Aplikasi Pembelajaran Penyortiran Menggunakan Algoritma Super Sort Berbasis Mobile

Jimmy[1], Felix Utama[2], Felix[3], Albert Prima Laia[4]
STMIK Mikroskil, Jl. Thamrin No. 112, 124, 140, Telp. (061) 4573767, Fax. (061) 4567789
Program Studi Teknik Informatika, STMIK Mikroskil, Medan
[1]171110134@students.mikroskil.ac.id, [2]171110932@students.mikroskil.ac.id,
[3]felix.pandi@ mikroskil.ac.id, [4]albert.laia@mikroskil.ac.id

Abstrak

Aplikasi pembelajaran untuk masalah algoritma pengurutan telah banyak diciptakan untuk menyelesaikan masalah pengenalan algoritma-algoritma tersebut yang terbilang sulit kepada khalayak yang lebih luas. Akan tetapi algoritma penyortiran yang dipakai kebanyakan adalah algoritma yang sering dijumpai. Padahal banyak algoritma pengurutan yang terbilang baru dan lebih efisien untuk dipelajari. Salah satunya adalah algoritma super sort.

Tujuan dari penelitian ini adalah untuk membangun aplikasi pembelajaran algoritma super sort berbasis mobile. Algoritma super sort akan diajarkan dalam bentuk materi dan simulasi yang dapat diakses dan digunakan oleh pengguna. Simulasi pengajaran algoritma yang dibuat akan menyangkut semua proses yang terjadi pada algoritma super sort.

Pengujian terhadap aplikasi akan menggunakan black box testing. Metode black box testing yang digunakan berupa equivalence partioning, boundary value analysis dan error guessing. Ketiga metode yang telah dilakukan terhadap aplikasi menghasilkan sebuah kesimpulan berupa semua simulasi yang telah diimplementasikan didalam aplikasi pembelajaran telah layak digunakan karena mempuyai persentase keberhasilan sebesar 91.7 % dari semua pengujian yang telah dilakukan.

Kata kunci*— Aplikasi pembelajaran, black box testing, super sort*

Abstract

Many learning applications for sorting algorithm problems have been created to solve the difficult problem of introducing these algorithms to a wider audience. However, the sorting algorithms that are used are mostly algorithms that are often encountered. Even though many sorting algorithms are relatively new and more efficient to learn. One of them is the super sort algorithm.

The purpose of this study is to build a sorting learning mobile-based application using super sort algorithm. The super sort algorithm will be taught in the form of theories and simulations that can be accessed by users. Algorithm teaching simulation that is made will involve all processes that occur in the super sort algorithm.

The test of sorting learning application will be using black box testing. The black box testing methods that will be used are equivalent partioning, boundary value analysis and error guessing. The three methods that have been carried out on the application produce a conclusion in the form that all simulations that have been implemented in the learning application are suitable for use because they have a success percantage of 91.7% of all tests that have been carried out.

Keywords*— black box testing, learning application, super sort*

1. PENDAHULUAN

Pada era sekarang ini teknologi berkembang dengan pesat, terutama teknologi gawai *mobile* yang membuat hidup manusia menjadi lebih mudah. Bukti kemudahan tersebut dapat dilihat dari penggunaan gawai *mobile* sebagai media yang mengakses informasi edukasi bagi para pelajar. Aplikasi pembelajaran *mobile* adalah sebuah mekanisme pembelajaran yang membantu pengguna di dalam

memahami pesan pembelajaran dengan menggunakan perangkat *mobile* kapan saja dan di mana saja [1]. Banyak ilmu pembelajaran yang dapat diakses dan dipelajari oleh semua orang pada aplikasi pembelajaran *mobile*, contohnya algoritma. Mata kuliah algoritma dan struktur data memiliki persentase tingkat kepahaman yang rendah, yaitu 11,5% dan tingkat kesukaan hanya 8,2% [2]. Persentase ini menunjukan mahasiswa yang mengerti dan menyukai pelajaran algoritma masih sedikit.

Algoritma adalah setiap prosedur komputasi yang mengambil beberapa nilai, atau sekumpulan nilai, sebagai masukan dan menghasilkan beberapa nilai, atau kumpulan nilai, sebagai keluaran [3]. Banyak masalah yang dapat diselesaikan dengan penciptaan sebuah algoritma. Salah satu masalah yang dapat diselesaikan adalah masalah pengurutan. Masalah pengurutan biasanya dituntaskan dengan algoritma penyortiran atau biasa disebut algoritma pengurutan. Algoritma pengurutan adalah algoritma yang meletakkan elemen-elemen suatu kumpulan data dalam urutan tertentu seperti *ascending* ataupun *descending*. Banyak jenis algoritma pengurutan yang ada, seperti *bubble sort*, *selection sort*, *merge sort* dan lain sebagainya. Akan tetapi, aplikasi pembelajaran sekarang lebih cenderung menampilkan algoritma-algoritma pengurutan yang lebih dikenal sebagai dasar pembelajaran algoritma tersebut.

Sebelumnya, telah ada riset aplikasi pembelajaran algoritma pengurutan yang sudah dilakukan seperti "Aplikasi Belajar Interaktif Algoritma Sorting Berbasis Desktop" [4], namun masih memuat lima algoritma pengurutan yang sering dijumpai berupa *bubble sort*, *selection sort*, *insertion sort*, *binary insertion sort* dan *quick sort*. Padahal banyak algoritma pengurutan yang terbilang baru dan lebih efisien untuk dipelajari. Salah satunya adalah algoritma *super sort*. Algoritma ini masih terbilang baru karena diciptakan dua tahun yang lalu [5]. Salah satu contoh riset yang sudah pernah diterapkan menggunakan algoritma ini adalah "*Super Sort Algorithm Using MPI and CUDA*" [6]. Riset tersebut telah mengimplementasikan algoritma *super sort* dengan MPI dan CUDA untuk membandingkan waktu yang dibutuhkan dalam proses pengurutan. Namun, dari semua riset yang menggunakan algoritma *super sort*, belum ada riset yang melakukan implementasi algoritma tersebut menjadi sebuah aplikasi pembelajaran sehingga dapat lebih dikenal. Untuk memperkenalkan algoritma *super sort* kepada khalayak yang lebih luas maka dikembangkan sebuah aplikasi pembelajaran berdasarkan pada algoritma *super sort*.

Algoritma *super sort* merupakan algoritma yang menggunakan elemen urutan asli yang dimasukkan (*natural sequence*) yang diurutkan dalam *array* dan terdiri atas angka acak, sehingga dapat mengurangi jumlah langkah yang diperlukan untuk diurutkan [5]. Kompleksitas waktu dari algoritma *super sort* adalah $O\ (n\ log\ n)$ pada kasus rata-rata. Hal tersebut membuktikan bahwa algoritma *super sort* lebih cepat dari *bubble sort*, *insertion sort*, dan *selection sort* ketika mendapatkan kasus yang rata-rata, dimana ketiga algoritma tersebut memiliki kompleksitas waktu $O\ (n^2)$. Untuk algoritma terkenal lainnya seperti *quick sort* memiliki *worst case* yang berbanding terbalik dengan *best case* algoritma *super sort*, dimana ketika urutan kasus sudah terurut maka *quick sort* akan memerlukan waktu $O\ (n^2)$ untuk menyelesaikannya [7], tidak seperti *super sort* yang memerlukan waktu $O\ (n)$ untuk menyelesaikan *worst case* dari *quick sort* tersebut.

Dari banyaknya konsep yang ada, aplikasi yang di kembangkan akan hampir sama dengan "*Bubble Sort – play with the algorithm*" [8] dengan tujuan mengembangkan aplikasi yang mengajarkan cara kerja algoritma *super sort* kepada pengguna dengan fitur simulasi yang berbasis *mobile*. Aplikasi akan dapat memberikan gambaran visual berupa grafik batang dan angka dari langkah-langkah proses penyelesaian algoritma *super sort*.

Berdasarkan uraian di atas, maka disusunlah judul aplikasi "Aplikasi Pembelajaran Penyortiran Menggunakan Algoritma *Super Sort* Berbasis *Mobile*".

2. METODE PENELITIAN

2.1 Tinjauan Pustaka

2.1.1 Aplikasi Pembelajaran Mobile

Aplikasi pembelajaran *mobile* atau *mobile learning* merupakan sebuah alat atau servis yang menyediakan informasi kepada para pelajar menggunakan alat elektronik [9]. Aplikasi pembelajaran menyediakan konten edukasi yang membantu masyarakat untuk mendapatkan ilmu yang diinginkan. Informasi atau ilmu yang diberikan diharapkan dapat membantu pengguna dalam mempelajari sesuatu yang berguna baginya secara nyata, seperti pelajaran bahasa inggris yang membantu melancarkan kemampuan berbahasa inggris ataupun pembelajaran algoritma untuk membantu lebih mengerti langkah-langkah yang ada pada algoritma yang ingin dipelajari.

Ketika mengembangkan aplikasi pembelajaran *mobile*, ada dua aspek yang harus diperhatikan yaitu pelajar (*user*) dan *usability* [10]. Untuk bisa memudahkan pelajar dalam belajar, aplikasi yang dibuat haruslah berdasarkan analisa para *user* dari segi tingkat studi yang ingin ditargetkan. Tingkat studi yang tepat akan membuat pengguna lebih mengerti informasi yang akan diberikan. Aspek kedua terkait *usability* secara umum berupa sebuah istilah yang menunjukkan kemudahan pengguna untuk menggunakan suatu aplikasi. Aspek ini mengharuskan aplikasi yang dirancang mempunyai fitur-fitur yang mudah dipelajari untuk digunakan. Beberapa aspek yang perlu diperhatikan ketika mengembangkan aplikasi pembelajaran *mobile* yaitu [10] :

1. Navigasi pada aplikasi harus sederhana dan jelas dari halaman ke halaman.
2. Navigasi yang kompleks harus dihindari ketika mengembangkan aplikasi.
3. Mengurangi penggunaan *scrolling*.
4. Aplikasi harus *user-friendly* dan bisa membuat pengguna mengerti cara kerja aplikasi dalam beberapa menit (mudah dimengerti).
5. Tombol aksi dan informasi yang diberikan harus konsisten pada setiap halaman (diletakkan pada posisi yang sama pada setiap halamannya).
6. Tampilan harus fleksibel.
7. Memberikan informasi yang bersangkutan dengan fungsi dari aplikasi sebagai aplikasi pembelajaran. Misalnya, aplikasi pembelajaran algoritma penyortiran hanya memberikan informasi tentang algoritma penyortiran.
8. Mengurangi informasi tertulis dan menambahkan informasi dalam bentuk grafik dan animasi untuk meningkatkan motivasi pengguna dalam belajar.

2.1.2 Algoritma Penyortiran

Algoritma penyortiran adalah algoritma yang menempatkan elemen pada sebuah daftar menjadi sebuah urutan tertentu [11]. Pada dasarnya, algoritma membandingkan elemen-elemen pada *list* dan menggantinya satu sama lain sesuai dengan kriteria dari hasil yang diinginkan. Hasil yang diinginkan dapat berupa sebuah daftar elemen (*array*) yang telah terurut dari nilai terkecil sampai terbesar (*ascending*) ataupun sebaliknya (*descending*). Sebuah algoritma penyortiran dapat dikatakan baik jika *output* yang dihasilkan sesuai dengan yang diinginkan dan proses yang dilakukan efisien.

Secara garis besar, algoritma penyortiran dapat dikelompokkan menjadi dua kategori, yaitu algoritma pengurutan berbasis perbandingan (*comparison based*) dan tidak berbasis perbandingan (*non-comparison based*) [12]. Contoh algoritma *comparison sort* berupa *bubble sort*, *insertion sort* dan *quick sort.* Contoh algoritma *non comparison sort* berupa *radix sort*, *counting sort* dan *bucket sort*. Perbedaan di antara kedua penyortiran adalah elemen yang dapat diurutkan oleh keduanya, dimana *comparison based sort* dapat mengurutkan berbagai tipe data yang diberikan sedangkan *non-comparison based sort* hanya dapat mengurutkan tipe data tertentu seperti *integer*. Tetapi sebaliknya algoritma *comparison sort* lebih lambat daripada non *comparison sort.* Hal tersebut terjadi karena *comparison based sort* tidak dapat mengurutkan lebih baik dari $O\ (n\ log\ n)$, sedangkan *non-comparison based* sort dapat mengurutkan lebih baik dari $O\ (n\ log\ n)$ [13].

2.1.3 Algoritma Super Sort

Algoritma *Super Sort* adalah algoritma yang bekerja dengan memilih urutan dari elemen yang telah terurut (*natural sequence*) pada *array* yang masih belum terurut [5]. *Natural sequence* nantinya akan diambil dari *array* yang bermasalah dan dimasukkan ke dalam *array* baru. Teknik pada algoritma *super sort* yang digunakan untuk mendapatkan urutan elemen tersebut ada dua yaitu *forward selection*

dan *backward selection*. *Forward selection* bekerja dengan mengambil *natural sequence* dari awal *array* sampai ke akhir *array*. *Backward selection* bekerja hampir sama dengan *forward selection*, hanya saja *natural sequence* yang diambil dimulai dari akhir *array* menuju ke awal *array*. *Backward selection* akan selalu dilakukan setelah *forward selection* selesai dilakukan.

Proses *forward selection* secara *detail* adalah sebagai berikut [5]:

1. Hal yang dilakukan pada proses *forward selection* untuk pertama kali adalah memilih elemen yang terdapat pada *index* 0 (awal dari *array* yang bermasalah) dan memindahkannya ke tempat penampungan nilai *temporary* berupa *current_highest* beserta *array* baru (*array forward*) yang akan menampung semua nilai yang disimpan nantinya oleh pengecekan *forward selection*. Elemen yang dipilih tersebut akan menjadi elemen yang terbesar (*current highest*).
2. Setelah mendapatkan *current_highest* untuk pertama kalinya maka *index* akan bertambah satu dan melakukan proses banding. Nilai elemen *integer* yang sedang ada pada *current_highest* akan dibandingkan dengan yang ada pada *index* yang sedang dibandingkan. Jika elemen *current_highest* lebih besar dari elemen yang ada pada *index* maka elemen tidak akan dihiraukan dan dilanjutkan ke langkah ke tiga. Jika elemen *current_highest* sama dengan ataupun lebih kecil dari elemen yang ada pada *index* maka elemen akan dipindahkan menggantikan elemen yang ada di *current_highest* dan juga dimasukkan ke dalam *array forward*.
3. *Index* akan berlanjut bertambah satu dan dilakukan langkah kedua kembali. Hal ini dilakukan berulang kali seterusnya sampai semua elemen yang ada pada *array* yang bermasalah telah ditelusuri.
4. Setelah semua elemen yang ada di dalam *array* ditelusuri maka didapatkan sebuah *array* baru berupa *array forward* dan *array* sisa yang berisikan elemen yang tidak dipindahkan ke *array forward* tadinya.

Proses *backward selection* secara *detail* adalah sebagai berikut [5]:

1. Hal yang dilakukan pada proses *backward selection* untuk pertama kali adalah memilih elemen yang terdapat pada *index* terakhir dari *array* bermasalah yang telah selesai diproses oleh *forward selection* dan memindahkannya ke tempat penampungan nilai *temporary* berupa *current_highest* beserta *array* baru (*array backward*) yang akan menampung semua nilai yang disimpan nantinya oleh pengecekan *backward selection*. Elemen yang dipilih tersebut akan menjadi elemen yang terbesar (*current highest*).
2. Setelah mendapatkan *current_highest* untuk pertama kalinya maka *index* akan berkurang satu dan melakukan proses banding. *Index* berkurang dikarenakan proses yang dilakukan berupa *backward* bukan *forward* lagi. Nilai elemen *integer* yang sedang ada pada *current_highest* akan dibandingkan dengan yang ada pada *index* yang sedang dibandingkan. Jika elemen *current_highest* lebih besar dari elemen yang ada pada *index* maka elemen tidak akan dihiraukan dan dilanjutkan ke langkah ke tiga. Jika elemen *current_highest* sama dengan ataupun lebih kecil dari elemen yang ada pada *index* maka elemen akan dipindahkan menggantikan elemen yang ada di *current_highest* dan juga dimasukkan ke dalam *array backward*.
3. *Index* akan berlanjut berkurang satu dan dilakukan langkah ke-2 kembali. Hal ini dilakukan berulang kali seterusnya sampai semua elemen yang ada pada *array* yang bermasalah telah ditelusuri.
4. Setelah semua elemen yang ada di dalam *array* ditelusuri maka didapatkan sebuah *array* kedua yang baru berupa *array backward* dan *array* sisa yang berisikan elemen yang tidak diambil tadinya.

Setelah *forward* dan *backward selection* selesai dilakukan maka didapatkan tiga *array* yang berbeda satu sama lain yaitu *array forward*, *array backward* dan *array* sisa. *Array forward* dan *backward* merupakan *array* yang dihasilkan dari teknik *forward* dan *backward selection* pada *array* yang bermasalah. Kedua *array* tersebut nantinya akan digabung dengan teknik *merging* menjadi sebuah *array* hasil pertama. *Merging* sendiri merupakan salah satu tahapan algoritma *super sort* yang akan selalu dipakai untuk menggabungkan dua *array* yang dihasilkan oleh *forward* dan *backward selection*.

Pada bagian *merging*, akan ada sebuah fungsi *merge* yang akan menerima masukan berupa dua buah *list* (*array* yang telah terurut) yaitu *forward* dan *backward array*. Terdapat lima variabel yang digunakan di dalam fungsi tersebut yaitu *merging_list* sebagai *list* baru yang kosong, m sebagai panjang dari *list M* (*forward array*), n sebagai panjang dari *list N* (*backward array*), serta i dan j sebagai banyak elemen yang sudah berada di dalam *merging_list*. Perulangan akan terus dilakukan selama jumlah i dan j masih lebih kecil dari jumlah m dan n, dimana kondisi perulangan ini menandakan apakah banyak elemen pada *merging_list* sudah sama dengan jumlah banyak elemen pada *list M* dan *list N*. Di dalam perulangan terdapat beberapa kondisi *if-else* yang digunakan untuk memasukan elemen ke dalam *merging_list* yaitu:

1. Kondisi 1
   Jika i sama dengan m maka dapat diartikan bahwa semua elemen pada *list M* sudah dimasukkan ke dalam *merging_list*. Hal tersebut menandakan bahwa elemen *list M* telah habis dan akan dimasukkan elemen ke-j dari *list N*, kemudian j ditambah 1.
2. Kondisi 2
   Jika j sama dengan n maka dapat diartikan bahwa semua elemen pada *list N* sudah dimasukkan ke dalam *merging_list*. Hal tersebut menandakan bahwa elemen *list N* telah habis dan akan dimasukkan elemen ke-i dari *list M*, kemudian i ditambah 1.
3. Kondisi 3
   Jika nilai elemen ke-i pada *list M* lebih kecil sama dengan dari elemen ke-j pada *list N* maka *merging_list* akan dimasukkan elemen ke-i pada *list M*, kemudian i ditambah 1.
4. Kondisi 4
   Jika nilai elemen ke-i pada *list M* lebih besar dari elemen ke-j pada *list N* maka *merging_list* akan dimasukkan elemen ke-j pada *list N*, kemudian j ditambah 1. Setelah perulangan berhenti, nilai dari *merging_list* akan dikembalikan.

Array sisa yang merupakan *array* ketiga dari kumpulan *array* yang dihasilkan dari *forward* dan *backward selection* berupa kumpulan sisa elemen yang tidak diambil ketika kedua tahap *selection* dijalankan. *Array* sisa tersebut akan dibagi menjadi dua dari tengah *array* dengan teknik *partition*. Teknik *partition* merupakan tahapan yang akan selalu dilakukan setelah *array* sisa didapatkan. Setelah dibagi maka akan didapatkan dua *array* yang berupa *left sublist* dan *right sublist*. Teknik *forward* dan *backward selection* akan dilakukan lagi pada *left* dan *right sublist*, hal ini akan terus dilakukan (rekursif) sampai sisa elemen yang ada pada *array* lebih kecil dari satu. Kondisi dimana sisa elemen di dalam *array* lebih kecil dari satu merupakan ketentuan *boundary condition*. Setiap *array left* dan *right sublist* yang tadinya dihasilkan secara rekursif akan di *merge* satu sama lain lagi dengan teknik *merging* menjadi sebuah *array* hasil kedua. *Array* hasil kedua nantinya akan di *merge* kembali dengan *array* hasil pertama dan menghasilkan sebuah *array* hasil akhir yang telah diurutkan.

2.1.4 Black Box Testing

Black box testing adalah teknik pengujian yang mengabaikan rincian struktur internal suatu sistem dan hanya berfokus kepada masukan yang diterima, keluaran yang dihasilkan dan kondisi eksekusi sebuah sistem [14]. Proses ini memungkinkan penguji untuk mengidentifikasi keluaran yang mungkin salah atau tidak sesuai dengan ekspektasi. Metode ini berfokus kepada keperluan fungsional dari sebuah aplikasi. Oleh karena itu tipe *testing* ini memungkinkan para pengembang untuk membuat kondisi *input* yang akan memeriksa seluruh syarat-syarat fungsional sebuah sistem. Beberapa kesalahan yang dapat dilihat dengan menggunakan *black box testing* adalah kesalahan bagian antarmuka, kinerja, *functions*, akses *database* dan perilaku pada aplikasi.

Pada *black box testing* sendiri terdapat banyak teknik yang dapat digunakan untuk memeriksa kesalahan aplikasi. Contoh teknik-teknik tersebut adalah *equivalence partitioning*, *graph-based testing*, *state transition testing*, *fuzz testing*, *all-pairs testing*, *boundary value analysis*, *error guessing* dan seterusnya. Teknik-teknik tersebut dapat digunakan untuk mendapatkan *test case* yang akan digunakan pada proses *black box testing* terhadap sistem. *Test case* berupa daftar langkah yang akan dilakukan untuk penguji ketika melakukan *black box testing*. Pada bagian pengujian akan digunakan tiga teknik *black box testing* yaitu *equivalence partitioning*, *boundary value analysis* dan *error guessing*.

2.1.4.1 Equivalence Partioning

Equivalence partioning merupakan teknik *black box testing* yang membagi satu set kondisi pengujian (*domain*) menjadi kelompok (*subset*) atau biasa disebut *equivalence classes* [15]. Kelompok atau bisa dikatakan *test case* yang dibagi dari kondisi pengujian biasanya memuat nilai yang hampir sama, oleh karena itu pada dasarnya apa yang ada pada satu kelompok hanya akan menghasilkan hasil *valid* atau *invalid*. Karena *value* pada setiap kelompok di teknik ini akan dianggap sama maka pengujian pada setiap kelompok hanya akan menggunakan salah satu nilai yang ada di dalamnya (hanya dilakukan sekali), hal ini akan membuat *test case* yang digunakan menjadi lebih sedikit.

2.1.4.2 Boundary Value Anaylsis

Boundary value analysis adalah teknik pengujian didasarkan pada batas awal dan akhir antar kelompok (*subset*) yang telah dipisahkan. Ide dari metode *boundary value analysis* sendiri ditemukan dari banyaknya kesalahan yang sering ditemukan dari pengujian nilai batas sebuah kelompok [15]. Pada *boundary value analysis* variabel yang akan dikelompokkan harus berurutan, hal ini dikarenakan pada teknik ini variabel harus bisa dibandingkan mana yang lebih besar dan mana yang lebih kecil.

Langkah awal dari cara kerja *Boundary value analysis* hampir sama dengan *equivalence partioning* yaitu melakukan pembagian variabel ke dalam kelompok menjadi *valid* dan *invalid*. Langkah selanjutnya berupa mengambil nilai variabel yang akan digunakan dari batas awal dan akhir dari kelompok yang telah dibagi sebelumnya. Variabel yang diambil berupa variabel sebelum dan sesudah dari kedua batas tersebut. Contohnya, jika ada *textbox* yang hanya bisa menerima angka *integer* tiga sampai dengan sepuluh maka *test case* yang dapat digunakan dengan *boundary value analysis* berupa angka $2, 3, 4, 9, 10, 11$. Angka $2, 3$ dan 4 berupa variabel yang didapatkan dari batas awal sedangkan $9, 10$ dan 11 merupakan variabel yang didapatkan dari batas akhir.

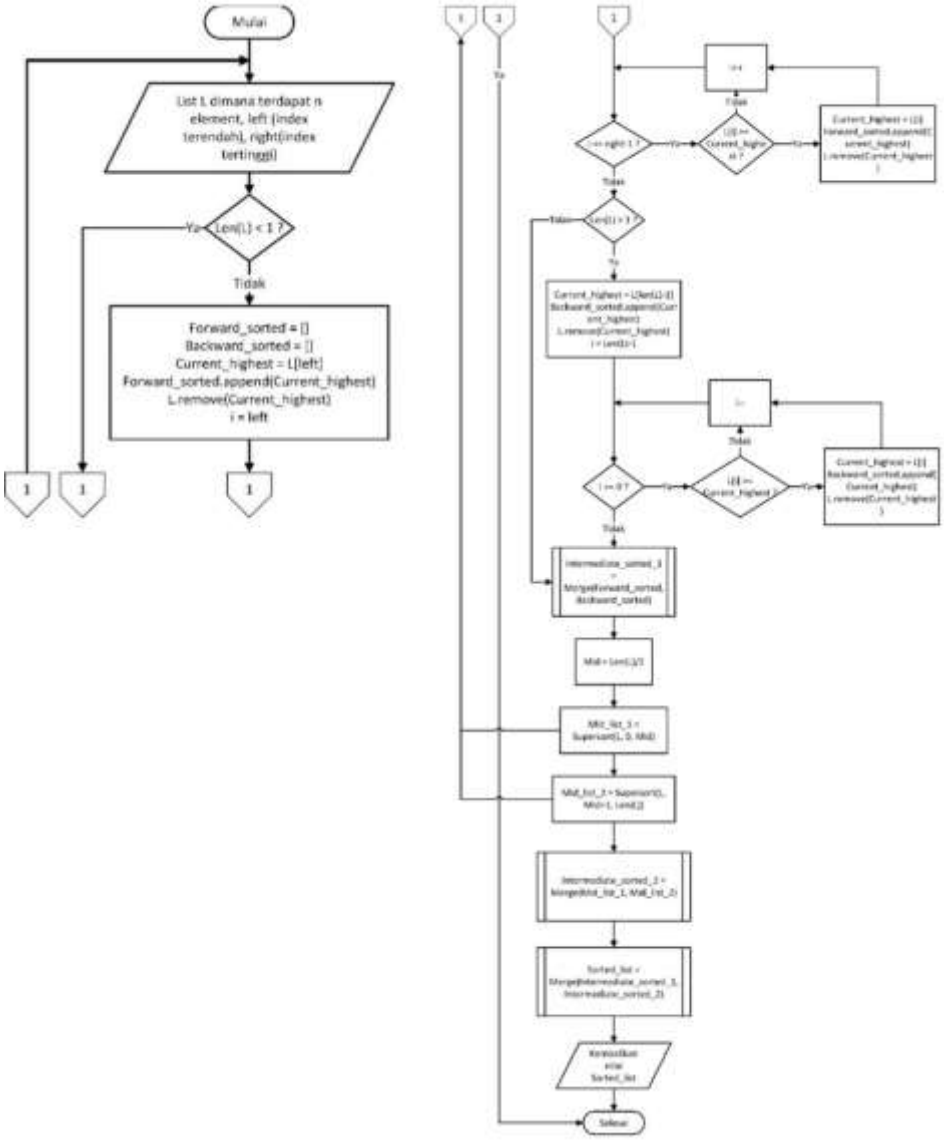
2.1.4.3 Error Guessing

Error guessing merupakan pengujian yang dilakukan dengan membuat daftar kemungkinan kesalahan yang terjadi pada aplikasi dan kemudian pengujian akan dilakukan dengan mengikuti alur pengujian sesuai dengan daftar yang telah dibuat [16]. Pada teknik ini, setiap penguji akan mendapatkan *value* yang akan dimasukkan kedalam daftar kemungkinan kesalahan berdasarkan pemahaman dan asumsi mereka terhadap aplikasi. Teknik ini bekerja sesuai dengan kemampuan sang penguji dimana semakin baik seorang penguji maka semakin baiklah pengujian berjalan.

Teknik *error guessing* sebenarnya lebih tepat digunakan setelah teknik lainnya yang lebih formal telah digunakan. Hal ini dikarenakan ketika penguji menggunakan teknik yang lebih formal, penguji akan lebih mengerti cara kerja dari sistem yang telah dirancangnya [17]. Dengan pemahaman yang telah didapatkan maka penguji dapat lebih bisa memperhitungkan kondisi yang memungkinkan terjadinya kesalahan di dalam sistem.

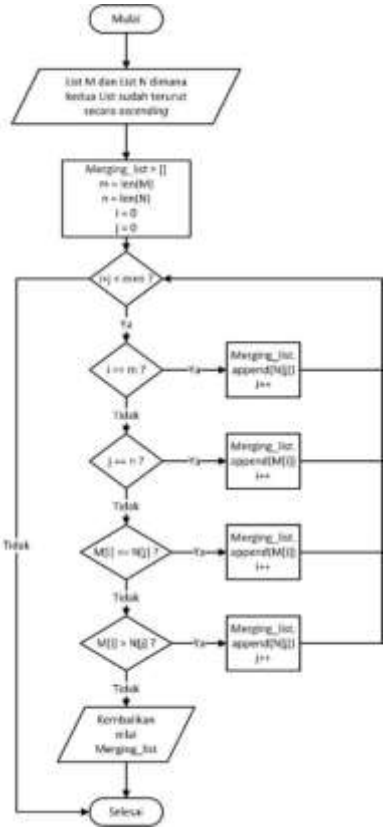
2.2 Analisis Proses

Analisis proses merupakan tahap menganalisis langkah algoritma bekerja ketika pengguna memasukkan sekumpulan angka yang akan disimulasikan proses pengurutannya. *List* angka yang dimasukkan akan diproses dengan algoritma *super sort* yang dilakukan dari tahap awal (sekumpulan angka yang masih acak) sampai akhir (sekumpulan angka yang telah terurut). Tahapan kerja dari algoritma *super sort* pada sistem terdiri atas empat proses utama yang bekerja dengan rekursif. Keempat proses tersebut berupa *forward selection*, *backward selection*, *merge* dan *partition*. Proses kerja *super sort* dapat digambarkan dalam bentuk *flowchart* seperti yang ada pada Gambar 1.



Gambar 1 *Flowchart* algoritma *super sort*

Proses kerja dari algoritma *super sort* ini memerlukan sebuah fungsi berupa *merge* yang digunakan untuk menggabungkan *list* yang didapatkan dari proses *forward* dan *backward selection* beserta semua list hasil perulangan secara rekursif *super sort* yang didapatkan. Proses kerja dari fungsi *merge* dapat digambarkan dalam bentuk *flowchart* seperti yang ada pada Gambar 2.



Gambar 2 *Flowchart* algoritma fungsi *merge*

3. HASIL DAN PEMBAHASAN

3.1 Hasil

Pengujian yang dilakukan pada aplikasi hasil berupa *black box testing* untuk mengetahui kecocokan *output* dari setiap simulasi tahapan *super sort* yang disediakan beserta pengecekan input yang dapat dimasukkan pada simulasi yang disediakan pada aplikasi pembelajaran.

3.1.1 Pengujian Sistem dengan Black Box Testing

Pengujian yang dilakukan dengan *black box testing* meliputi 3 teknik yang digunakan yaitu *equivalence partitioning*, *boundary value analysis*, dan *error guessing*. Ketiga teknik tersebut digunakan terhadap empat simulasi yang disediakan dalam aplikasi yaitu simulasi rekursif, simulasi *forward* dan *backward selection*, simulasi *merging* dan simulasi *super sort*. Berikut data pengujian dengan ketiga teknik yang dimasukkan kedalam aplikasi:

1. Simulasi rekursif

Tabel 1 Pengujian Simulasi Rekursif dengan metode *equivalence partitioning*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan
1	Input < 0	-10	Tidak dapat dimasukkannya angka minus	Tidak ada keluaran	Sukses
2	0 < Input < 10000	10	Berjalannya simulasi rekursif dengan keluaran yang tepat	Animasi rekursif berjalan dan memberikan *output* yang tepat	Sukses

3	Input > 10000	15000	Adanya *Toast* yang menyatakan kesalahan *input*	Muncul *Toast* "Masukkan lebih dari 10000"	Sukses

Tabel 2 Pengujian Simulasi **r**ekursif dengan metode *boundary value analysis*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan
1	Input < 0 = -1	-1	Tidak dapat dimasukkannya angka minus	Tidak ada keluaran	Sukses
2	Input > 0 = 1	1	Berjalannya simulasi rekursif	Animasi rekursif berjalan dan memberikan *output* yang tepat	Sukses
3	Input > 10000 = 10001	10001	Adanya *Toast* yang menyatakan kesalahan *input*	Muncul *Toast* "Masukkan lebih dari 10000"	Sukses
4	Input < 10000 = 9999	9999	Berjalannya simulasi rekursif dengan keluaran yang tepat	Pada faktor (17) telah menghasilkan angka minus karena *int overflow*	Gagal

Tabel 3 Pengujian Simulasi **r**ekursif dengan metode *error guessing*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan
1	Memasukkan sebuah simbol	!	Tidak dapat dimasukkannya huruf ataupun simbol	Tidak ada keluaran	Sukses

2. Simulasi *forward* dan *backward selection*

Tabel 4 Pengujian Simulasi *selection* dengan metode *equivalence partitioning*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan
1	*List Input* kosong	-	Adanya *Toast* yang menyatakan kesalahan *input*	*Toast* yang menuliskan "Masukkan kosong" muncul	Sukses
2	*List Input* lebih dari satu	[7, 5, 3, 4, 13, 24, 7, 8]	Berjalannya simulasi beserta keluarnya hasil simulasi	Animasi grafik proses *selection* berjalan dan memberikan *output* yang tepat	Sukses

Tabel 5 Pengujian Simulasi *selection* dengan metode *boundary value analysis*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan
1	*List Input* satu buah elemen	[1]	Berjalannya simulasi *selection* beserta keluarnya hasil simulasi	Animasi grafik proses *selection* berjalan dan memberikan *output* yang tepat	Sukses

Tabel 6 Pengujian Simulasi *selection* dengan metode *error guessing*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan

1	*Input List* dengan elemen yang sama	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	Berjalannya simulasi beserta keluarnya hasil simulasi	Animasi grafik proses *forward* dan *backward selection* berjalan dan memberikan *output* yang tepat	Sukses
2	*Input List* dengan elemen yang terurut	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	Berjalannya simulasi beserta keluarnya hasil simulasi	Animasi grafik proses *forward* dan *backward selection* berjalan dan memberikan *output* yang tepat	Sukses
3	*Input List* dengan elemen terurut yang terbalik	[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	Berjalannya simulasi beserta keluarnya hasil simulasi	Animasi grafik proses *forward* dan *backward selection* berjalan dan memberikan *output* yang tepat	Sukses
4	*Input List* dengan beberapa elemen yang sama	[20, 21, 20, 20, 1, 3]	Berjalannya simulasi beserta keluarnya hasil simulasi	Animasi grafik proses *forward* dan *backward selection* berjalan dan memberikan *output* yang tepat	Sukses
5	*Input List* dengan elemen yang banyak	[1, 8, 9, 81, 23, 45, 20, 11, 77, 19, 20, 31, 41, 5, 78, 9, 20, 23, 25, 27, 28, 29, 90, 100, 70, 12, 34, 56, 1, 44]	Berjalannya simulasi beserta keluarnya hasil simulasi	Animasi grafik proses *forward* dan *backward selection* berjalan dan memberikan *output* yang tepat	Sukses
6	*Input List* dengan elemen yang besar	[10000, 9998, 3455, 7891, 2345, 2, 3456, 1]	Berjalannya simulasi beserta keluarnya hasil simulasi	Animasi grafik proses *forward* dan *backward selection* berjalan dan memberikan *output* yang tepat	Sukses
7	*Input List* dengen elemen string	[a]	Tidak dapat dimasukkannya huruf ataupun simbol	Tidak ada keluaran	Sukses

3. Simulasi *merging*

Tabel 7 Pengujian Simulasi *merging* dengan metode *equivalence partitioning*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan
1	*List Input* kosong	List 1 = [-] List 2 = [-]	Adanya *Toast* yang menyatakan kesalahan *input*	*Toast* yang menuliskan “Masukkan kosong” muncul	Sukses
2	*List Input* lebih dari satu	List 1 = [1, 5, 7, 9]	Berjalannya simulasi *merging* beserta	Animasi keluaran proses *merging* antara dua *list* berjalan dan	Sukses

		List 2 = [1, 6, 8, 9]	keluarnya hasil simulasi	memberikan *output* yang tepat	

Tabel 8 Pengujian Simulasi *merging* dengan metode *boundary value analysis*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan
1	*List Input* satu buah elemen	List 1 = [20] List 2 = [8791]	Berjalannya simulasi *merging* beserta keluarnya hasil simulasi	Animasi keluaran proses *merging* antara dua *list* berjalan dan memberikan output yang tepat	Sukses

Tabel 9 Pengujian Simulasi *merging* dengan metode *error guessing*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan
1	*Input List* dengan elemen yang sama	List 1 = [1, 1, 1, 1] List 2 = [1, 1, 1, 1]	Berjalannya simulasi *merging* beserta keluarnya hasil simulasi	Animasi keluaran proses *merging* antara dua *list* berjalan dan memberikan *output* yang tepat	Sukses
2	*Input List* dengan beberapa elemen yang sama	List 1 = [1, 2, 2, 3] List 2 = [5, 6, 6, 9]	Berjalannya simulasi *merging* beserta keluarnya hasil simulasi	Animasi keluaran proses *merging* antara dua *list* berjalan dan memberikan *output* yang tepat	Sukses
3	*Input List* dengan elemen yang banyak	List 1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30] List 2 = [1, 1, 5, 8, 9, 9, 11, 12, 19, 20, 20, 20, 23, 23, 25, 27, 28, 29, 31, 34, 41, 44, 45, 56, 70, 77, 78, 81, 90, 100]	Berjalannya simulasi *merging* beserta keluarnya hasil simulasi	Animasi keluaran proses *merging* antara dua *list* berjalan dan memberikan *output* yang tepat	Sukses
4	*Input List* dengan elemen yang besar	List 1 = [1999, 2213, 2345, 9321] List 2 = [1222, 1346, 6000, 9991]	Berjalannya simulasi *merging* beserta keluarnya hasil simulasi	Animasi keluaran proses *merging* antara dua *list* berjalan dan memberikan *output* yang tepat	Sukses
5	*Input List* dengen elemen string	List 1 = [a] List 2 = [a]	Tidak dapat dimasukkannya huruf ataupun simbol	Animasi keluaran proses *merging* antara dua *list* berjalan dan memberikan *output* yang tepat	Sukses

4. Simulasi *super sort*

Tabel 10 Pengujian Simulasi *super sort* dengan metode *equivalence partitioning*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan
1	*List Input* kosong	-	Adanya *Toast* yang menyatakan kesalahan *input*	*Toast* yang menuliskan "Masukkan kosong" muncul	Sukses
2	*List Input* lebih dari satu	[7, 5, 3, 4, 13, 24, 7, 8]	Berjalannya simulasi *super sort* beserta keluarnya hasil simulasi	Animasi keluaran proses pengurutan *super sort* berjalan dan memberikan *output* yang tepat	Sukses

Tabel 11 Pengujian Simulasi *super sort* dengan metode *boundary value analysis*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan
1	*List Input* satu buah elemen	[1]	Berjalannya simulasi *super sort* beserta keluarnya hasil simulasi	Animasi keluaran proses pengurutan *super sort* berjalan dan memberikan *output* yang tepat	Sukses

Tabel 12 Pengujian Simulasi *super sort* dengan metode *error guessing*

No	*Test Case*	*Input*	Hasil yang diharapkan	*Output*	Kesimpulan
1	*Input List* dengan elemen yang sama	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	Berjalannya simulasi *super sort* beserta keluarnya hasil simulasi	Animasi keluaran proses pengurutan *super sort* berjalan dan memberikan *output* yang tepat	Sukses
2	*Input List* dengan elemen yang terurut	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	Berjalannya simulasi *super sort* beserta keluarnya hasil simulasi	Animasi keluaran proses pengurutan *super sort* berjalan dan memberikan *output* yang tepat	Sukses
3	*Input List* dengan elemen terurut yang terbalik	[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]	Berjalannya simulasi *super sort* beserta keluarnya hasil simulasi	Animasi keluaran proses pengurutan *super sort* berjalan dan memberikan *output* yang tepat	Sukses
4	*Input List* dengan beberapa elemen yang sama	[20, 21, 20, 20, 1, 3]	Berjalannya simulasi *super sort* beserta keluarnya hasil simulasi	Animasi keluaran proses pengurutan *super sort* berjalan dan memberikan *output* yang tepat	Sukses
5	*Input List* dengan elemen	[1, 8, 9, 81, 23, 45, 20, 11, 77, 19, 20,	Berjalannya simulasi *super sort* beserta	Animasi keluaran proses pengurutan *super sort* berjalan dan	Sukses

	yang banyak	31, 41, 5, 78, 9, 20, 23, 25, 27, 28, 29, 90, 100, 70, 12, 34, 56, 1, 44]	keluarnya hasil simulasi	memberikan *output* yang tepat	
6	*Input List* dengan elemen yang besar	[10000, 9998, 3455, 7891, 2345, 2, 3456, 1]	Berjalannya simulasi *super sort* beserta keluarnya hasil simulasi	Animasi keluaran proses pengurutan *super sort* berjalan dan memberikan *output* yang tepat	Sukses
7	*Input List* dengen elemen string	[a]	Tidak dapat dimasukkannya huruf ataupun simbol	Tidak ada keluaran	Sukses

3.2 Pembahasan

Berdasarkan hasil pengujian yang dilakukan, maka dapat ditarik informasi bahwa fitur rekursif memiliki satu kegagalan dalam pengujiannya yaitu hanya dapat menampilkan hasil faktorial sampai dengan masukkan angka enam belas. Hal ini dikarenakan *integer overflow* yang terjadi ketika faktorial ketujuh belas dipanggil. Oleh sebab itu fitur simulasi rekursif mempunyai persentase keberhasilan berupa 66,7% karena gagal dalam satu dari tiga metode *black box testing* yang diuji. Fitur simulasi *selection*, *merging* dan *super sort* telah melewati ketiga metode *black box testing* dengan persentase yang sempurna yaitu 100 %. Berdasarkan semua persentase tersebut didapatkan bahwa fitur simulasi pada aplikasi pembelajaran yang dibuat telah melewati *black box testing* karena mempuyai persentase 91.7 % yang didapatkan dari hasil penambahan setiap persentase dan dibagi dengan jumlah simulasi yang ada.

4. KESIMPULAN DAN SARAN

4.1 Kesimpulan

Berdasarkan hasil yang didapatkan, maka dapat diperoleh kesimpulan sebagai berikut :

1. Berdasarkan hasil persentase pengujian *black box testing* yang berupa 91,7 %, dapat disimpulkan bahwa fitur simulasi pada aplikasi pembelajaran yang dibuat telah mampu digunakan oleh pengguna.
2. Simulasi rekursif yang disediakan mampu menunjukkan hasil sampai dengan $faktorial$ (16). Hal ini dikarenakan terjadinya *integer overflow* ketika $faktorial$ (17) dipanggil oleh sistem.

4.2 Saran

Adapun saran - saran yang diberikan sebagai bahan pertimbangan untuk mengembangkan penelitian ini.

1. Bagi peneliti lain yang ingin melakukan penelitian mengenai algoritma penyortiran, maka dapat menerapkan algoritma *super sort* untuk menyelesaikan kasus penelitian lainnya.
2. Peneliti lain dapat menambahkan algoritma penyortiran yang baru sebagai bahan perbandingan terhadap algoritma *super sort* dengan tujuan memberikan informasi atau pengetahuan yang baru.
3. Peneliti lain dapat menambahkan fitur pembelajaran yang baru untuk menyelesesaikan masalah simulasi rekursif. Hal tersebut dapat berupa perubahan cara mengajarkan yang berdasarkan simulasi ataupun mengatasi masalah *integer overflow* tersebut.
4. Peneliti lain dapat menemukan cara pembelajaran pada aplikasi *mobile* yang lebih efektif dan mengimplementasikannya pada pembelajaran algoritma penyortiran.

DAFTAR PUSTAKA

[1] Yahaya, N. S. & Salam, S. N. A., 2014. Mobile Learning Application for Children: Belajar Bersama Dino. *Procedia - Social and Behavioral Sciences*, Juni, Volume 155, hal. 398-404.

[2] Taurusta, C. & Findawati, Y., 2017. Rancang Bangun Game Algoritma dan Struktur Data Berbasis Role Playing Game (RPG) Sebagai Media Pembelajaran Mahasiswa Teknik Informatika Universitas Muhammadiyah Sidoarjo. *KINETIK*, Agustus, 2(3), hal. 175-188.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C., 2009. *Introduction to Algorithms.* 3rd ed. Massachusetts: Massachusetts Institute of Technology.

[4] Jabar, A. A. & Anas, A. S., 2019. Aplikasi Belajar Interaktif Algoritma Sorting Berbasis Desktop. *Jurnal Teknologi Informasi dan Multimedia*, Mei, 1(1), hal. 23-29.

[5] Gugale, Y., 2018. Super Sort Sorting Algorithm, *2018 3rd International Conference for Convergence in Technology (I2CT)*, Pune, April 6-8.

[6] Pereira, S. D. et al., 2021. Advances in Intelligent Systems and Computing volume 1177, Springer, Singapore.

[7] Sedgewick, R., 1978. Implementing Quicksort programs. *Communications of the ACM*, 21(10), hal. 847-857.

[8] Kalvslund, B., 2018. *Bubble Sort - play with the algorithm (Version 1.0).* [Online] Available at: https://play.google.com/store/apps/details?id=com.bkalvslund.Bubblesort&hl=in [Accessed 7 Agustus 2020].

[9] Hidayat, A. & Utomo, V. G., 2014. Implementing Code Igniter Framework in Open Source Mobile Learning Application. *International Journal of Computer Applications*, 1 1, 108(18), pp. 9-14.

[10] Hashim, A. S., Ahmad, W. F. W. & Rohiza, A., 2010. *A study of design principles and requirements for the m-learning application development.* Shah Alam, IEEE, pp. 226-231.

[11] Qian, X. j. & Xu, J. b., 2011. *Optimization and implementation of sorting algorithm based on multi-core and multi-thread.* Xi'an, IEEE.

[12] Kusmira, M., Mulyani, Y. S. & K., 2015. *Komparasi Algoritma Quicksort dan Bucket Sort Pengurutan Data Integer Menggunakan Bahasa C++*. Tasikmalaya, Konferensi Nasional Ilmu Sosial & Teknologi (KNiST), pp. 139-144.

[13] Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C., 2009. *Introduction to Algorithms.* 3rd ed. Massachusetts: Massachusetts Institute of Technology.

[14] Naik, K. & Tripathy, P., 2008. *Software Testing and Quality Assurance: Theory and Practice.* 1st ed. Hoboken: John Wiley & Sons.

[15] Roman, A., 2018. *A Study Guide to the ISTQB® Foundation Level 2018 Syllabus.* 1st ed. Cham: Springer.

[16] Wamiliana, Kurniasari, D. & Rokhman, F., 2013. Implementasi Metode Dynamic Programming. *Jurnal Komputasi*, 1(1), pp. 10-14.

[17] Graham, D., Veenendaal, E. v., Evans, I. & Black, R., 2006. *Foundations of Software Testing: ISTQB Certification.* s.l.:Thomson.