BAB II

TINJAUAN PUSTAKA

2.1. *Bin Packing Problem*

Menurut Korte dan Vygen (2006), dalam permasalahan pengemasan barang ke dalam *bin*, barang-barang dengan *volume* yang berbeda harus dikemas ke dalam *bin* atau kontainer dengan jumlah yang terbatas dimana harus meminimalkan jumlah *bin* yang digunakan. Permasalahan ini dianggap sebagai permasalahan *combinatorial* NP-*hard*.

Terdapat banyak variasi dari masalah ini, seperti pengemasan 2D, pengemasan linear, pengemasan berdasarkan berat, pengemasan berdasarkan biaya, dan sebagainya. Terdapat berbagai aplikasi, seperti pengisian kontainer, memuat truk dengan kapasitas berat yang terbatas, menciptakan *file backups*, dan lain-lain.

Terlepas dari kenyataan bahwa masalah pengemasan ke dalam *bin* memiliki kompleksitas yang bertingkat *computational* NP-*hard*, solusi optimal dapat dicapai dengan algoritma yang baik. Selain itu, banyak algoritma heuristik yang sudah dikembangkan.

2.2. 3D *Bin Packing Problem*

Menurut Martello *et al.* (2007), 3D *bin packing problem* meminta solusi untuk kemasan orthogonal dari sekumpulan barang berbentuk persegi panjang agar dapat dikemas ke dalam jumlah wadah minimal

berbentuk persegi panjang. 3D-BPP (3D *bin packing problem*) sangat terkait dengan masalah pengemasan kontainer 3D, antara lain:

a. *Knapsack Loading*. Dalam *knapsack loading* sebuah kontainer, setiap barang memiliki keuntungan, dan masalahnya adalah bagaimana memilih sekumpulan barang yang cocok ke dalam satu kontainer sehingga memperoleh keuntungan maksimal. Jika keuntungan dari setiap barang adalah *volume*-nya, maka untuk dapat mencapai keuntungan maksimal adalah dengan meminimalisir *volume* yang tidak terpakai.
b. *Container Loading*. Dalam permasalahan ini, semua barang harus dapat dikemas ke dalam sebuah wadah atau kontainer tunggal yang tingginya tidak terbatas. Masalahnya adalah bagaimana menemukan solusi yang layak sehingga tinggi dari kontainer yang diisi diminimalkan.
c. *Bin packing*. 3D-BPP membutuhkan solusi dimana semua barang dapat dikemas ke dalam kontainer yang memiliki ukuran yang terbatas, dan tujuannya adalah untuk menemukan solusi dengan menggunakan jumlah kontainer paling sedikit.

2.3. *First Fit*

Menurut Dósa and Sgall (2013), Algoritma *first fit* akan mengemas semua barang ke dalam wadah pertama dimana terdapat ruang untuk barang tersebut, kemungkinan akan membuat sebuah wadah baru jika

barang tidak dapat dikemas ke dalam ruang yang ada pada wadah yang sudah ada. Wadah yang digunakan dalam *first fit* diurutkan berdasarkan waktu dimana mereka dibuat, sehingga barang pertama akan dikemas ke dalam wadah pertama. Cara kerja dari algoritma *first fit* adalah sebagai berikut:

1. *Input* berupa sejumlah barang dengan ukuran tertentu.
2. *Input* berupa ukuran wadah yang akan digunakan dalam mengemas barang.
3. Eksekusi algoritma akan dilakukan dengan cara memasukkan barang berdasarkan urutan mulai dari yang pertama hingga ke-*n* pada wadah yang sudah dibuat.
4. Jika wadah yang sudah dibuat tidak memiliki ruang yang cukup untuk mengemas barang, algoritma ini akan membuat wadah baru dengan ukuran yang sama yang sudah ditetapkan pada *input*.
5. Sisa barang yang masih ada di luar akan dikemas dengan cara memulai dari wadah pertama hingga wadah ke-*i* yang memiliki ruang untuk mengemas barang dan akan dilanjutkan dengan barang berikutnya.
6. Pengulangan akan terjadi hingga semua barang yang ditetapkan sudah dikemas ke dalam wadah.

2.4. *First Fit Decreasing*

Menurut Dósa (2007), *first fit decreasing* adalah sebuah algoritma pengemasan barang yang klasik 2 dimensi dimana barang akan diurutkan secara *nonincreasing*, dan selanjutnya dimasukkan ke dalam wadah pertama dimana barang tersebut muat.

Menurut Guo (diakses tanggal 1 Desember 2016), *first fit decreasing* akan mengurutkan barang dalam *non-increasing order* dengan memperhatikan ukuran dari barang tersebut. Dengan kata lain, *first fit decreasing* akan mengurutkan barang dengan ukuran terbesar hingga terkecil. Setelah mengurutkan barang, maka pengemasan akan dilanjutkan sama dengan metode *first fit*.

2.5. *Largest Area First-Fit*

Algoritma ini pertama kali diperkenalkan oleh M. Zahid Gürbüz *et al.* (2009) pada penelitiannya yang berjudul *An Efficient Algorithm for 3D Rectangular Box Packing*. Berikut ini akan dijelaskan mengenai *input* dan *output* dari algoritma *Largest Area First-Fit* (LAFF). Algoritma ini akan meletakkan barang dengan luas permukaan terbesar dengan meminimalkan tinggi dari dasar kontainer.

2.5.1. *Input* untuk Algoritma LAFF

Input pertama adalah sejumlah *box* dengan ukuran yang berbeda-beda dilambangkan dengan N. *Input* berikutnya adalah dimensi dari setiap macam *box*. *Input parameter* ini akan

dinotasikan dengan empat nilai (an, bn, cn, kn) dimana an adalah lebar, bn adalah kedalaman, cn adalah tinggi, dan kn adalah jumlah *box*.

Berdasarkan bagaimana anggapan melihat sebuah *box*, setiap dimensi dapat dianggap sebagai lebar, kedalaman atau tinggi. Jika menganggap bn adalah lebar, makan an dan cn dapat dianggap sebagai tinggi atau kedalaman juga, karena *box* yang digunakan berbentuk tiga dimensi dan dapat dirotasi dan dilihat dari berbagai perspektif. Berikut ini adalah daftar *input parameter* untuk Algoritma LAFF:

Tabel 2.1 *Input Parameter* LAFF

Nama	**Variabel**
Jumlah *box* dengan ukuran yang berbeda-beda	N
Lebar dari *box* ke-n	an
Kedalaman dari *box* ke-n	bn
Tinggi dari *box* ke-n	cn
Box ke-n	kn

2.5.2. *Ouput* dari Algoritma LAFF

Setelah Algoritma LAFF dieksekusi, hasilnya adalah sebagai berikut:

O1: *Volume* dari kontainer

O2: *Volume* yang terpakai (jumlah *volume* dari semua *box* yang ditempatkan)

O3: Ruang yang tidak terpakai atau terbuang

O4: Waktu yang digunakan untuk mengeksekusi

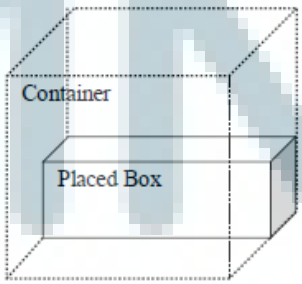
2.5.3. Cara kerja Algoritma LAFF

Seperti yang sudah dijelaskan sebelumnya, 3D *packing problem* adalah permasalahan dengan tingkat NP-*hard*. Hal ini berarti solusi optimal dari permasalahan ini dapat ditemukan dengan mencoba berbagai kombinasi dari solusi yang ada. Namun, jika jumlah *box* bertambah, jumlah iterasi atau perulangan akan bertambah pula dan tidak dapat diselesaikan dalam waktu *polynomial* meskipun menggunakan komputer dengan kecepatan komputasi dan teknologi tercepat. Dalam beberapa asumsi, permasalahan ini dapat diselesaikan dengan menggunakan algoritma pencarian heuristik yang dapat memberikan solusi yang hampir optimal.

Algoritma yang akan digunakan kali ini adalah Algoritma *Largest Area First-Fit*. Algoritma ini menggunakan pencarian heuristik yang meletakkan *box* dengan luas permukaan terbesar dengan tinggi terendah terlebih dulu ke dalam kontainer. Cara kerjanya adalah sebagai berikut:

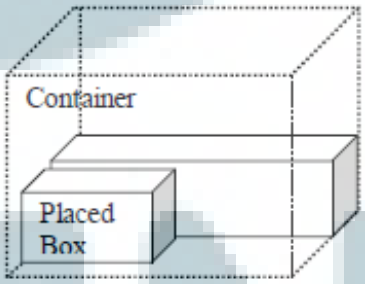
1. Pertama kali, lebar dan kedalaman dari kontainer akan ditentukan. Lebar dan kedalaman dari kontainer akan ditentukan pada awal algoritma dan akan tetap selama algoritma berjalan. Untuk tinggi dari kontainer diasumsikan tidak terbatas dan akan bertambah selama algoritma berjalan.

2. Setelah menentukan lebar dan kedalaman dari kontainer, *box* yang ada dapat mulai ditempatkan ke dalam kontainer. Dalam melakukan penempatan, terdapat dua macam algoritma. Pertama adalah metode penempatan yang akan mengalokasikan ruang untuk sebuah *box* yang akan meningkatkan tinggi dari kontainer. Kedua adalah metode penempatan yang akan mengalokasikan ruang untuk *box* yang masih di luar jika ukurannya cocok dengan ruang kosong yang ada di sekitar *box* yang sudah ada di dalam kontainer tanpa melebihi tinggi dari *box* yang sudah ada di dalam.
3. Pada metode penempatan yang pertama, *box* dengan luas permukaan terbesar ditentukan dan *box* yang terpilih ini akan dicari *box* mana yang memiliki tinggi terendah dari semua yang terpilih. Setelah itu, *box* dengan tinggi terendah akan diletakkan di dalam kontainer.



Gambar 2.1 Metode Penempatan yang Pertama

4. Pada metode penempatan yang kedua, algoritma akan mencoba mengalokasikan ruang untuk *box* yang masih ada di luar. Ruang yang dimaksud adalah ruang di sekitar *box* yang sudah ditempatkan dengan metode penempatan yang pertama. Jika ada ruang yang tersisa di sekitar *box* pertama seperti pada Gambar 2.1, algoritma akan mencoba untuk mengisi ruang tersebut dengan metode penempatan yang kedua. Jika ada *box* yang cocok untuk mengisi ruang kosong tersebut, satu atau lebih, akan ditempatkan *box* dengan *volume* terbesar seperti pada Gambar 2.2. Iterasi akan dilakukan hingga tidak ada *box* yang cocok di ruang kosong yang ada.



Gambar 2.2 Metode Penempatan yang Kedua

5. Pada metode penempatan barang yang kedua, jika tidak ada ruang kosong di sekitar *box* yang sudah ditempatkan, maka algoritma akan melanjutkan dengan metode penempatan barang yang pertama dan seterusnya.

2.6. *Prototype*

Menurut O'Brien dan Marakas (2007:374) dalam *"Enterprise Information Systems", prototype* adalah sebuah model kerja, khususnya model kerja dari suatu sistem informasi yang mencakup versi tentatif dari masukan dan keluaran pengguna, *database* dan *file*, metode pengendalian, serta pengolahan rutinitas.

Menurut Rainer dan Turban (2009:307) dalam *"Introduction to Information Systems", prototype* adalah versi kecil yang dibuat secara cepat, berdasarkan sistem yang sedang dikembangkan, dimana pengguna memberikan masukan untuk meningkatkan serta memperbaiki kinerja dari *prototype*.

Berdasarkan pengertian dari para ahli diatas, dapat disimpulkan bahwa *prototype* adalah seluruh atau sebagian dari sistem yang sedang dirancang dimana pengguna dapat berperan aktif dalam memberikan kritik dan saran terhadap sistem tersebut.

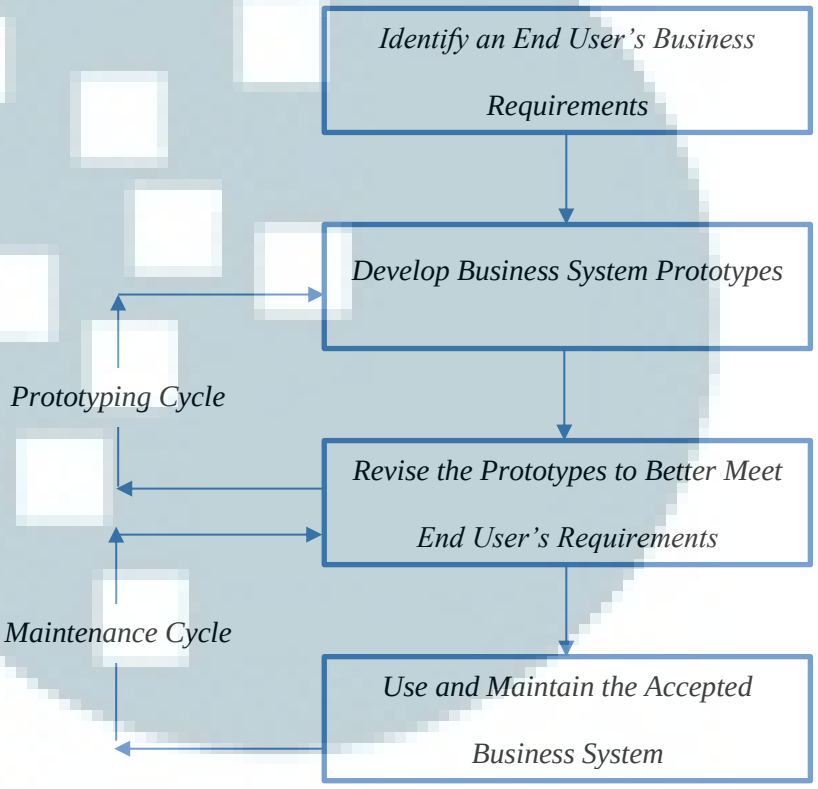
2.6.1. *Prototyping*

Menurut Mulyanto (2009), metode *prototype* atau disebut *prototyping* sangat baik digunakan untuk menyelesaikan masalah kesalahpahaman antara *user* dan analis yang timbul akibat *user* tidak mampu mendefinisikan secara jelas kebutuhannya.

Prototyping berguna untuk membuat proses pembuatan menjadi lebih mudah dan lebih cepat, khususnya utuk proyek yang

memiliki kebutuhan akhir pengguna yang sulit didefinisikan pada tahap awal.

Menurut O'Brien dan Marakas (2007:375), *prototyping* dibagi menjadi 4 tahapan, yaitu:



Gambar 2.3 *Prototyping Step*

1. *Identify an End User's Business Requirements*

   Pada tahap ini, dilakukan investigasi atau analisa kebutuhan bisnis dan melakukan penilaian terhadap kelayakan dari beberapa alternatif solusi sistem informasi.

2. *Develop Business System Prototypes*

   Pada tahap ini, pengguna akan melakukan analisa dan desain, kemudian perancang akan menggunakan alat-alat pengembangan untuk melakukan desain secara interaktif dan dapat melakukan pengetesan *prototype* untuk memenuhi kebutuhan bisnis pengguna.

3. *Revise the Prototypes to Better Meet End User's Requirement*

   Pada tahap ini, perancang akan melanjutkan desain dan memperbaiki *prototype.* Tahapan ini akan dilakukan secara berulang hingga pengguna akhir dapat menerima *prototype* tersebut.

4. *Use and Maintain the Accepted Business System*

   Pada tahap ini, dilakukan penggunaan sistem yang sudah disetujui, serta dilakukan perawatan dan pengendalian terhadap kemungkinan kegagalan sistem yang berjalan.

Menurut Walker *et al.* (2002), berdasarkan tingkat kerinciannya atau *fidelity*, *prototype* dapat dibedakan menjadi beberapa jenis, sebagai berikut:

1. *Low fidelity prototype*

   Prototype jenis ini adalah *prototype* yang tidak terlalu rinci dalam menggambarkan sistem. Karakteristiknya adalah mempunyai fungsi atau interaksi yang terbatas, lebih menggambarkan konsep perancangan dan *layout* dibandingkan dengan model interaksi, tidak memperlihatkan secara rinci operasional sistem, mendemonstrasikan secara umum *feel and look* dari antarmuka pengguna dan hanya menggambarkan konsep pendekatan secara umum.

2. *High fidelity prototype*

   Prototype jenis ini adalah *prototype* yang lebih rinci menggambarkan sistem. *Prototype* ini memiliki interaksi penuh dengan pengguna dimana pengguna dapat memasukkan data dan berinteraksi dengan sistem, mewakili fungsi-fungsi inti sehingga dapat menyimulasikan sebagian besar fungsi dari sistem akhir dan mempunyai penampilan yang sangat mirip dengan produk sebenarnya.

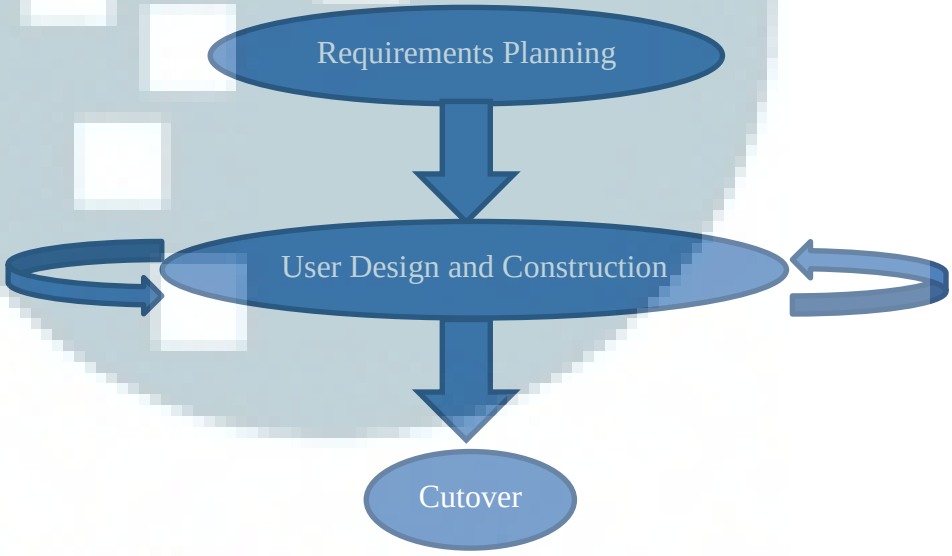
Sementara, berdasarkan fitur yang diimplementasikan, *prototype* dibagi menjadi teknik vertikal dan *horizontal*. *Vertical prototype* mengandung fungsi yang detil tetapi hanya untuk beberapa fitur terpilih, tidak pada keseluruhan fitur sistem. Sementara, *horizontal protoype* mencakup seluruh fitur antarmuka pengguna tetapi tanpa fungsi pokok, hanya berupa simulasi dan belum dapat digunakan untuk melakukan pekerjaan yang sebenarnya.

2.7. *Rapid Application Development* (RAD)

Menurut Pressman (2012), RAD adalah proses model perangkat lunak inkremental yang menekankan siklus pengembangan yang singkat. Model RAD adalah sebuah adaptasi "kecepatan tinggi" dari model *waterfall*, di mana perkembangan pesat dicapai dengan menggunakan pendekatan konstruksi berbasis komponen. Jika tiap-tiap kebutuhan dan batasan ruang lingkup proyek telah diketahui dengan baik, proses RAD memungkinkan tim pengembang untuk menciptakan sebuah "sistem yang berfungsi penuh" dalam jangka waktu yang sangat singkat. Dari penjelasan Pressman (2012) ini, satu perhatian khusus mengenai metodologi RAD dapat diketahui, yakni impelementasi metode RAD akan berjalan maksimal jika pengembang aplikasi telah merumuskan kebutuhan dan ruang lingkup pengembangan aplikasi dengan baik.

Sedangkan menurut Kendall (2010), RAD adalah suatu pendekatan berorientasi obyek terhadap pengembangan sistem yang mencakup suatu metode pengembangan serta perangkat-perangkat lunak. RAD bertujuan mempersingkat waktu yang biasanya diperlukan dalam siklus hidup pengembangan sistem tradisional antara perancangan dan penerapan suatu sistem informasi.

Menurut Cashman (2009), terdapat 4 fase yang terjadi dalam siklus *Rapid Application Development*:



Gambar 2.4 Siklus RAD

(Sumber: Cashman, 2009)

1. *Requirements Planning*

   Pada fase ini, pengguna dan penganalisis bertemu untuk menentukan tujuan-tujuan aplikasi atau sistem serta untuk mengidenfitikasikan syarat-syarat informasi yang ditimbulkan dari tujuan tersebut. Orientasi dalam fase ini adalah menyelesaikan masalah-masalah perusahaan. Meskipun

teknologi informasi dan sistem dapat berubah sebagian dari sistem yang diajukan, fokusnya akan selalu tetap pada upaya pencapaian tujuan-tujuan perusahaan.

2. *User Design*

   Pada fase ini, penganalisis merancang dan memperbaiki rancangan sistem yang akan dibuat, dapat disebut pula sebagai *workshop*. Penganalisis dan pemrogram dapat bekerja membangun dan menunjukkan representasi visual desain dan pola kerja kepada pengguna.

3. *Construction*

   Pada fase ini, penganalisis dan pemrogram membuat sistem dari hasil desain yang telah disetujui oleh pengguna. Selama fase ini, pengguna akan merespon prototipe yang ada dan penganalisis akan memperbaiki modul-modul yang dirancang berdasarkan respon pengguna.

4. *Cutover*

   Pada fase ini, sistem akan disetujui lalu diimplementasikan. Kemudian akan dilakukan uji coba kepada sistem dan perkenalan kepada organisasi.

Menurut Pressman (2012), pendekatan RAD mempunyai beberapa kelemahan, yaitu:

a. Untuk proyek yang berskala besar, RAD membutuhkan sumber daya manusia yang cukup untuk membuat tim RAD dengan jumlah yang tepat.
b. RAD membutuhkan pengembang dan pelanggan yang berkomitmen terhadap aktivitas gerak cepat yang dibutuhkan untuk membuat sistem selesai pada jangka waktu terbatas. Jika komitmen yang dimiliki terbatas, proyek RAD akan gagal.
c. Tidak semua jenis aplikasi cocok dengan RAD. Jika sistem tidak dapat dimodularisasi, membangun komponen yang penting bagi RAD dapat merepotkan.
d. RAD tidak layak digunakan ketika risiko teknis tinggi. Hal ini dapat terjadi ketika sebuah aplikasi baru menggunakan teknologi baru secara besar atau ketika *software* baru membutuhkan kerja sama yang tinggi dengan program komputer yang ada.

2.8. *Unified Modeling Language* (UML)

Menurut Nugroho (2010:6), UML (*Unified Modeling Language*) adalah bahasa pemodelan untuk sistem atau perangkat lunak yang berparadigma (berorientasi objek). Pemodelan sesungguhnya digunakan untuk penyederhanaan permasalahan-permasalahan yang kompleks sedemikian rupa sehingga lebih mudah dipelajari dan dipahami.

Menurut Widodo (2011), UML adalah bahasa pemodelan standar yang memiliki sintak dan semantik. UML sendiri terdiri dari beberapa jenis, yaitu:

1. *Use Case Diagram*

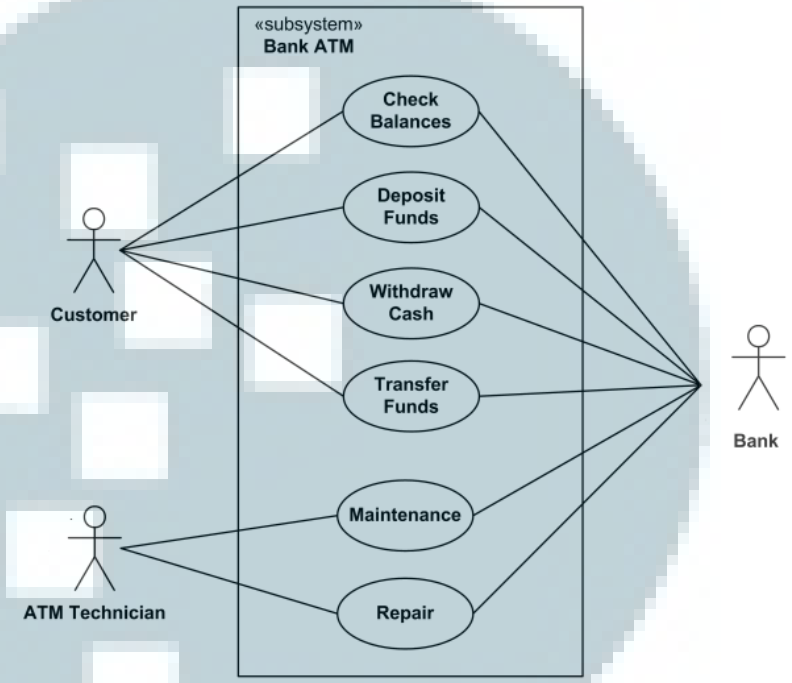
   Use case diagram merupakan sebuah jenis *diagram* pada UML yang menggambarkan interaksi antara sistem dengan *actor* atau penggunanya. *Use case diagram* umumnya bersifat menjelaskan secara detail, tipe interaksi antara pengguna dengan sistem. Tabel 2.2 akan menunjukkan simbol-simbol yang digunakan pada *use case diagram*:

Tabel 2.2 Simbol *Use Case Diagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*Actor*	Menspesifikasikan himpuan peran yang pengguna mainkan ketika berinteraksi dengan *use case*.
2	<< include >> 	*Include*	Menspesifikasikan bahwa *use case* sumber secara *eksplisit*.
3	<< extend >> 	*Extend*	Menspesifikasikan bahwa *use case* target memperluas perilaku dari *use case* sumber pada suatu titik yang diberikan.
4		*Association*	Apa yang menghubungkan antara objek satu dengan objek lainnya.
5		*System*	Menspesifikasikan paket yang menampilkan sistem secara terbatas.
6		*Use Case*	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor

(Sumber: https://widuri.raharja.info/index.php/SI1114465646)

Berdasarkan simbol-simbol tersebut, berikut ini adalah contoh penggunaan *use case diagram* yang menggambarkan sebuah sistem yang ada pada *bank*:



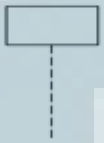
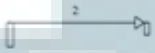
Gambar 2.5 Contoh *Use Case Diagram*

(Sumber: dumetschool.com)

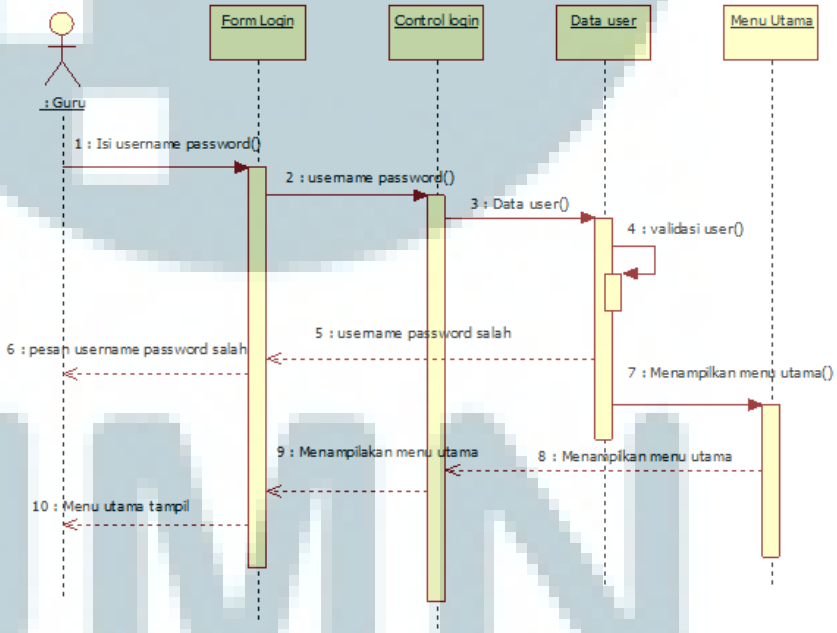
2. *Sequence diagram*

Sequence diagram merupakan jenis *diagram* dalam UML yang mendeskripsikan interaksi antar obyek berdasarkan urutan waktu. *Sequence diagram* sendiri memiliki fungsi untuk menggambarkan langkah yang harus dilakukan untuk dapat menghasilkan sesuatu. Tabel 2.3 akan menjelaskan simbol-simbol yang digunakan pada *sequence diagram* dan Gambar 2.6 akan menunjukkan contoh penggunaan *sequence diagram*:

Tabel 2.3 Simbol *Sequence Diagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*LifeLine*	Objek *entity*, antarmuka yang saling berinteraksi.
2		*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi
3		*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi

(Sumber: https://widuri.raharja.info/index.php/SI1111465712)



Gambar 2.6 Contoh *Sequence Diagram*

(Sumber: tutorialkampus.com)

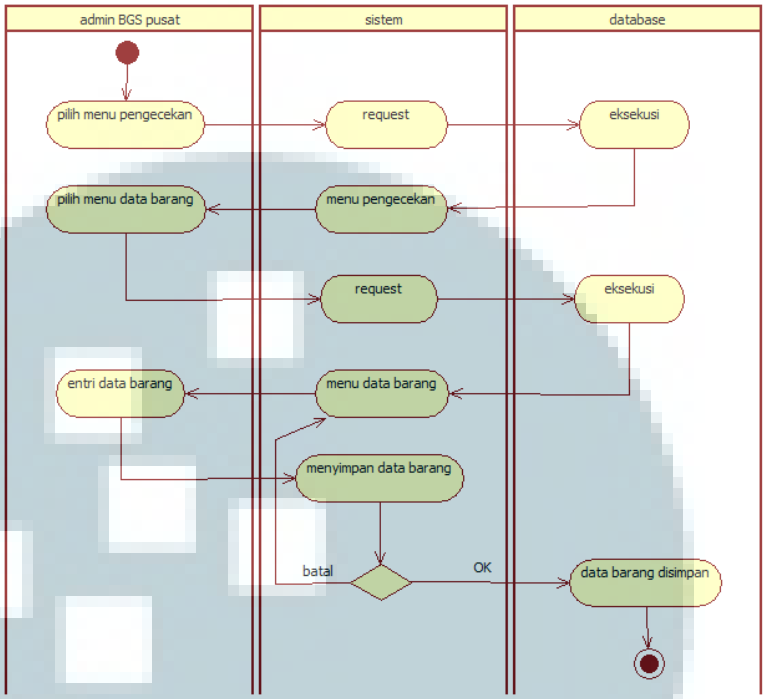
3. *Activity diagram*

Activity diagram merupakan *diagram* yang menggambarkan alur aktivitas atau kerja dalam suatu sistem. Tabel 2.4 berisi simbol-simbol yang digunakan pada *activity diagram* dan Gambar 2.7 adalah contoh penggunaan *activity diagram*:

Tabel 2.4 Simbol *Activity Diagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*Actifity*	Memperlihatkan bagaimana masing-masing kelas antarmuka saling berinteraksi satu sama lain
2		*Action*	State dari sistem yang mencerminkan eksekusi dari suatu aksi
3		*Initial Node*	Bagaimana objek dibentuk atau diawali.
4		*Actifity Final Node*	Bagaimana objek dibentuk dan dihancurkan
5		*Fork Node*	Satu aliran yang pada tahap tertentu berubah menjadi beberapa aliran

(Sumber: https://widuri.raharja.info/index.php/SI1111465712)



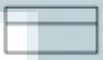
Gambar 2.7 Contoh *Activity Diagram*

(Sumber: tutorialkampus.com)

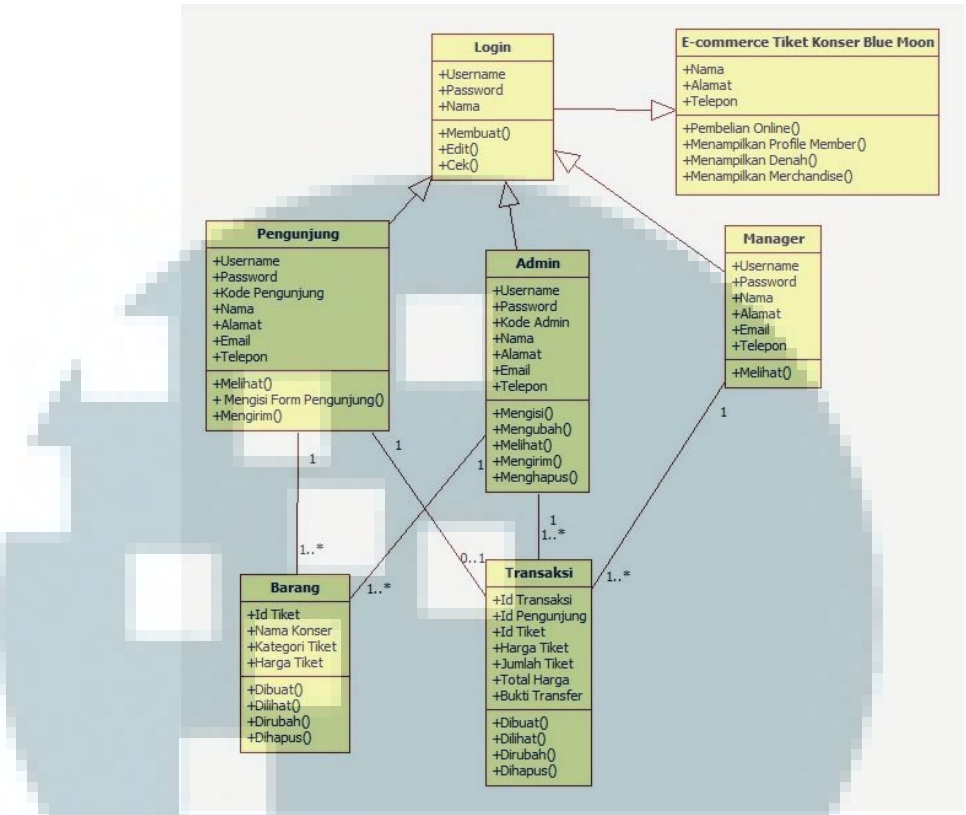
4. *Class Diagram*

Class diagram merupakan sebuah *diagram* pada UML yang berfungsi untuk menggambarkan kelas-kelas dalam sebuah sistem dan relasinya antara satu dengan yang lain, serta terdapat pula atribut dan operasinya. Tabel 2.5 berisi simbol-simbol yang digunakan pada *class diagram* dan Gambar 2.8 merupakan contoh penggunaan *class diagram*:

Tabel 2.5 Simbol *Class Diagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*Generalization*	Hubungan dimana objek anak *(descendent)* berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*).
2		*Class*	Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama.
3		*Collaboration*	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu actor
4		*Realization*	Operasi yang benar-benar dilakukan oleh suatu objek.
5		*Dependency*	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri *(independent)* akan mempegaruhi elemen yang bergantung padanya elemen yang tidak mandiri
6		*Association*	Apa yang menghubungkan antara objek satu dengan objek lainnya

(Sumber: https://widuri.raharja.info/index.php/SI1111465712)



Gambar 2.8 Contoh *Class Diagram*

(Sumber: tutorialkampus.com)

2.9. Simulasi

Menurut Law dan Kelton (2000), simulasi merupakan suatu teknik meniru operasi-operasi atau proses-proses yang terjadi dalam suatu sistem dengan bantuan perangkat komputer dan dilandasi oleh beberapa asumsi tertentu sehingga sistem tersebut dapat dipelajari secara ilmiah.

Menurut Law dan Kelton (2000), simulasi merupakan alat yang tepat untuk digunakan terutama jika diharuskan untuk melakukan eksperimen dalam rangka mencari kemungkinan terbaik dari komponen-komponen sistem. Hal ini dikarenakan sangat mahal dan memerlukan

waktu yang lama jika eksperimen dicoba secara riil. Dengan melakukan studi simulasi maka dalam waktu singkat dapat ditentukan keputusan yang tepat serta dengan biaya yang tidak terlalu besar karena semuanya cukup dilakukan dengan komputer.

Pendekatan simulasi diawali dengan pembangunan model sistem nyata. Model tersebut harus dapat menunjukkan bagaimana berbagai komponen dalam sistem saling berinteraksi sehingga benar-benar menggambarkan perilaku sistem. Setelah model dibuat maka model tersebut akan ditransformasikan ke dalam program komputer sehingga memungkinkan untuk disimulasikan.

Pada dasarnya model simulasi dikelompokkan menjadi tiga dimensi yaitu:

a. Model Simulasi Statis dengan Model Simulasi Dinamis

   Model simulasi statis digunakan untuk merepresentasikan sistem pada saat tertentu atau sistem yang tidak terpengaruh oleh perubahan waktu. Sedangkan model simulasi dinamis digunakan jika sistem yang dikaji dipengaruhi oleh perubahan waktu.

b. Model Simulasi Deterministik dengan Model Simulasi Stokastik

   Jika model simulasi yang akan dibentuk tidak mengandung variabel yang bersifat acak, model simulasi tersebut dikatakan sebagai smulasi deterministik. Pada umumnya, sistem yang

dimodelkan dalam simulasi mengandung beberapa *input* yang bersifat acak, maka simulasi yang cocok dibangun untuk sistem seperti ini adalah model simulasi stokastik.

c. Model Simulasi Kontinu dengan Model Simulasi Diskrit

Suatu sistem dikatakan diskrit jika variabel sistem yang mencerminkan status sistem berubah pada titik waktu tertentu, sedangkat kontinu jika perubahan variabel sistem berlangsung secara berkelanjutan seiring dengan perubahan waktu.

2.10. *Hypertext Preprocessor* (PHP)

Gambar 2.9 Logo PHP

(**Sumber: w3schools.com)**

Menurut Yank (2012), PHP merupakan *server-side scripting language* yang dapat digunakan pengguna untuk memasukkan berbagai macam instruksi ke dalam halaman *web* yang nantinya akan dieksekusi oleh *software* pada umumnya seperti *Apache* sebelum halaman tersebut dikirim ke *browser* yang memintanya. Ciri khas dari bahasa PHP adalah adanya penggunaan *tag* yang diawali dengan "<?php" dan diakhiri dengan "?>".

Menurut Buana (2014:9), PHP merupakan aplikasi perangkat lunak *opensource*, di mana kepanjangan dari PHP adalah *Hypertext Preprocessor* yang diatur dalam aturan *General Purpose Licenses* (GPL). Pemrograman PHP merupakan pemrograman yang sangat cocok dikembangkan di lingkungan *web* karena dapat diletakkan pada *script* HTML ataupun sebaliknya. PHP tergolong sebagai pemrograman *web* dinamis karena mampu menghasilkan *website* yang dapat diubah secara terus menerus hasilnya atau kontennya tanpa harus masuk ke dalam *coding*. Hal tersebut bergantung pada permintaan terkini. Secara umum, pembuatan *database* sangat erat hubungannya untuk pembuatan *web* dinamis, sebagai tempat untuk sumber data yang akan digunakan.

PHP merupakan bahasa pemrograman berbasis *server*. Hal ini berarti setiap pemrograman PHP harus diletakkan di *server* terlebih dahulu, kemudian diterjemahkan oleh *web server* dan hasilnya dikirim ke *browser client*. Kemampuan dan fitur PHP yang paling mendukung banyak basis data yaitu, MSSQL, MySQL, Oracle, dan Postgre SQL. Pada bulan Juni 2014, PHP versi 5 dirilis dan sudah ditanamkan model yang banyak digunakan di semua bahasa pemrograman. Secara teknologi, bahasa pemrograman PHP sangat mirip dengan bahasa pemrograman yang berbasis *web* lainnya, contohnya bahasa ASP (*Active Server Page*), *Cold Fusion*, JSP (*Java Server Page*), ataupun Perl.

2.11. *Hypertext Markup Language* (HTML)

Gambar 2.10 Logo HTML 5

(Sumber: w3schools.com)

Menurut Willard (2006), *Hyper Text Markup Language* (HTML) adalah sebuah bahasa markah yang digunakan untuk membuat sebuah halaman *web*. HTML adalah sebuah standar yang digunakan secara luas untuk menampilkan halaman *web*. HTML dapat berupa sebuah kumpulan dari beberapa instruksi yang dapat digunakan untuk mengubah-ubah format suatu dokumen. HTML sendiri sudah memiliki banyak versi, yang terbaru adalah HTML5. HTML5 merupakan versi terbaru teknologi *hypertext* yang saat ini sedang dalam tahap *development.*

2.12. *Cascading Style Sheet* (CSS)

Gambar 2.11 Logo CSS 3

(Sumber: w3schools.com)

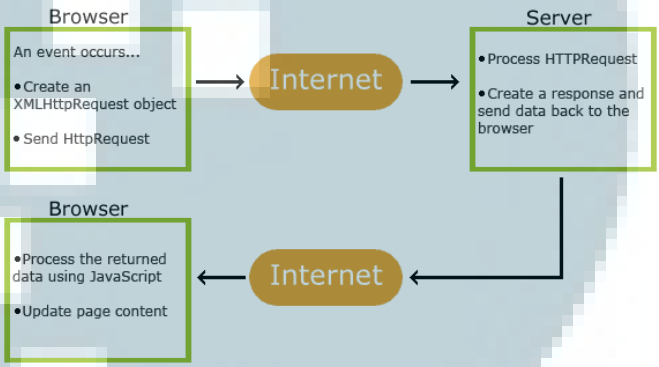
Menurut W3Schools (diakses tanggal 21 September 2016), *Cascading Style Sheet* mendeskripsikan bagaimana unsur-unsur dalam HTML ditampilkan dalam layar, kertas, maupun media lainnya. Dalam CSS, terdapat sebuah *tools* yang berguna untuk menciptakan kerangka yang menggabungkan unsur serta animasi yang lebih menyediakan *feedback* kepada pengguna.

2.13. *Asynchronous JavaScript and XMLHTTP*

Menurut W3Schools (diakses tanggal 29 November 2016), *Asynchronous JavaScript and XMLHTTP* atau disingkat AJaX adalah suatu teknik pemrograman berbasis *web* untuk menciptakan aplikasi *web* interaktif. AJaX bukanlah sebuah bahasa pemrograman melainkan sebuah teknik untuk mengakses *web servers* dari sebuah halaman *web*. AJaX menggunakan kombinasi dari *XMLHttpRequest* untuk meminta data dari

sebuah *web server* dan *JavaScript* dan HTML DOM untuk menampilkan atau menggunakan data tersebut. AJaX memungkinkan sebuah halaman *web* untuk diperbarui dengan menukar data pada *web server* tanpa diketahui pengguna. Hal ini berarti AJaX memungkinkan sebuah *web* untuk diperbarui beberapa bagiannya tanpa memuat ulang seluruh halaman *web*.

Berikut ini adalah cara kerja dari AJaX:



Gambar 2.12 Cara Kerja AJaX

(Sumber: w3schools.com/xml/ajax_intro.asp)

2.14. Penelitian Terdahulu

Terdapat 3 penelitian sejenis yang sudah dikembangkan sebelumnya yaitu *The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is FFD(I) ≤ 11/9 OPT(I) + 6/9* oleh György Dósa, *An Efficient Algorithm for 3D Rectangular Box Packing* oleh M. Zahid Gürbüz *et al.* (2009), dan Simulasi Pencarian Ruang Kosong Kontainer untuk Memaksimalkan Penempatan Barang oleh Andrias Rusli yang peneliti gunakan sebagai referensi dalam mengembangkan penelitian ini.

Pada penelitian yang dilakukan oleh Dósa (2007) yang berjudul *The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is FFD(I) ≤ 11/9 OPT(I) + 6/9*, disinggung mengenai algoritma *first fit decreasing* dalam mengatasi masalah pengemasan. Algoritma *first fit decreasing* akan mengemas setiap barang yang ada ke dalam wadah dengan mengurutkannya secara *decreasing* berdasarkan ukuran barang tersebut atau dengan kata lain mengurutkan barang berdasarkan dengan ukuran terbesar hingga terkecil. Setelah mengurutkan barang, algoritma ini akan memasukkan setiap barang ke dalam wadah yang ada sebagaimana algoritma *first fit* yaitu memasukkan barang ke dalam wadah pertama dan akan membuka wadah baru jika barang tidak dapat dimasukkan ke dalam wadah yang sudah ada. Contohnya adalah diberikan sejumlah wadah w_1, w_2, w_3 dengan ukuran masing-masing 30 dan barang A, B, C dengan ukuran masing-masing 5, 30, dan 15. Pertama, algoritma ini akan mengurutkan barang berdasarkan ukuran terbesar terlebih dahulu yaitu B, C, dan A. Setelah mengurutkan barang, algoritma ini akan mengemas barang B ke dalam wadah w_1 dimana sisa ruang dalam w_1 adalah 0. Barang kedua yaitu C yang memiliki ukuran 15 akan dikemas ke dalam wadah w_1 terlebih dahulu dan ternyata tidak muat, maka barang C akan dikemas ke dalam wadah w_2 dan sisa ruang pada wadah w_2 adalah 15. Lalu barang ketiga yaitu A yang memiliki ukuran 5 akan dikemas ke dalam wadah w_2 dikarenakan wadah w_1 tidak memiliki ruang kosong lagi dan wadah w_2 masih tersedia.

Dari penelitian tersebut, peneliti dapat menyimpulkan cara kerja algoritma *first fit decreasing* dimana algoritma ini akan mengemas barang berdasarkan ukuran terbesar tanpa memperhatikan aspek lainnya ke dalam wadah yang tersedia. Algoritma ini akan terus melakukan pengulangan pada setiap wadah yang ada dimana setiap barang yang ada akan dimasukkan pada wadah pertama, wadah kedua, dan seterusnya hingga ditemukan wadah yang cocok dan akan mengulangnya untuk barang selanjutnya. Terdapat kelemahan utama dalam algoritma ini dimana membutuhkan sumber daya untuk melakukan proses perulangan tersebut dikarenakan algoritma ini akan terus memeriksa semua wadah yang ada. Jika wadah yang digunakan banyak, maka proses pengemasan akan semakin lama pula. Namun, terdapat pula kelebihan dari algoritma ini dimana proses pengemasan barang tidak membutuhkan banyak perhitungan atau operasi dimana hanya perlu menghitung besar ukuran setiap barang saja dan langsung memasukkannya ke dalam wadah.

Penelitian selanjutnya yang digunakan sebagai referensi oleh peneliti adalah penelitian yang dilakukan oleh M. Zahid Gürbüz *et al.* (2009) yang berjudul *An Efficient Algorithm for 3D Rectangular Box Packing*. Pada penelitian ini, dijabarkan mengenai sebuah algoritma baru yaitu *Largest Area First-Fit* (LAFF). Algoritma ini akan mengemas barang ke dalam wadah dengan melihat barang yang memiliki luas permukaan terbesar terlebih dahulu. Jika ditemukan dua barang dengan luas permukaan yang sama besarnya, algoritma ini akan memeriksa tinggi

dari kedua barang tersebut dimana barang yang memiliki tinggi lebih rendah akan dikemas terlebih dahulu ke dalam wadah. Setelah mengemas barang dengan luas permukaan terbesar, algoritma ini akan memeriksa apakah masih tersedia ruang kosong di sekitar barang dan barang mana yang muat pada ruang kosong tersebut. Jika tidak ada, algoritma ini akan menaruh barang diatas barang tertinggi yang sudah masuk dan seterusnya hingga wadah penuh.

Dari penelitian tersebut, peneliti dapat menyimpulkan cara kerja dari algoritma *largest area first-fit.* Menurut peneliti, algoritma LAFF memiliki kelebihan dimana algoritma ini mengutamakan efisiensi dalam penggunaan wadah. Algoritma LAFF akan mengemas barang ke dalam sebuah wadah dengan mengutamakan barang yang memiliki tinggi terendah terlebih dahulu tetapi memiliki lebar terbesar. Namun, terdapat kelemahan dari algoritma LAFF yang penulis temukan yaitu pada saat algoritma ini mengemas benda dan tingkat bawah sudah penuh, maka algoritma ini akan membuat sebuah tingkat baru berdasarkan dengan tinggi tertinggi benda didalamnya. Hal ini akan menyebabkan ruang kosong yang terbuang di dalam kontainer. Terlebih lagi, jika algoritma ini menemukan barang dengan luas terbesar dan akan langsung mengemasnya ke dalam kontainer tanpa membandingkan tinggi benda tersebut maka ruang kosong yang ada dapat semakin besar. Selain itu, kekurangan dari algoritma LAFF ini adalah pada penelitian yang terdahulu hanya menggunakan sebuah bentuk yaitu kubus atau balok.

Penelitian selanjutnya yang peneliti gunakan sebagai referensi adalah penelitian yang dilakukan oleh Rusli (2016) yang berjudul Simulasi Pencarian Ruang Kosong Kontainer untuk Memaksimalkan Penempatan Barang. Pada penelitian ini kembali digunakan algoritma LAFF dalam mengemas barang ke dalam kontainer. Tidak hanya menganalisa, Rusli juga merancang sebuah simulasi dua dimensi yang dapat digunakan secara langsung oleh pengguna. Namun, pada penelitian tersebut terdapat kekurangan dimana simulasi hanya berbentuk 2D, hanya memiliki sebuah ukuran kontainer, dan hanya menggunakan sebuah jenis barang yaitu kubus atau balok.

Tabel 2.6 Perbandingan Penelitian Terdahulu

No.	Peneliti	Tahun	Judul	Metode	Hasil
1	György Dósa	2007	*The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is FFD(I) ≤ 11/9 OPT(I) + 6/9*	*First fit decreasing*	*The bound of first fit decreasing is tight with FFD(I) ≤ 11/9 OPT(I) + 6/9*
2	M. Zahid Gürbüz *et al.*	2009	*An Efficient Algorithm for 3D Rectangular Box Packing*	*Largest area first-fit*	Algoritma baru untuk permasalahan pengemasan 3D yaitu algoritma LAFF
3	Andrias Rusli	2016	Simulasi Pencarian Ruang Kosong Kontainer untuk Memaksimalkan Penempatan Barang	*Largest area first-fit*	Simulasi penataan barang ke dalam kontainer