karena itu dengan dibuat sebuah algoritma pengurutan data kombinasi (*hybrid*) dari algoritma *Shell sort* dan algoritma *Quick sort* diharapkan dapat menutupi kekurangan pada masing-masing algoritma itu.

Namun, sekalipun algoritma *hybrid* tersebut mampu menutupi kekurangan pada masing-masing algoritma *sorting* dan memperbaiki efisiensi pada kompleksitas waktu, tetapi tetap saja akan mengalami keterbatasan waktu dalam hal kecepatan pemrosesan bila dilakukan dengan komputasi serial. Karena pada umumnya, sebuah algoritma itu dibangun dengan menggunakan komputasi serial, dimana hanya satu instruksi yang bisa berjalan pada satu waktu saja. Hal ini akan memunculkan permasalahan untuk eksekusi program yang membutuhkan sumber daya komputasi (prosesor dan memori) yang besar, yaitu waktu eksekusi yang panjang (Syaputra dan Akbar, 2011).

Permasalahan yang muncul dalam komputasi serial tersebut dapat diatasi dengan menggunakan komputasi paralel. Komputasi paralel adalah penggunaan beberapa sumber daya komputasi secara simultan untuk menyelesaikan suatu permasalahan komputasi. Dimana, beberapa komputer dengan banyak prosesor melakukan pekerjaan (pemecahan masalah) secara bersamaan (Syaputra dan Akbar, 2011).

Oleh karena itu, dengan mengimplementasikan pemrograman paralel dan mengkombinasikan (*hybrid*) algoritma *Shell sort* dan *Quick sort*, diharapkan dapat meningkatkan kecepatan waktu komputasi pada pengurutan data yang berukuran besar maupun berukuran kecil yang lebih efektif dan efisien. Kontribusi penelitian ini agar *programmer* bisa menggunakan metode kombinasi (*hybrid*) pada algoritma *sorting*, baik dengan menggunakan dua algoritma *sorting* maupun lebih dari dua algoritma *sorting* pada aplikasi yang menggunakan atau memerlukan pengurutan data.

1.2 Rumusan Masalah

Berdasarkan latar belakang, masalah yang akan dirumuskan dan akan dicari penyelesaiannya adalah bagaimana mengkombinasikan (*hybrid*) algoritma *Shell sort* dan *Quick sort* baik serial maupun paralel, kemudian bagaimana menganalisis kecepatan

pemrosesan (*speed up*) antara algoritma *hybrid* paralel dari *Shell sort* dan *Quick sort* dengan algoritma *hybrid* serial dari *Shell sort* dan *Quick sort*, serta bagaimana menghitung tingkat efisiensi yang dicapainya.

1.3 Tujuan Penelitian

Adapun tujuan yang ingin dicapai dari penelitian Tugas Akhir ini adalah dapat menghasilkan dua buah algoritma pengurutan data kombinasi (*hybrid*) yaitu algoritma *hybrid* serial dan algoritma *hybrid* paralel, dan dapat mengkalkulasi kecepatan pemrosesan (*speed up*) dan efisiensi dari *hybrid* algoritma *Shell sort* dan *Quick sort*.

1.4 Batasan Masalah

Pada penelitian ini mempunyai beberapa batasan masalah yang bertujuan untuk memfokuskan penelitian agar tidak menyimpang dari pokok permasalahan yang akan diteliti. Adapun batasan masalah pada penelitian Tugas Akhir ini antara lain:

a. Penggunaan pustaka pemrograman paralel pada algoritma yaitu *Message Passing Interface* (MPI) dengan komunikasi *point to point*.
b. Jumlah komputer yang digunakan adalah dua komputer
c. Besar ukuran ruang memori (RAM) yang digunakan pada komputer *master* adalah 512 *megabyte*, dan besar ruang memori pada komputer *slave* adalah 384 *megabyte*.

1.5 Manfaat Penelitian

Adapun manfaat yang bisa didapatkan dari penelitian ini adalah :

a. Manfaat ilmiah

   Dapat dijadikan sebagai salah satu referensi ilmu pengetahuan tentang algoritma pengurutan data (*sorting*), dan dapat juga digunakan sebagai tinjauan pustaka bagi peneliti yang ingin lebih dalam mengembangkan pemrograman paralel pada algoritma pengurutan data (*sorting*) sehingga dapat menemukan metode yang lebih efektif dan efisien dalam pengurutan data dan pemrograman paralel.

b. Manfaat terapan

   Dengan pengurutan data, suatu data dapat lebih teratur dan tersusun rapi sehingga dapat mempermudah *user* dalam melakukan pencarian data yang diinginkan. Selain itu, *user* juga bisa menerapkan algoritma *sorting* berbasis

paralel ini pada aplikasi-aplikasi yang membutuhkan pengurutan data, seperti aplikasi pada kontak telepon dalam pengurutan nama dan nomor kontak, aplikasi perbankan dalam pengurutan nama dan deposito nasabah, dan aplikasi pengurutan daftar lagu dan daftar video pada suatu *player*.