BAB II

TINJAUAN PUSTAKA

2.1 Sorting

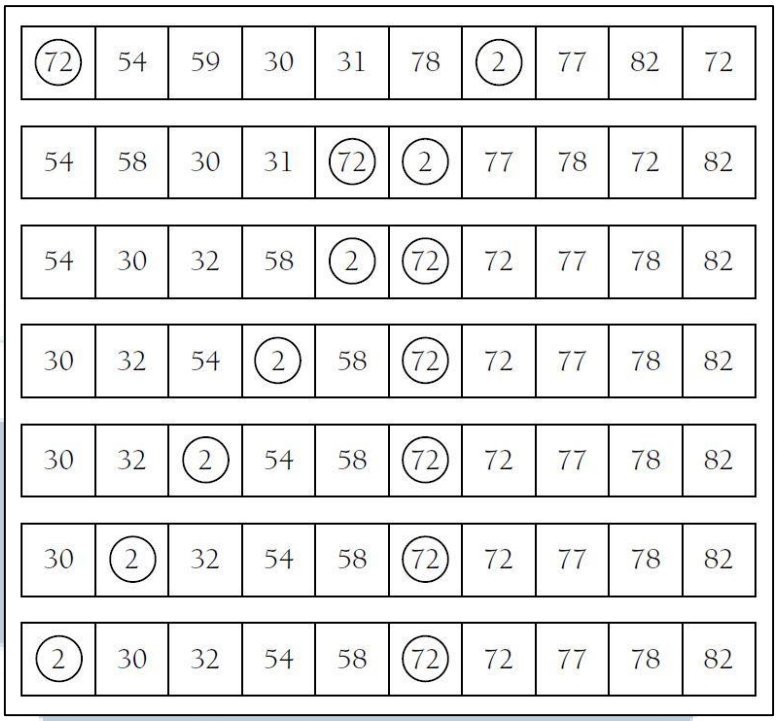
Pengurutan (*sorting)* dipahami sebagai sebuah proses untuk mengatur sekumpulan objek menjadi suatu susunan tertentu (Wirth, 1976). Terdapat beberapa algoritma *sorting* menurut McMillan (2005).

1. *Basic Sorting Algorithm*
    a. *Bubble Sort*
    b. *Selection Sort*
    c. *Insertion Sort*
2. *Advanced Sorting Algorithm*
    a. *Shell Sort*
    b. *Merge Sort*
    c. *Heap Sort*
    d. *Quicksort*

Basic Sorting Algorithm hanya memerlukan pengetahuan perbandingan dan *array*, sedangkan *Advanced Sorting Algorithm* membutuhkan pengetahuan mengenai *divide and conquer* dan *tree* (Munir dan Lidya, 2016).

2.1.1 Bubble Sort

Bubble sort biasanya dianggap sebagai algoritma pengurutan termudah, sehingga sering diajarkan pada awal pembelajaran pemrograman. Performa yang jelek mengakibatkan algoritma ini jarang digunakan untuk jumlah data yang besar (Canaan, Garai dan Daya, 2011). Meskipun begitu, *bubble sort* mudah untuk diingat dan diprogram, serta dapat menyelesaikan satu langkah dalam waktu singkat (Martin, 1971). Contoh proses *bubble sort* dapat dilihat pada Gambar 2.1.

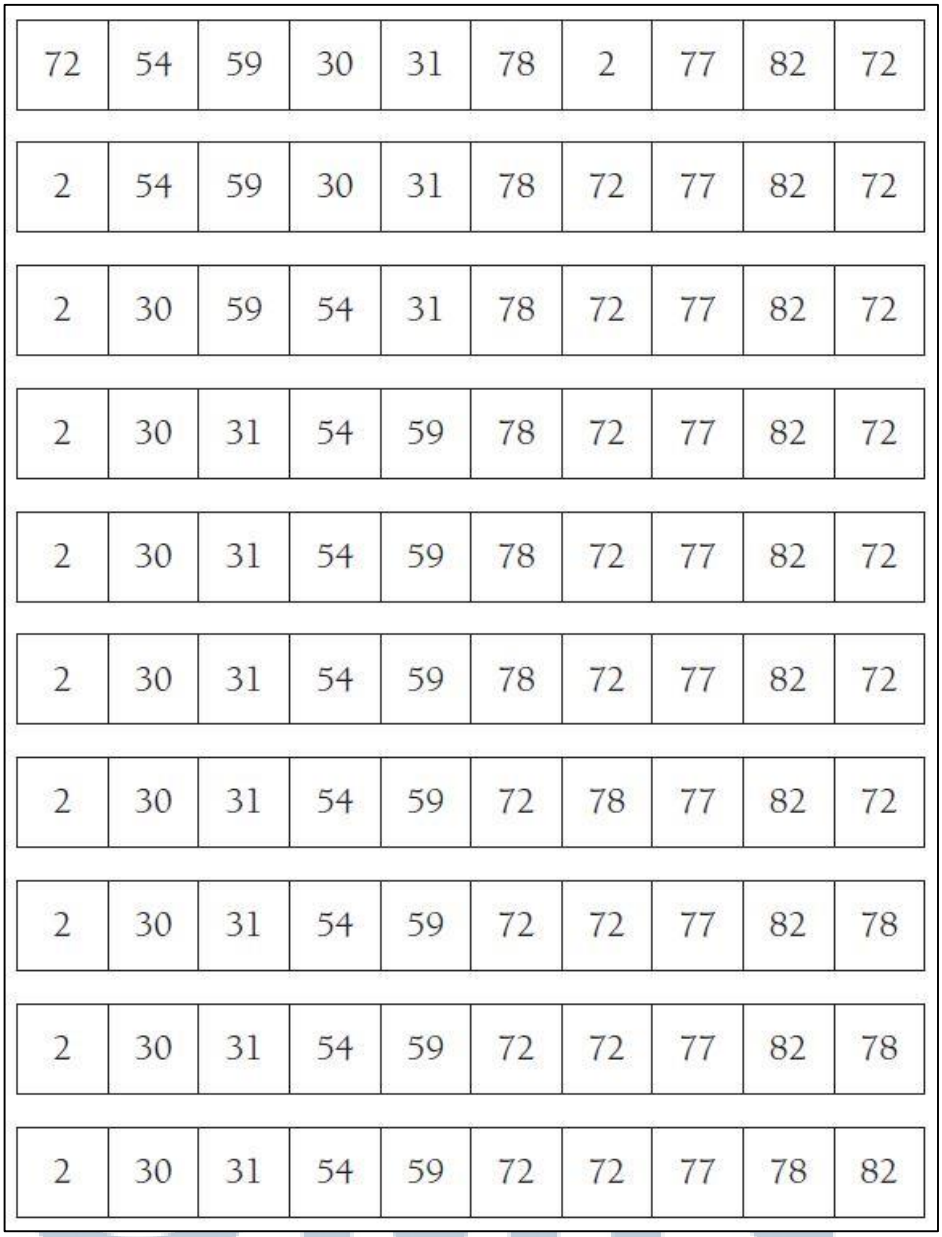


Gambar 2.1 Ilustrasi Cara Kerja *Bubble Sort*
(McMillan, 2005)

McMillan (2005) menyatakan bahwa cara kerja *bubble sort* adalah dengan melakukan iterasi pada *array* berkali-kali, membandingkan nilai yang berdekatan, dan melakukan pertukaran nilai apabila nilai kiri lebih besar daripada nilai kanan.

2.1.2 Selection Sort

Selection Sort merupakan proses pengurutan data yang mendekati intuisi manusia, yaitu menemukan elemen dengan nilai terbesar atau terkecil kemudian menempatkannya pada posisi yang tepat dengan melakukan *swap* pada nilai terbesar atau terkecil sebelumnya (Canaan, Garai dan Daya, 2011). Apabila *swap* sudah pernah dilakukan, maka elemen yang dipindahkan akan "diisolasi" dan tidak diikutkan untuk proses *sort* selanjutnya. Contoh proses *selection sort* dapat dilihat pada Gambar 2.2.



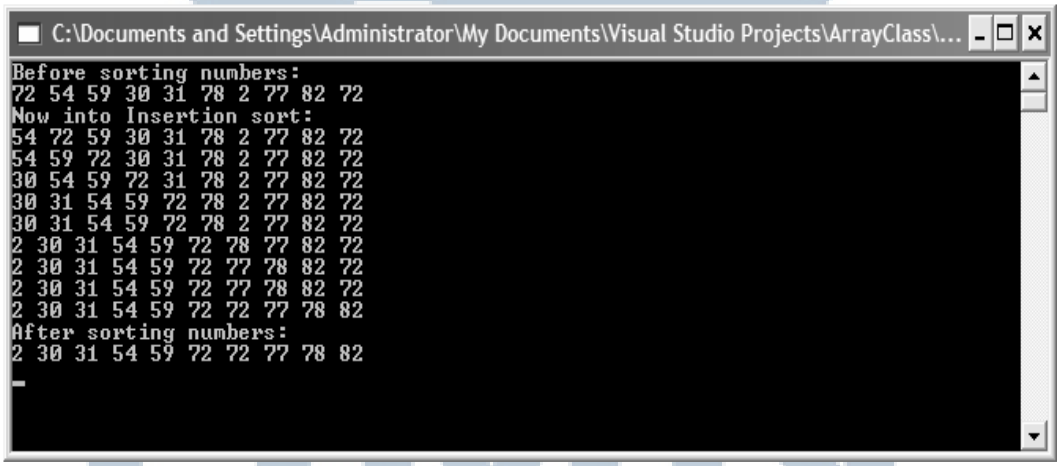
72	54	59	30	31	78	2	77	82	72
2	54	59	30	31	78	72	77	82	72
2	30	59	54	31	78	72	77	82	72
2	30	31	54	59	78	72	77	82	72
2	30	31	54	59	78	72	77	82	72
2	30	31	54	59	78	72	77	82	72
2	30	31	54	59	72	78	77	82	72
2	30	31	54	59	72	72	77	82	78
2	30	31	54	59	72	72	77	82	78
2	30	31	54	59	72	72	77	78	82

Gambar 2.2 Ilustrasi Cara Kerja *Selection Sort*
(McMillan, 2005)

Menurut McMillan(2005), cara kerja *selection sort* adalah memulai dari posisi *array* 0, membandingkan nilai posisi *array* 0 tersebut dengan nilai pada posisi *array* lainnnya, dan melakukan pertukaran nilai pada posisi *array* 0 dan posisi *array* dengan nilai terkecil. Proses *sorting* kemudian diulang pada posisi *array* 1. Proses ini dilakukan berulang kali sampai semua posisi *array* kecuali posisi terakhir telah digunakan untuk posisi awal pada *loop*.

2.1.3 Insertion Sort

Insertion Sort merupakan algoritma pengurutan dengan menyisipkan (*insertion)* suatu elemen pada posisi tertentu menuju posisi yang tepat sesuai urutan. Terjadi banyak pergeseran saat melakukan *sort,* karena itu algoritma ini memakan waktu yang lama untuk menyelesaikan proses apabila volume data besar (Munir dan Lidya, 2016). Namun *insertion sort* mudah untuk diimplementasikan, efisien untuk volume data kecil, dan juga dapat beradaptasi pada pertambahan data, karena cukup dilakukan penyisipan pada daftar lama yang sudah ter-*sort* sebelumnya (Canaan, Garai dan Daya, 2011). Contoh proses *insertion sort* dapat dilihat pada Gambar 2.3.

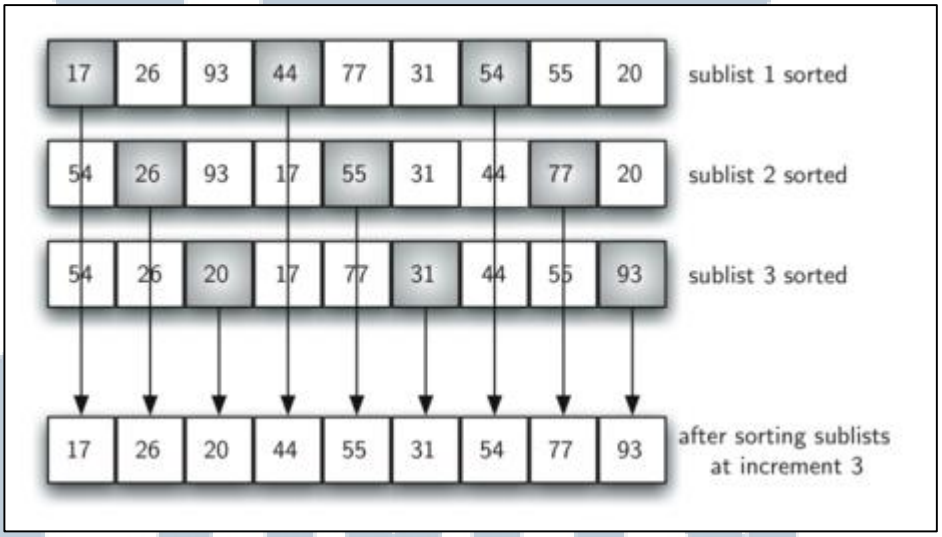


Gambar 2.3 Ilustrasi Cara Kerja *Insertion Sort*
(McMillan, 2005)

Menurut McMillan (2005), *insertion sort* berbeda dari *sort* lainnya karena perpindahan tidak dilakukan dengan menukar nilai antara dua posisi, namun dengan memindahkan elemen *array* dengan nilai lebih besar ke kanan untuk memberi ruang kepada elemen *array* yang memiliki nilai lebih rendah di bagian kiri *array*.

2.1.4 Shell Sort

Algoritma ini dinamakan dari penemunya, Donald Shell. Algoritma ini merupakan perkembangan dari *insertion sort*. Konsep utama algoritma ini adalah perbandingan antara indeks yang berjauhan, bukan yang saling berdekatan. Selama algoritma ini melakukan *looping* pada set data, jarak antar kedua indeks semakin berkurang hingga akhirnya algoritma akan membandingkan antar indeks yang saling berdekatan. Pada akhirnya, akan digunakan *insertion sort* untuk mengurutkan *array*. (McMillan, 2005). Contoh salah satu langkah dari *shell sort* dapat dilihat pada Gambar 2.4.

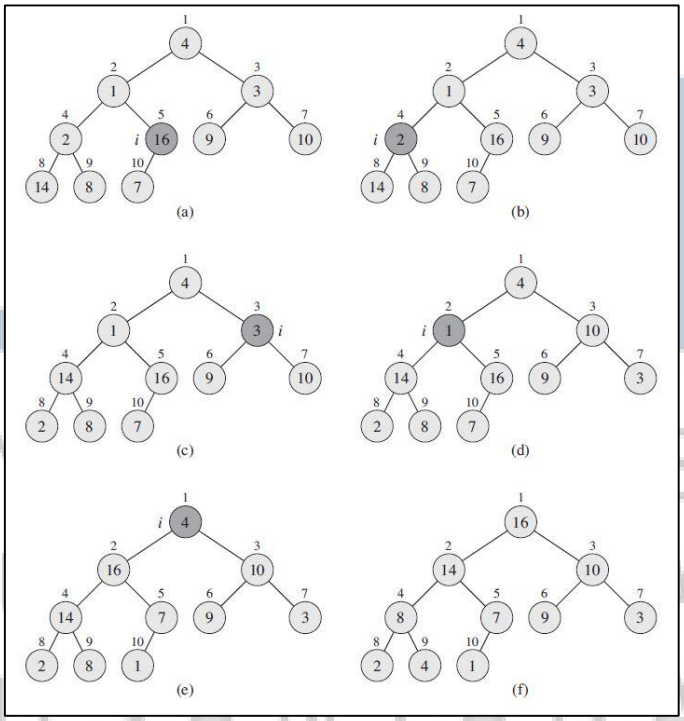


Gambar 2.4 Ilustrasi Langkah *Shell Sort*
(Miller dan David, 2013)

McMillan (2005) menyatakan bahwa setelah melakukan *shell sort*, *insertion sort* terakhir tidak perlu melakukan banyak pertukaran lagi karena *array* sudah di *pre-sort* dahulu oleh *shell sort*. Dengan kata lain, setiap *loop* menghasilkan *list* yang lebih terurut dari yang sebelumnya, sehingga *loop* terakhir menjadi sangat efisien.

2.1.5 Heap Sort

Heap Sort merupakan sebuah metode *sorting* yang menggunakan konsep *tree*. *Heap Sort* menggunakan sebuah struktur data bernama *heap* saat pengerjaannya. *Heap* merupakan satu set data yang membentuk suatu *complete binary tree*, dimana pada *binary tree* setiap *node* mempunyai 2 anak atau tidak sama sekali (Cormen et al., 2009). Pada *binary tree*, *node* yang merupakan *parent node* harus memiliki nilai yang lebih besar atau sama dengan *child node*. *Heap* dibuat dengan memasukkan *node* pada suatu *array heap.* Pada saat memasukkan nilai baru pada *array*, *node* harus diletakkan di tempat yang benar agar tidak mengubah struktur *heap*. Proses *sorting* dari *heap* sendiri adalah menarik *root node* satu demi satu sampai *binary tree* habis, dan pada setiap penarikan harus menyusun ulang *binary tree* agar tetap mengikuti aturan, proses ini disebut *heapify* (Canaan, Garai dan Daya, 2011).



Gambar 2.5 Ilustrasi Proses *Heapify* pada *Heap Sort*
(Cormen dkk., 2009)

Gambar 2.5 melambangkan *max-heap*, dimana nilai *node parent* yang berada pada posisi lebih tinggi harus memiliki nilai lebih besar dari *node child* yang berada di posisi bawah.

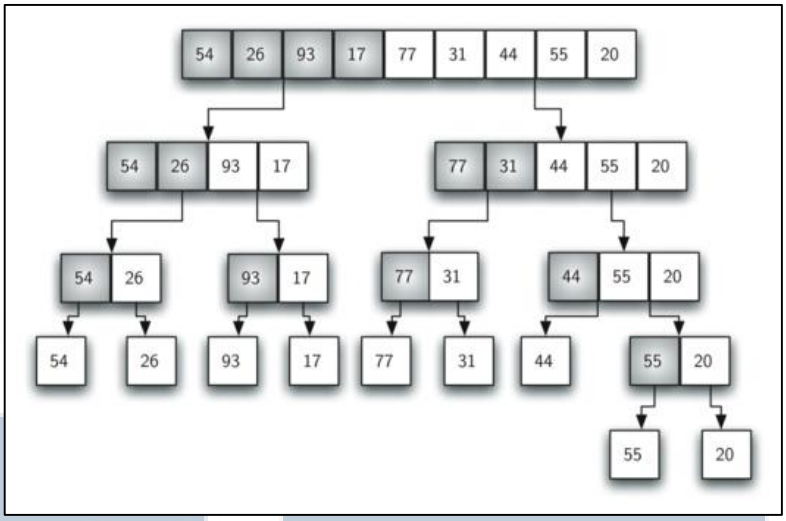
McMillan menyatakan bahwa terdapat 3 langkah untuk mengurutkan *array* menggunakan hasil heapify *max-heap*, tertulis sebagai berikut.

1. Hilangkan *node* pada *root* (posisi teratas).
2. Pindahkan *node* pada posisi terakhir ke *root*.
3. Susun ulang *tree* baru sehingga *node* yang baru bertukar posisi berada dibawah *node* dengan nilai lebih besar dan diatas *node* dengan nilai lebih kecil.

Menerapkan algoritma ini terus menerus akan membuat *node* yang dihilangkan terurut dari besar ke kecil.

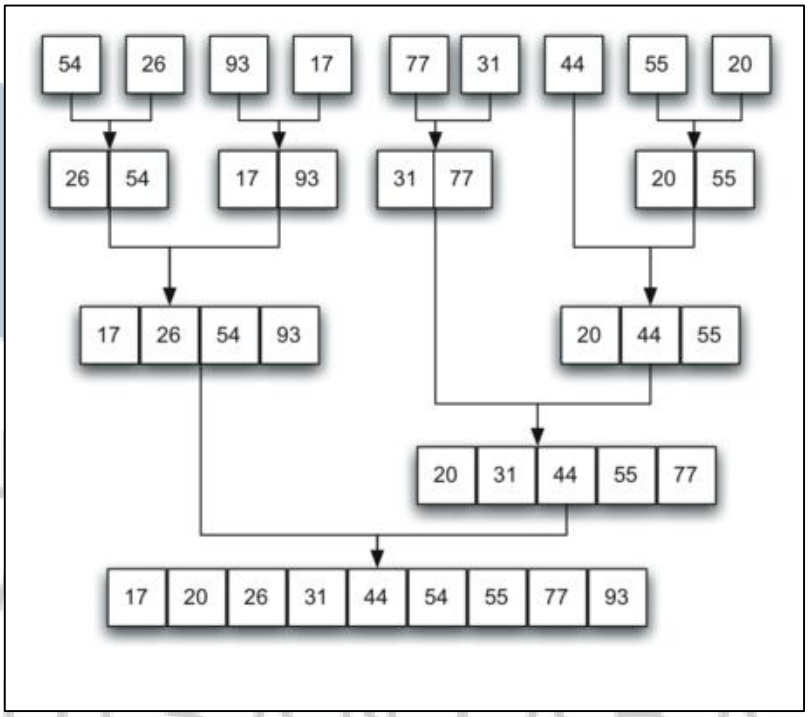
2.1.6 Merge Sort

Menurut McMillan (2005), algoritma *Merge Sort* menggunakan konsep *divide-and-conquer* untuk meningkatkan performa algoritma *sorting*. Sebuah set data akan dibagi setengah terus secara rekursif hingga hanya satu elemen yang tersisa, dan kemudian akan dimulai proses penggabungan. Satu elemen ini sudah dianggap terurut, kemudian setiap elemen akan dibandingkan dan disatukan kembali satu demi satu. Setiap penggabungan akan menghasilkan sebuah grup, dan ukuran grup akan semakin besar semakin banyak penggabungan yang terjadi. Contoh pembagian pada *merge sort* dapat dilihat pada Gambar 2.6.



Gambar 2.6 Contoh Pembagian pada *Merge Sort*
(Miller dan David, 2013)

Berdasarkan Gambar 2.6, terlihat bahwa sebuah *array* akan dibagi setengah berulang-ulang sampai hanya tersisa satu elemen *array* pada setiap hasil pembagian. Proses penyatuan *array* akan dilakukan setelah ini, seperti bisa dilihat pada Gambar 2.7.

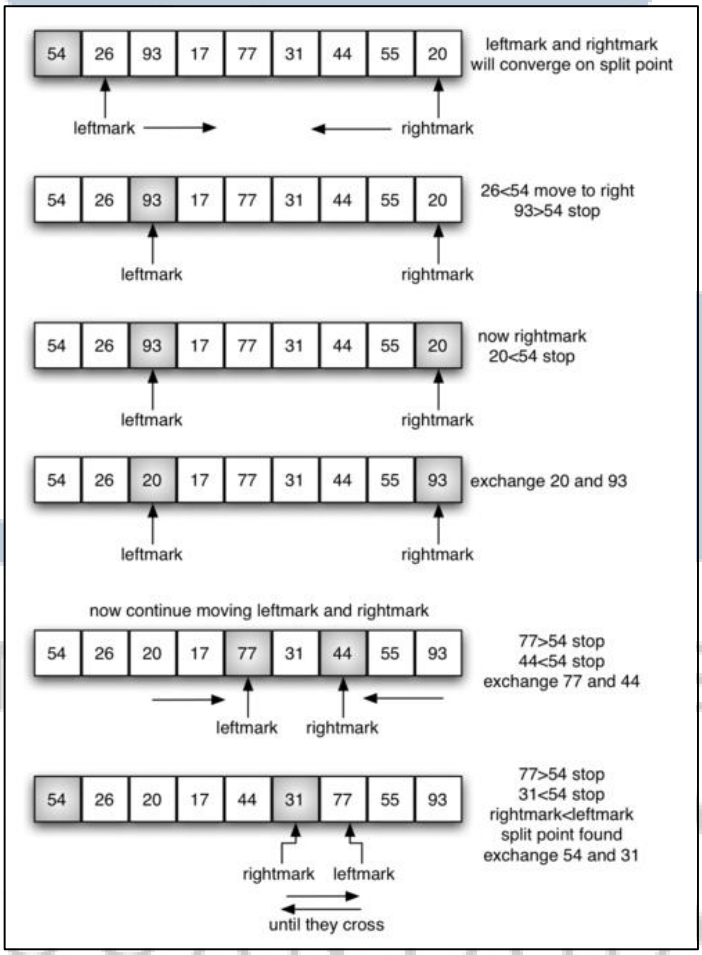


Gambar 2.7 Contoh Penyatuan pada *Merge Sort*
(Miller dan David, 2013)

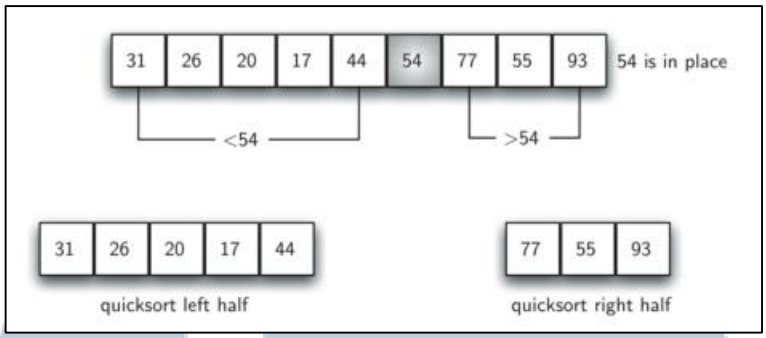
Berdasarkan Gambar 2.7, terlihat bahwa perbandingan akan dilakukan antara elemen *array* yang sudah dibagi, dan untuk setiap perbandingan akan terbentuk persatuan *array* baru.

2.1.7 Quick Sort

Menurut McMillan (2005), *quick sort* merupakan algoritma yang menggunakan konsep *divide and conquer*. Set data akan dibagi terus menerus berkali-kali sehingga membentuk banyak kelompok data, yang kemudian akan digabungkan kembali secara rekursif melalui proses *sorting* satu demi satu.Salah satu fitur unik *quick sort* adalah penggunaan *pivot. Pivot* merupakan angka "pembanding" yang dipilih menggunakan metode tertentu.



Gambar 2.8 Ilustrasi Proses *Quick Sort*
(Miller dan David, 2013)



Gambar 2.9 Hasil Satu *Loop Quick Sort*
(Miller dan David, 2013)

Berdasarkan Gambar 2.8 dan 2.9, pada setiap *loop* angka yang lebih kecil dari *pivot* akan berada di kiri, dan angka yang lebih besar akan berada di kanan (operasi ini disebut *partition operation).* Dengan melakukan ini, sudah terdapat 2 grup *subarray.* Angka *pivot* tersebut juga telah dianggap *sorted*, sehingga tidak akan diikutkan lagi pada iterasi selanjutnya. Kedua grup baru ini akan mengulangi proses rekursif pada diri mereka sendiri, hingga semua angka sudah menjadi *pivot* dan terurut (Canaan, Garai dan Daya, 2011).

2.2 Durstenfeld Shuffle

Durstenfeld Shuffle merupakan algoritma generator permutasi acak yang dibuat oleh Richard Durstenfeld (Durstenfeld, 1964). Algoritma ini dimodifikasi oleh Durstenfeld dari algoritma Fisher-Yates Shuffle. Kelebihan dari algoritma *shuffle* ini adalah pengacakan suatu *array* langsung pada *array* itu sendiri, sehingga tidak perlu membuang memori untuk pembuatan *array* baru sebagai penampung hasil pengacakan. Serupa juga dengan Fisher-Yates Shuffle, algoritma ini bersifat *unbiased*, agar berarti setiap permutasi memiliki peluang yang sama untuk keluar.

Langkah-langkah untuk melakukan *Durstenfeld Shuffle* adalah sebagai berikut (Carr, 2013).

1. Tentukan panjang dari *array* yang akan diacak.

2. Lakukan *loop* pada nilai-nilai di antara panjang array dan 1 (asumsi nilai mulai dari index 1), dan kurangi nilai *loop control variable* (*lcv*), agar *loop* tidak berjalan selamanya.
3. Pilih suatu nilai (*n*) antara 1 dan nilai *loop control variable*.
4. Tukar nilai array pada index *n* dan *lcv*. Ini dilakukan agar nilai yang ditukar tidak diperhitungkan lagi pada *loop* pengacakan selanjutnya.
5. Ulangi loop dari langkah 3 dengan nilai *lcv* yang sudah dikurangi.

2.3 Game

Menurut Salen dan Zimmerman (2003), sebuah *game* adalah sistem tempat pemain melakukan konflik buatan dengan berbagai peraturan, yang menghasilkan suatu hasil yang dapat diukur. Pernyataan ini dapat dibagi menjadi beberapa bagian.

1. Sistem

   Game sendiri terdiri dari berbagai bagian yang saling berkorelasi untuk membuat suatu persatuan kompleks.
2. Pemain

   Sebuah *game* memiliki satu atau lebih pemain, dan pemain berinteraksi dengan sistem dari sebuah game.
3. Buatan

   Game memiliki batasan dari kehidupan nyata. Meskipun *game* terjadi pada kehidupan nyata, fakta bahwa *game* hanya buatan merupakan salah satu fitur utama *game.*

4. Konflik

   Semua game memiliki sebuah “kontes” yang dapat memiliki berbagai bentuk, baik kooperasi maupun kompetisi, dari konflik solo hingga konflik *multiplayer*.

5. Peraturan

   Peraturan merupakan salah satu bagian terpenting pada *game*. Peraturan mendefinisikan apa yang dapat dan tidak dapat dilakukan pemain.

6. Hasil yang dapat diukur

   Game memilki suatu tujuan atau hasil. Pada akhir *game*, pemain telah menang atau kalah, atau mendapat suatu skor.

Menurut Lee (2014), terdapat beberapa jenis *genre* untuk *game*.

1. *Action: Game* dengan penekanan pada sejumlah aksi yang dilakukan pemain untuk mengikuti sekumpulan objektif.
2. *Action/ Adventure: Game* dengan dunia untuk dijelajahi oleh pemain dengan sekumpulan objektif untuk diselesaikan.
3. *Driving/ Racing:* Pada *genre game* ini terdapat sekumpulan kendaraan sebagai aksi utama, terkadang dengan tujuan untuk memenangkan suatu lomba.
4. *Fighting: Game* dengan pengontrolan suatu karakter *game* untuk melakukan perkelahian dengan suatu musuh.
5. *Puzzle: Game* dengan objektif untuk mencari solusi dengan menyelesaikan teka-teki.
6. *RPG: Game* dengan penekanan pada perkembangan karakter pemain dan komponen naratif.

7. *Shooter: Game* penembakan, dan cenderung menghancurkan, sekumpulan musuh atau objek.
8. *Simulation: Game* simulasi kegiatan dunia nyata pada dunia *game*.
9. *Sports: Game* simulasi suatu kegiatan olahraga pada dunia *game*.
10. *Strategy: Game* dengan fokus pada keputusan pemain untuk menghasilkan suatu hasil yang diinginkan.

2.4 Struktur Game

Pada suatu permainan, terdapat berbagai elemen yang membentuk permainan tersebut. Elemen ini ada dua jenis, yaitu elemen formal, yang membentuk struktur dari suatu permainan, serta elemen dramatis, yang berguna untuk memberikan konteks pada permainan, seta menggabungkan elemen formal menjadi suatu hal yang memikat pemain (Fullerton, 2014).

2.4.1 Formal Elements

Terdapat beberapa elemen formal pada *game* yang akan dijelaskan sebagai berikut (Fullerton, 2014).

1. *Players*

   Seorang pemain pada permainan adalah seseorang yang dengan sukarela mengikuti permainan tersebut. Pemain harus mengikuti aturan dan batasan pada permainan. Pemain membuat pilihan, aktif bermain, serta berpotensi menjadi pemenang.

2. *Objectives*

   Dengan adanya suatu *objective* pada permainan, maka pemain akan memiliki alasan untuk menyelesaikan permainan. *Objective* yang baik adalah *objective* yang menantang, tetapi masih mungkin untuk diselesaikan.

3. *Procedures*

   Procedure merupakan metode permainan dan tindakan yang dapat dilakukan pemain unutk mencapai *objective*.

4. *Rules*

   Rules menentukan objek pada permaian dan tindakan yang diperbolehkan untuk pemain.

5. *Resources*

   Resource merupakan asset yang dimiliki pemain yang dapat digunakan untuk menyelesaikan *objective* tertentu, seperti *health* atau *powerups.*

6. *Conflict*

   Pemain tidak dapat begitu saja menyelesaikan *objective* karena keterbatasan tindakan yang ditentukan oleh *rules*. Inilah yang menyebabkan *conflict*.

7. *Boundaries*

   Boundaries merupakan hal yang memisahkan permainan dari apapun yang bukan permainan. Misalnya dalam suatu lapangan futsal ada 13 orang, namun hanya 10 orang yang bermain sedangkan 3 orang sisanya hanya melihat.

8. *Outcome*

   Outcome merupakan hasil dari permainan, hal yang didapat setelah suatu permainan berakhir. *Outcome* dapat berupa skor seperti pada permainan *pinball*, atau poin menang atau kalah seperti pada permainan catur.

2.4.2 Dramatic Elements

Selain elemen formal, terdapat juga elemen dramatis, yang akan dijelaskan sebagai berikut (Fullerton, 2014).

1. *Challenge*

   Challenge merupakan tantangan bagi *player* pada suatu permainan. Tantangan ini haruslah berupa pekerjaan yang memberikan kepuasan apabila diselesaikan. Tantangan harus memiliki tingkat kesusahan yang tepat, agar tidak terlalu susah untuk diselesaikan dan tidak terlalu mudah sehingga membosankan.

2. *Play*

   Play dapat dianggap sebagi kebebasan pada dunia permainan. Terdapat aturan dan batasan pada permainan, dan *play* merupakan kebebasan permain untuk bertindak selama dalam aturan itu.

3. *Premise*

   Premise merupakan elemen drama tradisional pada permainan, seperti *setting* atau *story* dari suatu permainan. Tanpa *premise*, permainan akan terlalu abstrak bagi pemain.

4. *Characters*

   Karakter merupakan media interaksi pemain pada dunia permainan. Karakterlah yang akan berperan dalam suatu *story* pada permainan, dan akan melakukan tindakan sesuai perintah pemain.

5. *Story*

   Story merupakan cerita yang akan diceritakan kepada pemain dari sebuah *game*. Sebuah cerita yang menarik akan mengikat pemain, dimana pemain ingin menyelesaikan permainan demi mengetahui *outcome* dari sebuah *story*.

6. *World Building*

   World Building merupakan desain dari dunia fiksi yang diciptakan oleh game, baik berupa peta ataupun sejarah dunia tersebut.

2.5 Motivasi Pembelajaran

Motivasi merupakan salah satu elemen penting pada pembelajaran (Keller & Kopp, 1987) karena dapat menstimulasi rasa ingin tahu mahasiswa, mendorong pembelajaran baik secara individu maupun kelompok (Dillenbourg et al. 2009). Permainan mendukung motivasi intrinsik, motivasi yang dianggap memiliki pengaruh besar pada pelajar (Habgood et al., 2005), sehingga permainan dapat digunakan sebagai alat pembelajaran.

Menurut Sardiman (2005), ada beberapa bentuk dan cara untuk menumbuhkan motivasi dalam kegiatan belajar.

1. Memberi Angka

   Merupakan symbol dari kegiatan belajar, banyak siswa belajar hanya demi mendapat angka/nilai yang baik

2. Hadiah

   Hadiah dapat digunakan sebagai pendorong motivasi, namun mungkin tidak menarik bagi pelajar yang tidak tertarik pada suatu pekerjaan.

3. Saingan

   Persaingan dapat digunakan untuk sebagai motivasi, baik persaingan individu atau kelompok

4. Keterlibatan diri

   Menumbuhkan kesadaran pada siswa atas pentingnya tugas dan menerimanya sebagai tantangan.

5. Memberi Ulangan

   Apabila mengetahui sebelumnya bahwa akan dilakukan ulangan, siswa akan terdorong untuk belajar.

6. Mengetahui Hasil

   Apabila pelajar mengetahui hasil dari pekerjaannya terutama jika mengetahui ia mengalami kemajuan, pelajar dapat terdorong untuk semakin giat belajar.

7. Pujian

   Memberi pujian sebagai hadiah positif sekaligus memberikan motivasi.

8. Hukuman

   Merupakan sebuah hadiah negatif, namun apabila diberikan dengan tepat dapat menjadi alat motivasi.

9. Hasrat untuk belajar

   Unsur kesengajaan dari pelajar langsung untuk belajar.

10. Minat

    Motivasi muncul karena adanya kebutuhan, begitu juga minat sehingga apabila jika siswa sudah memiliki minat, proses belajar akan berjalan lancar.

11. Tujuan yang diakui

    Apabila pelajar memahami tujuan yang harus dicapai, karena dirasa berguna untuk dirinya, maka pelajar akan terdorong untuk belajar.

2.6 Game Based Learning

Game mendukung motivasi intrinsik sehingga *game* dapat digunakan untuk keperluan pembelajaran (Habgood dkk, 2005). Felicia (2011) menyatakan bahwa ketika pemain melakukan aktivitas yang memotivasi secara intrinsik, mereka lebih tertarik untuk belajar lebih dalam, dan menggunakan pengetahuan yang didapat pada dunia nyata, seperti sekolah. *Game* dapat menjadi lingkungan pembelajaran ideal karena *game* pada dasarnya mengajak pemain untuk belajar dan

mengembangkan kemampuan baru untuk menyelesaikan masalah pada *game*. Menurut Roussou (2004), pada *learning through play* diperlukan keseimbangan antara *fun* dan pekerjaan. *Game* harus memiliki lingkungan yang *fun* untuk belajar, namun juga harus memiliki tugas pembelajaran yang bermakna, agar pelajar mampu belajar serius dan menyelesaikan tugas sulit.

Menurut Parvini dan Saber (2011), algoritma sendiri merupakan suatu pembelajaran konsep abstrak, karena itu sulit untuk diajarkan karena membutuhkan banyak penjelasan dan ilustrasi. Karena itu *game* dapat memberikan bantuan pengajaran algoritma karena *game* dapat memberi banyak visualisasi algortima untuk mempermudah pelajar mempelajari dan mengerti algoritma.

2.7 Metode Penelitian Kuantitatif

Metode penelitian kuantitatif digunakan untuk menguji sebuah hipotesis terhadap sebuah populasi. Metode penelitian ini dilakukan untuk memperoleh data kuantitatif atau statistik yang diteliti dari populasi atau sampel tersebut dan dianalisa. Salah satu metode penelitian kuantitatif adalah metode survei (Sugiyono, 2010).

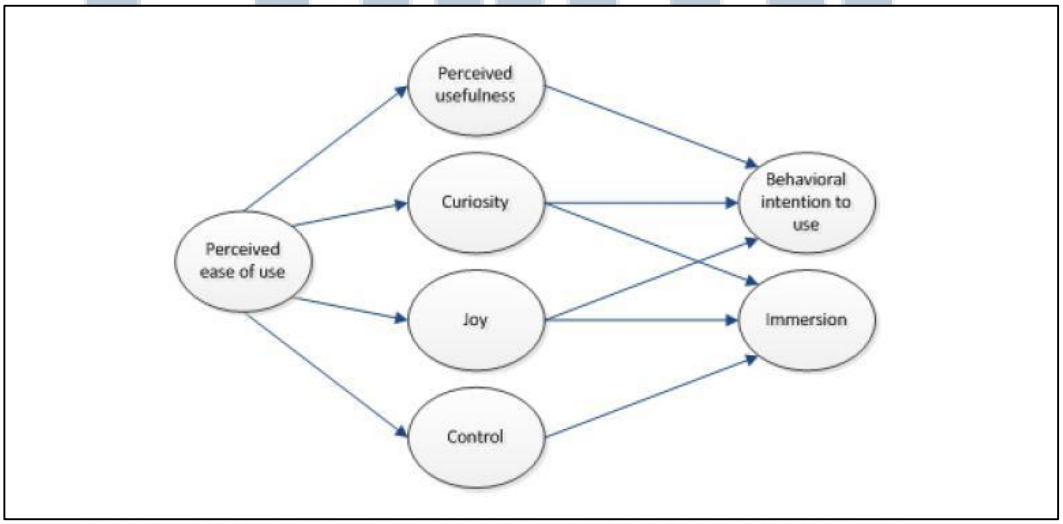
Kerlinger (2006) menyatakan bahwa metode survei dapat dilakukan pada populasi besar maupun kecil, tetapi data yang dipelajari adalah data dari sampel yang diambil dari populasi tersebut, sehingga ditemukan kejadian-kejadian relatif, distribusi, dan hubungan-hubungan antar variabel sosiologis maupun psikologis. Berdasarkan kedua pernyataan di atas, metode penelitian kuantitatif dengan metode survei cocok untuk penelitian mengenai variabel psikologis dari sampel populasi tertentu tanpa terlalu mempermasalahkan ukuran sampel.

2.8 Purposive Sampling

Menurut Sugiyono (2010) *purposive sampling* adalah teknik pengambilan sampel sumber data dengan pertimbangan tertentu. Latham (2007) menyatakan bahwa sampel dipilih dengan basis pengetahuan peneliti mengenai populasi, elemen, dan tujuan penelitian. Peneliti kemudian dapat memilih penguji coba dipilih berdasarkan pengetahuan internal terhadap karakteristik penelitian. Sampel yang tidak memenuhi kriteria yang telah ditentukan, akan berdampak tidak baik pada hasil penelitian yang dilakukan.

2.9 HMSAM (Hedonic Motivation System Adoption Model)

HMSAM adalah sebuah model untuk mengukur motivasi intrinsik dari suatu sistem yang mengadaptasi motivasi hedonis. HMSAM merupakan sebuah model pengukuran yang berdasarkan pada Hedonic-Motivation System (HMS). sendiri merupakan sistem yang biasanya digunakan untuk memenuhi kebutuhan intrinsik pengguna yang didasar pada sifat hedonisme (Lowry dkk, 2013). Ilustrasi untuk model ini dapat dilihat pada Gambar 2.10.



Gambar 2.10 HMSAM Model
(Lowry dkk., 2013)

Berdasarkan Gambar 2.10 terdapat 5 faktor pengukuran HMSAM.

1. *Perceived Usefulness*, yaitu pengukuran kinerja penggunaan suatu sistem
2. *Perceived Ease of Use*, yaitu pengukuran kemudahan penggunaan sistem
3. *Control*, yaitu pengukuran persepsi pengguna seakan pengguna berinteraksi dengan sistem secara langsung
4. *Joy*, yaitu kesenangan yang didapat dari interaksi antar pengguna dan sistem
5. *Curiosity*, yaitu pengukuran rasa ingin tahu pengguna.

Lima faktor ini akan mempengaruhi 2 aspek utama yang akan dinilai, yaitu *behavioral intention to use* dan *immersion* dari *game*. *Immersion* merupakan tingkat intensitas pengguna merasa *engaged* dengan *game*, sehingga kebutuhan lain dapat dianggap tidak diperlukan untuk beberapa saat. *Behavioral Intention To Use* merupakan tingkat niat pemain memainkan *game* lagi setelah memainkannya pertama kali (Lowry dkk, 2013). Kedua aspek ini merupakan aspek utama yang dinilai oleh HMSAM. Berdasarkan Gambar 2.10, nilai 2 aspek ini dapat ditemukan dengan Rumus 2.1 dan 2.2.

$$BEOU = \frac{(PEOU) + (PU) + (C)}{n\ of\ aspect} \quad \ldots(2.1)$$

$$I = \frac{(PEOU) + (C) + (Co)}{n\ of\ aspect} \quad \ldots(2.2)$$

Dengan keterangan sebagai berikut.

1. *BEOU* adalah *behavioral intention to use*
2. *I* adalah *immersion*
3. *PEOU* adalah *Perceived Ease Of Use*
4. *PU* adalah *Perceived Usefulness*
5. *C* adalah *Curiosity*
6. *Co* adalah *Control*

7. *n of aspect* adalah jumlah aspek yang menjadi faktor penilaian.

2.10 Likert Scale

Data yang didapat akan diolah secara statistik, karena itu diperlukan suatu metode untuk menyusun hasil pengumpulan data menjadi suatu data statistik. Menurut Sugiyono (2010), skala Likert merupakan sebuah metode untuk mengukur data kualitatif menjadi kuantitatif. Skala likert mempunyai sejumlah pilihan yang berisikan suatu pernyataan dimana masing-masing pernyataan tersebut mewakili sebuah nilai. Himpunan pilihan Likert harus bersifat simetris dan imbang dengan rentang pilihan yang bersifat negatif maupun positif (Uebersax, 2006) serta disertai dengan skor tertentu yang menghasilkan nilai tertentu yang dapat digunakan untuk pengolahan data (Likert, 1932).

Likert Scale dengan tujuh kriteria memiliki pembagian bobot sebagai berikut.

1. Kategori "Strongly Agree" diberikan bobot 7.
2. Kategori "Agree" diberikan bobot 6.
3. Kategori "Agree Somewhat" diberikan bobot 5.
4. Kategori "Neutral" diberikan bobot 4.
5. Kategori "Disagree Somewhat" diberikan bobot 3.
6. Kategori "Disagree" diberikan bobot 2.
7. Kategori "Strongly Disagree" diberikan bobot 1.

Kriteria interpretasi skor yang digunakan untuk mengolah jawaban dari Likert Scale adalah sebagai berikut dengan 'X' sebagai nilai.

1. "Strongly Agree" dengan persyaratan $X \geq 86\%$
2. "Agree" dengan persyaratan $72\% \leq X < 86\%$

3. “Agree Somewhat” dengan persyaratan 58% ≤ X < 72%
4. “Neutral” dengan persyaratan 43% ≤ X < 58%
5. “Disagree Somewhat” dengan persyaratan 29% ≤ X < 43%
6. “Disagree” dengan persyaratan 14%≤ X < 29%
7. “Strongly Disagree” dengan persyaratan X < 14%

Menurut Boone dan Boone (2012), cara menghitung nilai dari *likert scale* dapat dilakukan dengan menggunakan skor komposit dari *item* pada Likert Scale, berupa *mean*. Rumus *mean* yang digunakan untuk menghitung skor suatu kriteria dapat dilihat pada Rumus 2.3.

$$X = \frac{(SA * nSA) + (A * nA) + (AS * nAS) + (N * nN) + (DS * nDS) + (D * nD) + (SD * nSD)}{nKriteria * nSampel} \quad \ldots(2.3)$$

Dengan Keterangan

1. *SA* merupakan nilai *Strongly Agree* dan *nSA* adalah jumlah jawaban *Strongly Agree*
2. *A* merupakan nilai *Agree* dan *nA* adalah jumlah jawaban *Agree*
3. *AS* merupakan nilai *Agree Somewhat* dan *nAS* adalah jumlah jawaban *Agree Somewhat*
4. *N* merupakan nilai *Neutral* dan *nN* adalah jumlah jawaban *Neutral*
5. *DS* merupakan nilai *Disagree Somewhat* dan *nDS* adalah jumlah jawaban *Disagree Somewhat*
6. *D* merupakan nilai *Disagree* dan *nD* adalah jumlah jawaban *Disagree*
7. *SD* merupakan nilai *Strongly Disagree* dan *nSD* adalah jumlah jawaban *Strongly Disagree*
8. nKriteria adalah jumlah kriteria interpretasi skor
9. nSampel adalah jumlah sampel.

2.11 Cronbach's Alpha

Untuk mengukur korelasi antara pertanyaan-pertanyaan pada survei dapat digunakan metode Cronbach's Alpha (Gliem dan Gliem, 2003). Cronbach's Alpha merupakan metode pengukuran hubungan antar *item* atau skala pada sebuah grup (GoForth, 2015). Hasil perhitungan yang disebut *alpha* menentukan tingkat kepercayaan survey berdasarkan skala Likert.

Jarak menentukan korelasi berdasarkan skala Likert umumnya sebagai berikut (George dan Mallery, 2003).

1. $\alpha \geq 0.9$ berarti sangat bagus,
2. $0.9 > \alpha \geq 0.8$ berarti bagus,
3. $0.8 > \alpha \geq 0.7$ berarti dapat diterima,
4. $0.7 > \alpha \geq 0.6$ berarti dapat dipertanyakan,
5. $0.6 > \alpha \geq 0.5$ berarti buruk, dan
6. $0.5 > \alpha$ berarti tidak dapat diterima.

Rumus untuk Cronbach's Alpha dapat dilihat pada Rumus 2.4 (GoForth, 2015).

$$\alpha=\left(\frac{k}{k-1}\right)\left(1-\frac{\sum_{i=1}^{k}\sigma_{y_i}^2}{\sigma_x^2}\right) \quad \ldots(2.4)$$

Dengan keterangan:

1. *k* adalah jumlah pertanyaan
2. $\sigma_{y_i}^2$ adalah *variance* dari item ke-i
3. σ_x^2 adalah *variance* dari keseluruhan survei.

Berdasarkan rumus diatas dapat dilihat bahwa *variance* yang tinggi memberikan nilai *alpha* yang tinggi, dan sebaliknya. Nilai *variance* yang tinggi sendiri artinya nilai-nilai yang didapat memiliki ragam yang banyak, sedangkan

nilai *variance* yang rendah artinya nilai yang didapat tidak terlalu beragam. Terkecuali jika peserta survei memiliki sifat yang tidak jauh berbeda, tes yang dilakukan kurang berguna. Pertanyaan dengan nilai *alpha* rendah cenderung memiliki korelasi antar *item* yang rendah. (Cronbach, 1951).

2.12 Unity

Unity3D merupakan sebuah *game engine* yang bersifat *cross-platform*. Mudah digunakan untuk pemula dan memiliki banyak peralatan untuk *expert*, dokumentasi lengkap dari *Unity Technologies*, serta komunitas *developer* yang sangat suportif membuat *Unity* banyak diminati. Sifatnya yang *cross-platform* juga membuat *game* dapat dibuat di banyak *platform*, baik *Windows* maupun *Android*. Tidak hanya itu, *Unity* juga mendukung lebih dari satu bahasa pemrograman, yaitu C#, Javascript dan Boo (Zomacj, 2012).