

SKRIPSI

**PERBANDINGAN AGORITMA *HORSPPOOL* DAN ALGORITMA *RAITA*
PADA APLIKASI KAMUS ISTILAH PSIKOLOGI BERBASIS *ANDROID***

Diajukan Untuk Memenuhi
Salah Satu Syarat Memperoleh Gelar Sarjana Teknik



MAMTA CULKARI. P

E1E1 15 026

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS HALU OLEO

KENDARI

2019

BAB I

PENDAHULUAN

1.1 Latar Belakang

Kamus merupakan sumber rujukan yang membuat kosakata dan penjelasan maknanya, kamus bertujuan untuk menyediakan makna yang tepat bagi kata yang dicari oleh pengguna (Mutiawani, 2011). Pada umumnya kamus berbentuk buku cetak yang penggunaannya menyulitkan, karena pengguna harus mencari arti dan istilah secara manual. Disisi lain kamus yang tebal dan berat menyulitkan dibawa kemana-mana dan proses pencarian istilah yang diinginkan memakan waktu yang cukup lama.

Mengatasi masalah tersebut dibutuhkan suatu teknologi yang dapat digunakan untuk menggantikan kamus berbentuk buku yang penggunaannya masih secara manual. Salah satu cara yang dapat digunakan untuk mengatasi hal tersebut adalah dibutuhkan sebuah aplikasi *smartphone* yang dapat membantu dalam mencari kata atau istilah, yaitu kamus digital berbasis *android*.

Banyaknya penderita gangguan jiwa yang belum tertangani secara medis, dikarenakan masih minimnya tenaga kesehatan jiwa profesional di Indonesia. Hal tersebut tentunya semakin menghambat upaya pencegahan dan penanganan persoalan kesehatan jiwa masyarakat. Jumlah tenaga kesehatan jiwa profesional di Indonesia masih belum mampu memenuhi kuota minimal yang telah ditetapkan oleh Organisasi Kesehatan Dunia atau WHO. Saat ini Indonesia dengan penduduk sekitar 250 juta jiwa baru memiliki sekitar 451 psikolog klinis (0,15 per 100.000 penduduk), 773 psikiater (0,32 per 100.000 penduduk), dan perawat jiwa 6.500 orang (2 per 100.000 penduduk) (Kementrian Kesehatan RI, 2013). WHO telah menetapkan standar jumlah tenaga psikolog dan psikiater dengan jumlah penduduk adalah 1:30 ribu orang, atau 0,03 per 100.000 penduduk. Dari data tersebut maka perlu adanya suatu kamus digital untuk memudahkan para tenaga kesehatan jiwa khususnya mahasiswa psikologi untuk mempelajari istilah-istilah dalam ilmu psikologi.

Pembuatan kamus istilah dalam bentuk teknologi digital dapat diimplementasikan dengan menggunakan metode pencocokan *string*. Pencocokan *string* banyak digunakan dalam aplikasi pengolahan teks untuk pencarian kata dalam berkas teks. Pencarian *string* di dalam teks disebut juga *string matching* (Ryan Vidho Valba, 2012). Adapun Algoritma yang digunakan yaitu Algoritma *Horspool* dan Algoritma *Raita*. Dalam penelitian ini penulis akan membandingkan kedua metode tersebut.

Dalam Penelitian yang dilakukan oleh Adhi Kusnadi (2017) menyimpulkan bahwa Algoritma *Horspool* lebih cepat daripada Algoritma *Zhu-Takaoka*. Algoritma *Horspool* lebih cepat 19.82845% pada uji coba pertama menggunakan 50 *file text file* kelipatan 1000 kata dengan *pattern* yang sama dan 15.9442% pada ujicoba kedua dengan *file text* 7000 kata. Selanjutnya penelitian dilakukan Suwitri (2017) menyimpulkan bahwa Algoritma *Raita* melakukan pencarian *string* lebih cepat dibandingkan Algoritma *Quick Search* dengan total rata-rata *running time* untuk Algoritma *Raita* adalah 0,0126 *ms* dan Algoritma *Quick Search* adalah 9,7261 *ms*, sehingga dianggap sangat tepat untuk membandingkan kedua algoritma ini. Metode penyelesaian yang berbeda dari kedua metode ini juga dianggap akan menghasilkan perbedaan.

Berdasarkan uraian tersebut, maka penulis mengambil judul tugas akhir **“Perbandingan Algoritma *Horspool* dan Algoritma *Raita* Pada Aplikasi Kamus Istilah Psikologi Berbasis *Android*”** untuk mengetahui perbandingan Algoritma *string matching* yang lebih baik antara Algoritma *Horspool* dan Algoritma *Raita* pada aplikasi kamus istilah psikologi. Aplikasi ini juga diharapkan dapat membantu para mahasiswa jurusan psikologi maupun para psikolog dalam memahami arti istilah psikologi dengan mudah dan cepat yang membantu kegiatan mereka sehari-hari.

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, maka rumusan masalah pada penelitian ini adalah membandingkan kecepatan waktu proses dan kompleksitas dari

Algoritma *Horspool* dan Algoritma *Raita* dalam pencarian kata istilah pada Aplikasi Kamus Istilah Psikologi berbasis *Android*.

1.3 Batasan Masalah

Batasan masalah yang ditetapkan dalam penelitian ini adalah sebagai berikut:

1. Menggunakan algoritma pencarian yaitu Algoritma *Horspool* dan Algoritma *Raita* sebagai metode pencarian kata yang terbatas pada istilah-istilah psikologi.
2. Aplikasi ini dapat dijalankan pada perangkat *Android 4.5* ke atas.
3. Tidak semua istilah psikologi dimasukkan sebagai data, dibatasi hanya sampai 500 kata.
4. Data istilah di simpan dalam *file Txt*.
5. Parameter pembandingnya adalah waktu (*ms*) dan kompleksitas algoritma (*Big O*).

1.4 Tujuan

Tujuan dari penelitian ini adalah untuk melakukan perbandingan kecepatan waktu dan menganalisis kompleksitas algoritma yaitu Algoritma *Horspool* dan Algoritma *Raita* pada Aplikasi Kamus Istilah Psikologi berbasis *Android* sehingga diketahui algoritma yang baik untuk digunakan dalam pencarian istilah kata psikologi.

1.5 Manfaat

Manfaat dari penelitian ini adalah sebagai berikut :

1. Mengetahui perbandingan Algoritma *Horspool* dan Algoritma *Raita* dalam hal kecepatan pencarian kata.
2. Menganalisis kompleksitas Algoritma *Horspool* dan Algoritma *Raita*.
3. Membantu pengguna *smartphone* untuk mencari istilah dalam psikologi.

4. Penelitian ini juga diharapkan dapat menjadi acuan tambahan bagi penulis maupun peneliti selanjutnya dalam membandingkan Algoritma *Horspool* dan Algoritma *Raita* dalam hal pencarian kata.

1.6 Sistematika Penulisan

Sistematika penulisan tugas akhir ini terdiri atas beberapa bagian utama sebagai berikut:

BAB I PENDAHULUAN

Merupakan bab pendahuluan yang menguraikan latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan sistematika penulisan.

BAB II LANDASAN TEORI

Membahas mengenai dasar-dasar teori pendukung yang berhubungan dengan masalah yang diambil dan program aplikasi yang akan digunakan dalam pembangunan sistem.

BAB III METODE PENELITIAN

Membahas mengenai waktu dan tempat penelitian, metode pengumpulan data, metode pembangunan sistem yang digunakan dalam penelitian tugas akhir ini.

BAB IV ANALISIS DAN PERANCANGAN

Bab ini akan membahas tentang analisis dan perancangan dari aplikasi yang akan dibuat, dengan menggunakan desain UML (*Unified Modelling Language*) serta desain *user interface*.

BAB V IMPLEMENTASI DAN PEMBAHASAN

Dalam bab ini akan dikaji mengenai implementasi hasil perancangan aplikasi yang dibuat serta melakukan pengujian terhadap sistem.

BAB VI PENUTUP

Bab ini berisi kesimpulan dari program yang telah dibuat serta saran yang diperlukan untuk pengembangan program berikutnya.

1.7 Tinjauan Pustaka

Penelitian ini didasarkan pada penelitian sebelumnya mengenai aplikasi kamus berbasis *mobile Android*. Penelitian yang dilakukan oleh Dewi Dessy Suprpti (2017) dalam penelitiannya yang berjudul Kamus Penyakit Dan Tanaman Obat Menggunakan Algoritma *Sequential Search With Sentinel* Berbasis *Android* menyimpulkan algoritma *Sequential Search with Sentinel* dapat diimplementasikan dengan baik ke dalam aplikasi kamus penyakit dan tanaman obat berbasis *Android*.

Penelitian yang dilakukan oleh Irna Rahayu (2015) dalam penelitiannya yang berjudul Implementasi Kamus Kedokteran dengan Metode Interpolasi (*Interpolation*) dan Mencari Kemiripan Kata Menggunakan Algoritma *Lavenshtein Distance* Pada Perangkat *Android* dapat diimplementasikan ke dalam aplikasi kamus kedokteran berbasis *Android*. Aplikasi ini dapat menambah kata, menghapus kata, dan mengubah kata yang sudah ada.

Penelitian yang dilakukan oleh Muhammad Hatta (2015) dalam penelitiannya yang berjudul Implementasi Kamus Kata Serapan Bahasa Indonesia – Inggris Dengan Fitur *Auto Complete Text* Menggunakan Algoritma *Interpolation Search* Untuk *Smartphone* Berbasis *Android* dapat diimplementasikan dengan baik. Aplikasi ini dapat membantu pengguna yang ingin belajar ataupun ingin memahami bahasa Indonesia yang baik dan benar serta ingin mengetahui perbedaan ucapan antara bahasa Indonesia dan bahasa Inggris. Pencarian data ini menggunakan metode *Interpolation Search* secara teori yaitu menghitung dan menentukan posisi *index*, maka diperoleh hasil dengan *index* yang sama antara teori dengan pengujian secara praktik menggunakan program yang telah dibuat.

Penelitian yang dilakukan oleh Annisyah Januarti (2017) dalam penelitiannya yang berjudul Aplikasi Istilah Akuntansi Sebagai Media Pembelajaran Berbasis *Android* Menggunakan Algoritma *Reverse Colussi* menyimpulkan bahwa Algoritma *Reverse Colussi* berhasil diimplementasikan pada Aplikasi Istilah Akuntansi berbasis *Android*. Algoritma *Reverse Colussi* sangat efisien karena rata-rata waktu pencarian paling lambat kurang dari 2 detik, dan tepat dalam melakukan pencarian *string*, selama istilah yang dicari ada dalam *database*.

Dari beberapa penelitian sebelumnya penelitian hanya menggunakan satu metode dalam penerapan pencarian kata, sedangkan penelitian yang akan dilakukan penulis akan membuat aplikasi kamus dengan menerapkan dua metode *string matching* dalam penerapan pencarian kata dan akan membandingkan kedua metode tersebut dengan parameter pembandingnya adalah waktu (*ms*) dan kompleksitas Algoritma (*Big O*).

BAB II

LANDASAN TEORI

2.1 Kamus Istilah

Menurut Kamus Besar Bahasa Indonesia, Kamus Istilah (*Glosarium*) adalah kamus dalam bentuk yang ringkas dengan daftar kata beserta penjelasannya dalam bidang tertentu. *Glosarium* adalah suatu daftar alfabetis istilah dalam suatu ranah pengetahuan tertentu yang dilengkapi dengan definisi untuk istilah-istilah tersebut. Biasanya *glosarium* ada di bagian akhir suatu buku dan menyertakan istilah-istilah dalam buku tersebut yang baru diperkenalkan atau tidak umum ditemukan. *Glosarium* dwibahasa adalah daftar istilah dalam satu bahasa yang didefinisikan dalam bahasa lain atau diberi sinonim (atau paling tidak sinonim terdekat) dalam bahasa lain.

Menurut Badan Pengembangan dan Pendidikan Bahasa Kementrian Pendidikan dan Kebudayaan (2010), *Glosarium* atau kamus istilah merupakan sumber pengayaan pengetahuan tentang padanan istilah bidang ilmu yang dapat memperkaya perbendaharaan pustaka kebahasaan. Sebagai rujukan, *glosarium* memuat kumpulan istilah bidang ilmu dalam bahasa asing (Inggris) sebagai entri beserta padanannya dalam bahasa Indonesia. Seperti halnya dalam penyusunan kamus, dalam penyusunan *glosarium* istilah bidang ilmu yang berasal dari istilah asing disusun secara alfabetis. Istilah asing itu kemudian diberi penjelasan berupa padanannya dalam Bahasa Indonesia.

2.2 Psikologi

Psikologi berasal dari bahasa Yunani yaitu *psyche* yang artinya jiwa dan *logos* artinya ilmu. Jadi secara etimologi psikologi artinya ilmu yang mempelajari jiwa, baik mengenai macam-macam gejalanya, proses maupun latar belakangnya. Jiwa secara harfiah berasal dari perkataan sansekerta JIV, yang berarti lembaga hidup (*levenbeginsel*), atau daya hidup (*levenscracht*). Oleh karena itu jiwa merupakan pengertian yang abstrak, tidak bisa dilihat dan belum bisa diungkapkan secara lengkap dan jelas, maka orang lebih cenderung mempelajari “jiwa yang

memateri” atau gejala “jiwa yang meraga/menjasmani”, yaitu bentuk tingkah laku manusia (segala aktivitas, perbuatan, penampilan diri) sepanjang hidupnya (Danarjati, dkk. 2010).

Psikologi adalah ilmu pengetahuan yang mempelajari perilaku manusia dalam hubungan dengan lingkungannya. Menurut asalnya katanya, psikologi berasal dari bahasa Yunani Kuno; "ψυχή" (*Psychē* yang berarti jiwa) dan "-λογία" (-logia yang artinya ilmu) sehingga secara etimologis, psikologi dapat diartikan dengan ilmu yang mempelajari tentang jiwa. Oleh karena itu, psikologi butuh berabad-abad lamanya untuk memisahkan diri dari ilmu filsafat (Syah, 2003). *Motorola, Qualcomm, T-Mobile, NVIDIA* yang tergabung dalam OHA (*Open Handset Alliance*) dengan tujuan membuat sebuah standar terbuka untuk perangkat bergerak (*Mobile Device*) (Mulyadi, 2010).

2.3 Pencarian (*Searching*)

Pencarian atau *searching* merupakan tindakan untuk mendapatkan suatu data dalam kumpulan data. Pencarian seringkali ditemukan di dalam kehidupan sehari-hari, misalnya untuk menemukan nomor telepon seseorang pada buku telepon atau mencari suatu istilah dalam kamus. Pada aplikasi komputer, pencarian biasanya dilakukan untuk mendapatkan data dari seorang mahasiswa, mendapatkan informasi suatu kata dalam kamus digital, mendapatkan nomor telepon berdasarkan suatu alamat atau nama perusahaan.

Algoritma pencarian atau *searching algorithm* adalah algoritma yang menerima sebuah argumen kunci dengan langkah-langkah tertentu akan mencari rekaman dengan kunci tersebut. Setelah proses pencarian dilaksanakan, akan diperoleh salah satu dari dua kemungkinan, yaitu data yang dicari ditemukan (*successful*) atau tidak ditemukan (*unsuccessful*) (Sembiring, 2013).

2.4 Algoritma *Horspool*

Algoritma *Horspool* adalah penyederhanaan dari Algoritma *Boyer-Moore* yang dibuat oleh R. Nigel Horspool. Menurut Horspool, R.N.(1980), Algoritma *Horspool* bekerja dengan metode yang hampir sama dengan algoritma *Boyer-*

Moore namun tidak melakukan lompatan berdasarkan karakter pada *pattern* yang ditemukan tidak cocok pada teks. R. Nigel Horspool melakukan penelitian tentang Algoritma *Boyer-Moore*. Dari hasil penelitiannya beliau mengemukakan gagasan tambahan untuk Algoritma *Boyer-Moore*. Algoritma ini dikenal dengan Algoritma *Horspool* atau Algoritma *Boyer-Moore-Horspool* (Tambun 2010).

Algoritma *Horspool* mempunyai nilai pergeseran karakter yang paling kanan dari *window*. Pada tahap *observasi* awal (*preprocessing*), nilai *shift* akan dihitung untuk semua karakter. Pada tahap ini, dibandingkan *pattern* dari kanan ke kiri hingga kecocokan atau ketidakcocokan *pattern* terjadi. Karakter yang paling kanan pada *window* digunakan sebagai indeks dalam melakukan nilai *shift*. Dalam kasus ketidakcocokan (karakter tidak terdapat pada *pattern*) terjadi, *window* digeser oleh panjang dari sebuah *pattern*. Jika tidak, *window* digeser menurut karakter yang paling kanan pada *pattern* (Baeza-Yates & Regnier 1992). Terdapat dua tahap pada pencocokan *string* menggunakan Algoritma *Horspool* (Singh & Verma 2011), yaitu:

1. Tahap praproses

Pada tahap ini, dilakukan *observasi pattern* terhadap teks untuk membangun sebuah tabel *bad-match* yang berisi nilai *shift* ketika ketidakcocokan antara *pattern* dan teks terjadi. Secara sistematis, langkah-langkah yang dilakukan Algoritma *Horspool* pada tahap praproses adalah:

- a. Algoritma *Horspool* melakukan pencocokan karakter ter-kanan *pattern*.
- b. Setiap karakter pada *pattern* ditambah ke dalam Tabel *BmBc* dan dihitung nilai *shift*-nya.
- c. Karakter yang berada pada ujung *pattern* tidak dihitung dan tidak dijadikan karakter ter-kanan dari karakter yang sama dengannya.
- d. Apabila terdapat dua karakter yang sama dan salah satunya bukan karakter terkanan, maka karakter dengan indeks terbesar yang dihitung nilai *shift*-nya.
- e. Algoritma *Horspool* menyimpan panjang dari *pattern* sebagai panjang nilai *shift* secara *default* apabila karakter pada teks tidak ditemukan dalam *pattern*.

- f. Nilai (BM) *shift* yang akan digunakan dapat dicari dengan perhitungan panjang, dari *pattern* dikurang indeks terakhir karakter dikurang 1, untuk masing-masing karakter (BC) , $BM = m - i - 1$.

Sebagai contoh, dapat dilihat pada tabel 2.1 berikut:

Pattern : PUDING

P U D I N G

 0 1 2 3 4 5

Tabel 2.1 Tabel *BmBc* pada praproses

BC	BM
P	5
U	4
D	3
I	2
N	1
*	6

Untuk mencari nilai geser pada Tabel *BmBc*, digunakan rumus :

$$BM[i] = m - i - 1 \dots \dots \dots (2.1)$$

$$BM [0] = 6 - 0 - 1 = 5$$

$$BM [1] = 6 - 1 - 1 = 4$$

$$BM [2] = 6 - 2 - 1 = 3$$

$$BM [3] = 6 - 3 - 1 = 2$$

$$BM [4] = 6 - 4 - 1 = 1$$

(*) : karakter yang tidak dikenali.

2. Tahap Pencarian

Secara sistematis, langkah-langkah yang dilakukan Algoritma *Horspool* pada tahap pencarian adalah:

- Dilakukan perbandingan karakter paling kanan *pattern* terhadap *window*.
- Tabel *BmBc* digunakan untuk melewati karakter ketika ketidakcocokan terjadi.
- Ketika ada ketidakcocokan, maka karakter paling kanan pada *window* berfungsi sebagai landasan untuk menentukan jarak *shift* yang akan dilakukan.
- Setelah melakukan pencocokan (baik hasilnya cocok atau tidak cocok) dilakukan pergeseran ke kanan pada *window*.
- Prosedur ini dilakukan berulang-ulang sampai *window* berada pada akhir teks atau ketika *pattern* cocok dengan teks.

Sebagai contoh dalam mendeskripsikan Algoritma *Hoorspol* diberikan teks dan *pattern* sebagai berikut:

Teks = **MAMTA CULKARI PUDING**

Pattern = **PUDING**

Inisialisasi awal dan pembuatan *BmBc* terlihat pada Tabel 2.2 dan Tabel 2.3 berikut :

Tabel 2.2 Inisialisasi awal *BmBc*

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P	P	U	D	I	N	G														
I	0	1	2	3	4	5														

Tabel 2.3 Pembuatan *BmBc*

P	P	U	D	I	N	*
I	0	1	2	3	4	-
S	5	4	3	2	1	6

Dilihat dari Tabel 2.2, inisialisasi awal *BmBc* dilakukan. Setiap teks dan *pattern* masing masing diberi nilai m dan i , dimana m adalah panjang *pattern* dan i adalah indeks. Tabel 2.3 menunjukkan nilai pergeseran *BmBc* dengan menghitung nilai *BM* seperti yang telah dilakukan pada Tabel 2.1. Pada tahap awal pencarian, dilakukan perbandingan karakter paling kanan *pattern* terhadap *window*. Apabila terjadi ketidakcocokan maka akan dilakukan pergeseran ke kanan untuk melewati karakter yang tidak cocok di mana nilai pergeserannya terdapat pada Tabel *BmBc*. Karakter paling kanan teks pada *window* berfungsi sebagai landasan untuk menentukan jarak geser yang akan dilakukan. Hal ini terlihat pada Tabel 2.4 berikut :

Tabel 2.4 Iterasi Algoritma Horspool pertama

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P	P	U	D	I	N	G														
I	0	1	2	3	4	5														

Pada Tabel 2.4 terdapat ketidakcocokan antara karakter “Spasi” dan “G”. Karakter “Spasi” tidak terdapat pada Tabel *BmBc* sehingga digantikan oleh tanda (*). Tanda (*) bernilai sebesar 6 sehingga dilakukan pergeseran sebanyak 6 kali. Hal ini terlihat pada Tabel 2.5.

Tabel 2.5 Iterasi Algoritma Horspool kedua

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P							P	U	D	I	N	G								
I							0	1	2	3	4	5								

Pada Tabel 2.5 terdapat ketidakcocokan kembali antara karakter “R” dan “G”. Karakter “R” tidak terdapat pada Tabel *BmBc* sehingga digantikan oleh tanda (*). Tanda (*) bernilai sebesar 6 sehingga dilakukan pergeseran sebanyak 6 kali. Hal ini terlihat pada Tabel 2.6.

Tabel 2.6 Iterasi Algoritma *Horspool* ketiga

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P													P	U	D	I	N	G		
I													0	1	2	3	4	5		

Pada Tabel 2.6 terdapat ketidakcocokan kembali antara karakter “I” dan “G”. Karakter “I” terdapat pada Tabel *BmBc* yang bernilai sebesar 2 sehingga dilakukan pergeseran sebanyak 2 kali. Hal ini terlihat pada Tabel 2.7.

Tabel 2.7 Iterasi Algoritma *Horspool* keempat

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P															P	U	D	I	N	G
I															0	1	2	3	4	5

Pada Tabel 2.7 terdapat kecocokan pada karakter “G” dan “G”, karena karakter cocok, maka pencocokan dilanjutkan pada karakter sebelum karakter “G” yaitu karakter “N”. Hal ini terlihat pada Tabel 2.8.

Tabel 2.8 Iterasi Algoritma *Horspool* kelima

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P															P	U	D	I	N	G
I															0	1	2	3	4	5

Pada Tabel 2.8 terdapat kecocokan lagi pada karakter “N” dan “N”, karena karakter cocok, maka pencocokan dilanjutkan pada karakter sebelum karakter “N” yaitu karakter “I”. Hal ini terlihat pada Tabel 2.9.

Tabel 2.9 Iterasi Algoritma Horspool keenam

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P															P	U	D	I	N	G
I															0	1	2	3	4	5

Pada Tabel 2.9 terdapat kecocokan lagi pada karakter “I” dan “I”, karena karakter cocok, maka pencocokan dilanjutkan pada karakter sebelum karakter “I” yaitu karakter “D”. Hal ini terlihat pada Tabel 2.10.

Tabel 2.10 Iterasi Algoritma Horspool ketujuh

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P															P	U	D	I	N	G
I															0	1	2	3	4	5

Pada Tabel 2.10 terdapat kecocokan lagi pada karakter “D” dan “D”, karena karakter cocok, maka pencocokan dilanjutkan pada karakter sebelum karakter “D” yaitu karakter “U”. Hal ini terlihat pada Tabel 2.11.

Tabel 2.11 Iterasi Algoritma Horspool Kedelapan

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P															P	U	D	I	N	G
I															0	1	2	3	4	5

Pada Tabel 2.11 terdapat kecocokan lagi pada karakter “U” dan “U”, karena karakter cocok, maka pencocokan dilanjutkan pada karakter sebelum karakter “U” yaitu karakter “P”. Hal ini terlihat pada Tabel 2.12.

Tabel 2.12 Iterasi Algoritma *Horspool* kesembilan

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P															P	U	D	I	N	G
I															0	1	2	3	4	5

Pada Tabel 2.12 *window* telah berada pada akhir teks dan semua *pattern* cocok dengan teks. Seluruh pencocokan karakter menggunakan Algoritma *Horspool* telah selesai dan berhenti pada iterasi kesembilan.

2.5 Algoritma *Raita*

Algoritma *Raita* merupakan bagian dari algoritma *exact string matching* yaitu pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama. *Raita* merancang sebuah algoritma dengan membandingkan karakter yang terakhir dari pola dari karakter paling kanan dari *window*. Jika mereka cocok, kemudian karakter pertama dari pola teks paling kiri dari *window* juga dibandingkan. Jika mereka cocok, maka akan dibandingkan karakter tengah pola dengan karakter teks tengah *window*. Pada akhirnya, jika mereka benar-benar cocok, maka algoritma membandingkan karakter lain mulai dari karakter kedua ke karakter kedua terakhir, dan mungkin membandingkan dengan karakter tengah lagi (Charras & Lecroq 1997).

Tahap pencarian Algoritma *Raita*:

- Buat tabel pergeseran pola yang dicari sebagai kata yang akan dicari pada teks.
- Jika dalam proses pembandingan terjadi ketidakcocokan antara pasangan karakter pada akhir pola dengan karakter teks, pergeseran dilakukan sesuai nilai karakter pada Tabel *BmBc*.
- Jika dalam proses pembandingan akhir pola terjadi ketidakcocokan lagi maka karakter akan digeser lagi sesuai Tabel *BmBc*.
- Jika karakter akhir pola dengan karakter pada teks yang sedang dibandingkan cocok, maka posisi karakter pada pola dan teks akan memiliki nilai (0), dan

dilanjutkan pencocokan pada karakter awal pola. Jika cocok maka dilanjutkan pencocokan dengan karakter tengah pola.

- e. Jika akhir, awal dan tengah pola telah cocok. Pencocokan dilanjutkan dengan bagian kanan dari awal karakter pada pola, jika cocok maka dicocokkan pada bagian kanan tengah pola.

Sebagai contoh perhitungan Algoritma *Raita* akan diberikan teks dan *pattern* berikut:

Teks = **MAMTA CULKARI PUDING**

Pattern = **PUDING**


Untuk melakukan perhitungan maka dibuat Tabel *BmBc* dengan persamaan sebagai berikut :

$$m - 2 \dots \dots \dots (2.2)$$

Berfungsi sebagai batas pencarian karakter pada pola.

$$m - i - 1 \dots \dots \dots (2.3)$$

Berfungsi sebagai pencari nilai karakter pada Tabel *BmBc*.

P U D I N G

 0 1 2 3 4 5

Tabel 2.13 Tabel *BmBc*

BC	BM
P	5
U	4
D	3
I	2
N	1
*	6

Berdasarkan Tabel 2.13 dapat diketahui perhitungan Tabel *BmBC* dengan persamaan:

$$m - 2 =$$

$$6 - 2 = 4$$

$$\text{Maka } i = 0 - 4$$

Untuk mencari nilai geser pada Tabel *BmBc*, digunakan rumus :

$$BM[i] = m - i - 1 \dots \dots \dots (2.4)$$

$$BM [BC[P]] = 6 - 0 - 1 = 5$$

$$BM [BC[U]] = 6 - 1 - 1 = 4$$

$$BM [BC[D]] = 6 - 2 - 1 = 3$$

$$BM [BC[I]] = 6 - 3 - 1 = 2$$

$$BM [BC[N]] = 6 - 4 - 1 = 1$$

(*) : karakter yang tidak dikenali

Panjang pola pada contoh adalah sebesar 6. Maka untuk karakter yang tidak ada pada tabel diinisialisasikan dengan tanda (*) yang nilainya sesuai dengan panjang pola (m).

Pencarian Algoritma *Raita* tahap pertama yaitu mencocokkan akhir pola dengan teks, jika terjadi ketidakcocokan maka pola akan bergeser ke kanan sebanyak nilai teks yang ada di tabel 2.14.

Tabel 2.14 Pencarian pada text proses pertama

T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P	P	U	D	I	N	G														

Pada Tabel 2.14 terdapat ketidakcocokan antara karakter “Spasi” dan “G”. Karakter “Spasi” tidak terdapat pada Tabel *BmBc* sehingga digantikan oleh tanda (*). Tanda (*) bernilai sebesar 6 sehingga dilakukan pergeseran sebanyak 6 kali.

Pencarian Algoritma *Raita* tahap kedua yaitu melakukan pergeseran sebanyak 6 kali (sesuai dengan karakter “Spasi”).

Tabel 2.15 Pencarian pada teks proses kedua

T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P							P	U	D	I	N	G								

Dilihat pada Tabel 2.15 terjadi ketidakcocokan antara karakter “R” dan “G”. Karakter “R” tidak terdapat dalam Tabel *BmBc* sehingga digantikan dengan tanda (*). Tanda (*) memiliki nilai sebesar 6 sehingga dilakukan pergeseran sebanyak 6 kali.

Pencarian Algoritma *Raita* tahap ketiga yaitu melakukan pergeseran sebanyak 6 kali (sesuai dengan karakter “R”).

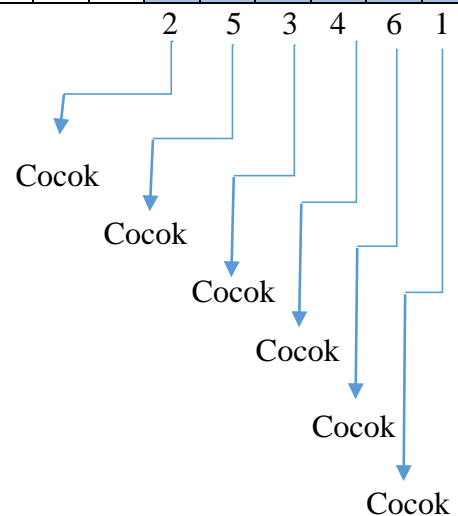
Tabel 2.16 Pencarian pada teks proses ketiga

T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P													P	U	D	I	N	G		

Pada Tabel 2.16 terdapat ketidakcocokan kembali antara karakter “I” dan “G”. Karakter “I” terdapat pada Tabel *BmBc* yang bernilai sebesar 2 sehingga dilakukan pergeseran sebanyak 2 kali. Hal ini terlihat pada Tabel 2.17.

Tabel 2.17 Pencarian pada teks proses keempat

T	M	A	M	T	A		C	U	L	K	A	R	I		P	U	D	I	N	G
P															P	U	D	I	N	G



Dilihat pada Tabel 2.17 terjadi kecocokan antara teks dan pola. Seluruh pencocokan karakter menggunakan Algoritma *Raita* telah selesai dan berhenti pada pencarian teks proses keempat.

2.6 Kompleksitas Algoritma

Suatu masalah dapat mempunyai banyak algoritma penyelesaian. Algoritma yang digunakan tidak saja harus benar, namun juga harus efisien. Efisien suatu algoritma dapat diukur dari waktu eksekusi algoritma dan kebutuhan ruang memori. Algoritma yang efisien adalah algoritma yang meminimumkan kebutuhan waktu dan ruang. Dengan menganalisis beberapa algoritma untuk suatu masalah, dapat diidentifikasi suatu algoritma yang paling efisien. Besaran yang digunakan untuk menjelaskan model pengukuran waktu dan ruang ini adalah kompleksitas algoritma (Azizah, 2013).

Kompleksitas dari suatu algoritma merupakan ukuran seberapa banyak komputasi yang dibutuhkan algoritma tersebut untuk menyelesaikan masalah. Secara informal, algoritma yang dapat menyelesaikan suatu permasalahan dalam waktu yang singkat memiliki kompleksitas yang rendah, sementara algoritma yang membutuhkan waktu lama untuk menyelesaikan masalahnya mempunyai kompleksitas yang tinggi.

Kompleksitas waktu, dinyatakan oleh $T(n)$, diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n , dimana ukuran masukan (n) merupakan jumlah data yang diproses oleh sebuah algoritma. Sedangkan kompleksitas ruang, $S(n)$, diukur dari memori yang digunakan oleh struktur data yang terdapat di dalam algoritma sebagai fungsi dari masukan n . Dengan menggunakan kompleksitas waktu dan kompleksitas ruang, dapat ditentukan laju peningkatan waktu dan ruang yang diperlukan algoritma, seiring dengan ukuran masukan (n) (Azizah, 2013).

Waktu eksekusi algoritma dapat diklasifikasikan menjadi tiga kelompok besar, yaitu *best-case* (kasus terbaik), *average-case* (kasus rerata) dan *worst-case* (kasus terburuk). Pada pemrograman yang dimaksud dengan kasus terbaik, kasus terburuk dan kasus rerata suatu algoritma adalah besar kecilnya atau banyak

sedikitnya sumber-sumber yang digunakan oleh suatu algoritma maka semakin sedikit sumber-sumber algoritma makin semakin baik dan semakin banyak sumber-sumber algoritma makin buruk. Biasanya sumber-sumber yang paling dipertimbangkan tidak hanya waktu eksekusi tetapi bisa juga besar memori dan sumber-sumber lain.

2.7 Notasi Big-O

Notasi O menyatakan *running time* dari suatu algoritma untuk kemungkinan kasus terburuk. Notasi O memiliki dari beberapa bentuk. Notasi O dapat berupa salah satu bentuk maupun kombinasi dari bentuk-bentuk tersebut.

Bentuk $O(1)$ memiliki arti bahwa algoritma yang sedang dianalisis merupakan algoritma konstan. Hal ini mengindikasikan bahwa *running time* algoritma tersebut tetap, tidak bergantung pada n .

Bentuk $O(n)$ berarti bahwa algoritma tersebut merupakan algoritma linier. Artinya, bila n menjadi $2n$ maka *running time* algoritma akan menjadi dua kali *running time* semula.

Bentuk $O(n^2)$ berarti bahwa algoritma tersebut merupakan algoritma kuadrat. Algoritma kuadrat biasanya hanya digunakan untuk kasus dengan n yang berukuran kecil. Sebab, bila n dinaikkan menjadi dua kali semula, maka *running time* algoritma akan menjadi empat kali semula.

Bentuk $O(n^3)$ berarti bahwa algoritma tersebut merupakan algoritma kubik. Pada algoritma kubik, bila n dinaikkan menjadi dua kali semula, maka *running time* algoritma akan menjadi delapan kali semula.

Bentuk $O(2^n)$ berarti bahwa algoritma tersebut merupakan algoritma eksponensial. Pada kasus ini, bila n dinaikkan menjadi dua kali semula, maka *running time* algoritma akan menjadi kuadrat kali semula.

Bentuk $O(\log n)$ berarti algoritma tersebut merupakan algoritma logaritmik. Pada kasus ini, laju pertumbuhan waktu lebih lambat dari pada pertumbuhan n . Algoritma yang termasuk algoritma logaritmik adalah algoritma yang memecahkan persoalan besar dengan mentransformasikannya menjadi beberapa persoalan yang lebih kecil dengan ukuran sama. Basis algoritma tidak terlalu penting, sebab bila

misalkan dinaikkan menjadi dua kali semula, $\log n$ meningkat sebesar jumlah tetapan.

Bentuk $O(n \log n)$, terdapat pada algoritma yang membagi persoalan menjadi beberapa persoalan yang lebih kecil, menyelesaikan setiap persoalan secara independen, kemudian menggabungkan solusi masing-masing persoalan.

Sedangkan bentuk $O(n!)$ berarti bahwa algoritma tersebut merupakan algoritma faktorial. Algoritma jenis ini akan memproses setiap masukan dan menghubungkannya dengan $n - 1$ masukan lainnya. Bila n menjadi dua kali semula, maka *running time* algoritma akan menjadi faktorial dari $2n$ (Azizah, 2013).

2.8 Sistem Operasi *Android*

Android adalah sistem operasi yang didesain sebagai *platform open source* untuk perangkat *mobile* berbasis *Linux* yang mencakup sistem operasi, *middleware*, dan aplikasi. *Android* menyediakan *platform* yang terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri. *Android* juga menyediakan semua *tools* dan *framework* untuk mengembangkan aplikasi dengan mudah dan cepat. Dengan adanya *Android SDK (Software Development Kit)*, pengembang aplikasi dapat membuat aplikasi pada *platform android* menggunakan bahasa pemrograman *Java* (Roeslandy, 2012).

Secara umum, *Android* memiliki beberapa fitur pendukung, dimana fitur-fitur tersebut membuat *Android* menjadi alat telekomunikasi canggih. Adapun fitur-fitur yang mendukung sistem operasi *Android*, yaitu:

1. *Storage*

Menggunakan *SQLite*, mesin *database SQL embedded*, untuk menyimpan data.

2. *Connectivity*

Mendukung GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, *Bluetooth*, WiFi, LTE, dan WiMAX.

3. *Messaging*

Mendukung SMS dan MMS.

4. *Web browser*

Berbasis *open-source WebKit*, dengan *Chrome's V8 JavaScript engine*.

5. *Media support*

Mendukung media: *MP3, MIDI, WAV, JPEG, PNG*, dan lain-lain.

6. *Hardware support*

Mendukung *hardware: Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, GPS*.

7. *Multi-touch*

Mendukung *multi-touch screens*.

8. *Multi-tasking*

Mendukung *multi-tasking applications*.

9. *Flash support*

Android 2.3 mendukung *Flash 10.1*.

10. *Tethering*

Mendukung *sharing* koneksi internet seperti *wireless hotspot*.

2.9 Kelebihan dan Kekurangan Sistem Operasi *Android* :

Adapun kelebihan sistem operasi *android* adalah sebagai berikut:

1. Merupakan sistem operasi yang berbasis *open source*. Sistem operasi ini merupakan sistem operasi berbasis *linux* yang sifatnya *open source* sehingga pemilik *Android* bisa mengedit tampilan secara sendiri dengan mengubah kode *xml*.
2. Aplikasi yang terdapat di *playstore* merupakan aplikasi gratis dan mendukung semua layanan yang ada di *google*.

Adapun kekurangan dari sistem operasi *android* adalah sebagai berikut:

1. Dikarenakan banyaknya aplikasi gratis yang terdapat pada *android store*, mengakibatkan beberapa *developer* aplikasi tersebut mudah untuk memasang *malware* terhadap aplikasi yang disembarkannya.
2. Disebabkan harus bekerja ekstra untuk menjalankan semua proses yang ada pada *OS android*, sehingga penggunaan baterai ponsel akan menjadi boros.
3. Banyaknya iklan yang terdapat pada aplikasi *Android*.

2.10 *Ionic Framework*

Ionic Framework merupakan *framework* HTML5 yang membantu dalam mengembangkan aplikasi *mobile* dengan teknologi *web* seperti HTML, CSS dan *Javascript*. *Ionic Framework* adalah *platform* yang menargetkan seorang *Programmer Web* agar bisa membuat aplikasi *mobile* dengan teknologi *web*. Artinya, *Programmer web* yang ingin menjadi *Programmer Mobile* tidak perlu belajar *Java* atau *Objective C* atau *C#* untuk membuat versi aplikasi dari layanan webnya.

Ionic Framework terdiri dari sekumpulan teknologi yang dikembangkan untuk membangun aplikasi *mobile hybrid* yang *powerful*, cepat, mudah dan juga memiliki tampilan yang menarik. *Ionic Framework* menggunakan *AngularJS* sebagai *framework* berbasis *web* dan menggunakan *Cordova* untuk membangun aplikasi *mobile*.

Beberapa kelebihan *Ionic Framework* dibandingkan dengan *framework* lain yaitu:

1. *Ionic Framework* menggunakan lisensi *Open Source*

Ionic Framework platform menggunakan lisensi *open source*, dimana pengguna dapat membuat aplikasi *free* ataupun komersial dengan *Ionic*.

2. Menggunakan teknologi *web* terbaru

Ionic Framework memanfaatkan *AngularJS* untuk implementasi *logic* nya. *AngularJS* menawarkan performa dan respon cepat seperti aplikasi *native*.

3. Target Hanya untuk *Android* 4 dan *ios* 7 Keatas

Bagi yang suka dengan hal baru tanpa memikirkan kompatibilitas dengan versi *mobile OS* lama, maka *ionic framework* adalah *platform* yang paling pas.

4. Berbasis *Apache Cordova/Phonegap*

Ionic Framework hanya menyediakan *framework* nya, untuk membungkusnya menjadi aplikasi *Android* atau *iOS* anda tetap pakai *phonegap*. Artinya bagi para *programmer phonegap* dengan *platform* lain, keahliannya tetap bisa dipakai (Moreno, 2015).

2.11 Typescript

Typescript adalah bahasa pemrograman berbasis *Javascript* yang menambahkan fitur *strong-typing* & konsep pemrograman OOP klasik (*class*, *interface*). Di dalam dokumentasinya, *Typescript* disebut sebagai *super-set* dari *Javascript*, artinya semua kode *Javascript* adalah kode *Typescript* juga. Bahasa pemrograman ini menawarkan *class*, *module*, dan *interface* yang membuat *developer* bisa mengembangkan aplikasi kompleks dengan lebih mudah. Hal inilah yang membedakannya dengan *javascript* (Hidayat, 2016).

TypeScript menawarkan *class*, *module*, dan *interface* yang membuat *developer* bisa mengembangkan aplikasi kompleks dengan lebih mudah diantara keunggulan dan fitur – fiturnya (Hidayat, 2016), antara lain :

1. *Support Class dan Module*
2. *Static Type-checking*
3. *Support ES6 Feature*
4. *Clear Library API Definition*
5. *Build-in Support untuk JavaScript Packaging*
6. *Kesamaan Syntax untuk Backend*
7. *Superset dari JavaScript.*

2.12 Angular-CLI

Angular-CLI merupakan *generator* untuk *project Angular 2* dan *4* (serta versi di atasnya), dengan menggunakan bantuan *Angular-CLI* ini kita tidak perlu lagi *setup* banyak kebutuhan dasar seperti susunan *folder*, cara *run*, cara *build* dan lain-lain. Kesemuanya itu telah ada perintahnya (*command*) melalui *Angular-CLI*. Seperti halnya *Ember.js* yang mempunyai *Ember-CLI*, *Vuejs* dengan *Vue-CLI* dan *React* dengan “*create-react-app*” maka seperti itu pula fungsi *Angular-CLI* bagi *Angular* (Maulana, 2017).

2.13 *File.js*

File.js merupakan *file javascript* yang dibuat menjadi satu kesatuan, dari suatu *javascript* yang begitu panjang dapat dibuat menjadi satu *file* dengan *format* (.js).



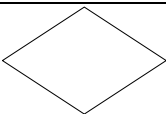

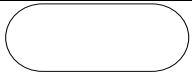


Suatu *javascript* yang mengandung *file.js*, memang umumnya sulit untuk di *edit*, karena harus membongkar terlebih dahulu *file.js* tersebut, kemudian baru dapat mengedit *script* yang terkandung dalam *file.js* tersebut.

File.js biasanya diimpor pada bagian kepala *HTML*. Selain itu dapat membandingkan fungsi-fungsi, yang melakukan validitas *form*, membuat menu-menu *drop-down* atau membuka dan menutup *Windows* (Kadir, 2002).



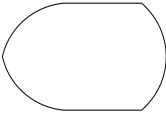
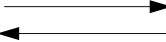
2.14 *Flowchart*

Flowchart adalah simbol-simbol pekerjaan yang menunjukkan bagan aliran proses yang saling terhubung sehingga setiap simbol *flowchart* melambangkan pekerjaan dan instruksinya (Ewolf, 2011). Simbol-simbol yang digunakan dalam *flowchart* adalah sebagai berikut:

Tabel 2.18 Simbol-simbol *flowchart*

NO.	GAMBAR	NAMA	KETERANGAN
1.		Proses	Mempresentasikan operasi.
2.		<i>Input / Output</i>	Mempresentasikan <i>Input</i> atau <i>Output</i> data yang diproses atau informasi.
3.		Keputusan	Keputusan dalam program.
4.		Dokumen	Dokument I / O dalam format cetak.
5.		<i>Terminal points</i>	Awal / akhir <i>flowchart</i> .
6.		<i>Preparation</i>	Pemberian harga awal.
7.		Manual input	<i>Input</i> yang dimasukkan secara manual dari <i>keyboard</i> .

Tabel 2.18 Simbol-simbol *flowchart* (Lanjutan)

NO.	GAMBAR	NAMA	KETERANGAN
8.		Penghubung	Keluar atau masuk dari bagian lain <i>flowchart</i> khususnya.
9.		Penghubung	Keluar atau masuknya dari bagian lain <i>flowchart</i> khususnya halaman lain.
10.		<i>Display</i>	<i>Output</i> yang ditampilkan pada terminal.
11.		Anak panah	Mempresentasikan alur kerja.

Sumber: Ewolf, 2011.










2.15 *Unified Modeling Language* (UML)

Unified Modeling Language (UML) merupakan bahasa dalam mendesain perangkat lunak secara visual. Dengan UML, desainer dapat melihat konsep global suatu desain. Desain kemudian dapat dijadikan panduan dalam proses pengembangan dan rekayasa perangkat lunak. Selain itu, UML dapat menjadi media komunikasi gagasan antara pengembang perangkat lunak dengan pengguna (Shalahudin, 2011). Ada beberapa jenis diagram dalam UML yaitu:

1. *Use Case Diagram*

Use case diagram menggambarkan sejumlah *external actors* dan hubungannya ke *use case* yang diberikan oleh sistem. *Use case* adalah deskripsi fungsi yang disediakan oleh sistem dalam bentuk teks sebagai dokumentasi dari *use case symbol* namun dapat juga dilakukan dalam *activity diagrams*. *Use case* digambarkan hanya yang dilihat dari luar oleh *actor* (keadaan lingkungan sistem yang dilihat *user*) dan bukan bagaimana fungsi yang ada di dalam sistem. (Shalahudin, 2011).

Tabel 2.19 Simbol *use case diagram*





NO	GAMBAR	NAMA	KETERANGAN
1.		<i>Actor</i>	Menspesifikasikan himpunan peran yang pengguna mainkan ketika berinteraksi dengan <i>use case</i> .
2.		<i>Dependency</i>	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri (<i>dependent</i>) akan mempengaruhi elemen yang bergantung padanya elemen yang tidak mandiri (<i>independent</i>).
3.		<i>Generalization</i>	Hubungan dimana objek anak (<i>descendent</i>) berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (<i>ancestor</i>).
4.		<i>Include</i>	Menspesifikasikan bahwa <i>use case</i> sumber secara eksplisit.
5.		<i>Extend</i>	Menspesifikasikan bahwa <i>use case</i> target memperluas perilaku dari <i>use case</i> sumber pada suatu titik yang diberikan.
6.		<i>Association</i>	Apa yang menghubungkan antara objek satu dengan objek lainnya.
7.		<i>System</i>	Menspesifikasikan paket yang menampilkan sistem secara terbatas.
8.		<i>Use case</i>	Deskripsi dari uraian aksi-aksi yang ditampilkan system yang menghasilkan suatu hasil yang terukur bagi suatu <i>actor</i> .
9.		<i>Collaboration</i>	Interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dari jumlah dan elemen-elemennya (sinergi).

Sumber: Shalahuddin, M., 2011.

2. Activity Diagram

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana alir berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Tabel 2.20 Simbol *activity diagram*

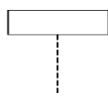
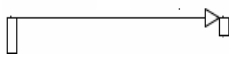
NO	GAMBAR	NAMA	KETERANGAN
1.		<i>Activity</i>	Memperlihatkan bagaimana masing-masing kelas antarmuka saling berinteraksi satu sama lain.
2.		<i>Action</i>	State dari sistem yang mencerminkan eksekusi dari suatu aksi.
3.		<i>Initial Node</i>	Bagaimana objek dibentuk atau diawali.
4.		<i>Activity Final Node</i>	Bagaimana objek dibentuk dan diakhiri.

Sumber: Shalahuddin, M., 2011.

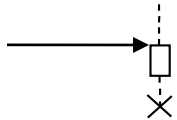
3. Sequence Diagram

Sequence diagram menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu objek dan pesan yang dikirimkan dan diterima antar objek. Jumlah *Sequence diagram* yang akan digambar adalah minimal sama dengan jumlah *use case* yang didefinisikan (Sukamto, 2013).

Tabel 2.21 Simbol *sequence diagram*

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

Tabel 2.21 Simbol *sequence diagram* (Lanjutan)






NO	GAMBAR	NAMA	KETERANGAN
3		<i>Message</i>	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi.
4		<i>Message</i>	Menyatakan suatu objek mengakhiri hidup objek lain, arah panah mengarah pada objek yang diakhiri, sebaiknya jika ada <i>create</i> maka ada <i>destroy</i> .

Sumber: Sukamto dan Shalahuddin, 2013.

4. *Class Diagram*

Class diagram adalah kumpulan objek-objek yang mempunyai struktur umum, *behavior* umum, relasi umum, dan *semantic* atau kata yang umum. *Class-class* ditentukan atau ditemukan dengan cara memeriksa objek-objek dalam *sequence diagram* dan *collaboration diagram*. Sebuah *class* digambarkan seperti sebuah bujur sangkar dengan tiga bagian ruangan.

Tabel 2.22 Simbol *class diagram*

NO	GAMBAR	NAMA	KETERANGAN
1.		<i>Association</i>	Apa yang menghubungkan antara objek satu dengan objek lainnya
2.		<i>Class</i>	Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama.
3.		<i>Collaboration</i>	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang.
4.		<i>Realization</i>	Operasi yang benar-benar dilakukan oleh suatu objek.
5.		<i>Dependency</i>	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri akan mempengaruhi elemen yang bergantung padanya elemen yang tidak mandiri.

Sumber: Sukamto dan Shalahuddin, 2013.

2.16 Metode Pengembangan Sistem

Metode Pengembangan sistem yang digunakan pada tugas ini adalah metode *Rational Unified Process* (RUP). RUP merupakan salah satu proses rekayasa perangkat lunak yang menyediakan pendekatan untuk menentukan tugas dan tanggung jawab dalam pengembangan suatu organisasi, tujuannya adalah untuk memastikan produksi kualitas tinggi, *software* memenuhi dengan kebutuhan *user* sesuai dengan jadwal dan biaya yang telah dirancang. Dalam metode RUP ini, terdiri dari 4 tahap, yaitu:

a. *Inception*

Tahap ini membangun *business case* untuk sistem dan membatasi ruang lingkupnya, untuk melakukan hal ini diharuskan untuk mengidentifikasi semua entitas eksternal yang akan berinteraksi dengan sistem, dan mendefinisikan interaksi pada level tertentu. Ini juga termasuk mengidentifikasi semua *use cases* dan menjelaskan beberapa yang signifikan. *Business case* termasuk kriteria keberhasilan, perkiraan resiko, dan mengestimasi sumber daya yang dibutuhkan.

b. *Elaboration*

Tujuan dari tahap *elaboration* adalah menganalisis domain masalah, membuat sebuah dasar arsitektur, membangun rencana proyek, dan mengeliminasi resiko terbesar dari proyek. Untuk menjalankan objek-objek tersebut diperlukan melihat lebih luas dan lebih dalam terhadap sistem. Pada tahap ini merupakan tahap paling sulit karena pada tahap ini memastikan bahwa arsitektur, kebutuhan, dan perencanaan cukup stabil sehingga waktu dan biaya tidak berubah.

c. *Construction*

Dalam tahap ini semua komponen dan fitur aplikasi yang dibuat dan diintegrasikan kedalam *software*. Dalam tahapan ini juga dituntut untuk mengoptimalkan sumber daya, biaya, jadwal dan kualitas. Pada tahapan ini meliputi bagaimana suatu aplikasi biasa diimplementasikan dan diuji coba.

1. Implementasi

Penjelasan mengenai perangkat keras dan perangkat lunak apa saja yang dibutuhkan untuk mengimplementasikan sistem.

2. *Coding*

Proses pengkodean dilakukan dengan menggunakan bahasa pemrograman. Pengkodean sendiri berisi tahapan-tahapan perhitungan metode.

3. *Testing*

Pada tahap ini dilakukan pengujian terhadap aplikasi yang telah dibangun untuk mengetahui tingkat akurasi dan kualitas dari aplikasi tersebut, apakah sudah sesuai dengan yang diharapkan atau tidak. *Testing* dilakukan dengan menguji semua tombol-tombol yang terdapat pada aplikasi apakah sudah berjalan sesuai dengan fungsi nya atau tidak.

d. *Transition*

Pada tahap ini dilakukan *testing* akhir pada sistem yang telah jadi, kemudian dilakukan sosialisasi penggunaan perangkat lunak yang telah dibangun ke *administrator*.

2.17 Metode Pengujian Sistem

Menurut Pressman (2002), pengujian *black-box* berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengujian *black-box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi masukan yang menggunakan semua persyaratan fungsional untuk suatu program. Pengujian *black-box* bukan alternatif dari teknik *white-box*, tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas kesalahan daripada metode *white-box*.

Pengujian *black-box* menemukan kesalahan dalam kategori sebagai berikut:

1. Fungsi-fungsi yang tidak benar atau hilang.
2. Kesalahan *interface*.
3. Kesalahan dalam struktur data atau akses *database* eksternal.
4. Kesalahan kinerja.
5. Inisialisasi dan kesalahan terminasi.

Tujuan dari metode *black-box testing* adalah mendapatkan kesalahan sebanyak-banyaknya. Metode ini dilakukan dengan cara menjalankan program yang dihasilkan. Kemudian diamati apakah hasil dari program tersebut sesuai dengan yang diinginkan. Jika masih terdapat kesalahan atau hasil yang tidak sesuai dengan yang diinginkan, maka kesalahan ataupun ketidaksesuaian tersebut dicatat untuk selanjutnya diperbaiki (Pratiwi, H., 2014).

BAB III

METODOLOGI PENELITIAN

3.1 Waktu dan Tempat Penelitian

3.1.1 Waktu

Waktu pelaksanaan penelitian tugas akhir dilaksanakan mulai dari bulan Desember 2018 sampai dengan Maret 2019. Rincian kegiatan dapat dilihat pada Tabel 3.1 berikut:

Tabel 3.1 Gantt Chart waktu penelitian

No	Uraian	Waktu (2018-2019)															
		Desember				Januari				Februari				Maret			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	<i>Inception</i>																
2	<i>Elaboration</i>																
3	<i>Construction</i>																
4	<i>Transition</i>																

3.1.2 Tempat Penelitian

Adapun tempat penelitian tugas akhir yang akan dilakukan tidak mengacu pada tempat tertentu.

3.2 Metode Pengumpulan Data

Metode pengumpulan data yang digunakan dalam penelitian ini adalah dengan melakukan studi kepustakaan. Melalui studi pustaka penulis menghimpun data dari jurnal, Perpustakaan Prodi Teknik Informatika Universitas Halu Oleo, dan *interview* Ibu Astri Yunita, S.Psi.,M.Psi.,Psikolog sebagai dosen psikologi yang berkaitan dengan data kata istilah pada aplikasi Kamus Istilah Psikologi.

3.3 Metode Pengembangan Sistem

Metode pengembangan sistem yang digunakan dalam sistem ini adalah metode *Rational Unified Process* (RUP). Dalam metode ini, terdapat empat tahap pengembangan perangkat lunak, yaitu:

3.3.1 Permulaan (*Inception*)

Pada *fase* ini dilakukan proses pengidentifikasian sistem, dilakukan dengan analisis kebutuhan akan aplikasi, melakukan kajian terhadap penelitian yang terkait dengan Algoritma *Horspool* dan Algoritma *Raita*.

3.3.2 Perluasaan/Perencanaan (*Elaboration*)

Setelah menentukan ruang lingkup penelitian, tahap ini akan dilakukan perancangan dan analisis sistem menggunakan *flowchart* meliputi *flowchart flowchart* Sistem, Algoritma *Horspool* dan Algoritma *Raita*. Pada perancangan ini, digunakan juga UML (*Unified Modelling Language*) yang meliputi *use case diagram*, *activity diagram*, *class diagram* dan *sequence diagram*.

3.3.3 Konstruksi (*Construction*)

Proses yang dilakukan pada tahap ini yaitu membangun aplikasi dengan perancangan yang telah dilakukan sebelumnya, mulai dari tampilan *interface* menu Beranda, Kamus Psikologi (memilih pencarian Algoritma *Horspool* dan pencarian Algoritma *Raita*), menu Favorit dan menu Tentang Aplikasi. Proses yang juga dilakukan pada tahap ini yaitu penerapan *coding* Algoritma *Horspool* dan Algoritma *Raita* pada aplikasi sebagai metode pencarian.

3.3.4 Transisi (*Transition*)

Pada tahap *Transition* difokuskan untuk melakukan proses pengujian terhadap aplikasi. Dalam penelitian ini, dilakukan pengujian menggunakan pengujian *black box*, pengujian 1 kata, pengujian 2 kata, pengujian 3 kata, pengujian kata yang tidak terdapat pada aplikasi, dan pengujian kompleksitas algoritma (*big O*) terhadap aplikasi.

BAB IV

ANALISIS DAN PERANCANGAN SISTEM

4.1 Analisis Sistem

Analisis sistem merupakan suatu tahapan yang bertujuan untuk mengetahui dan mengamati apa saja yang terlibat dalam suatu sistem. Pembahasan yang ada pada analisis sistem ini yaitu analisis kebutuhan fungsional meliputi perancangan sistem menggunakan bahasa pemodelan UML (*Unified Modeling Language*), perancangan tampilan *interface* serta analisis kebutuhan fungsional dan nonfungsional yang meliputi kebutuhan perangkat keras dan perangkat lunak yang akan digunakan.

4.1.1 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional adalah segala bentuk data yang dibutuhkan oleh sistem agar sistem dapat berjalan sesuai dengan prosedur yang dibangun melalui perancangan sistem. Adapun kebutuhan fungsional dari sistem yang akan dibangun, yaitu;

1. Perancangan diagram sistem menggunakan bahasa pemodelan UML (*Unified Modeling Language*) yang meliputi pembuatan *flowchart* sistem, *flowchart* metode, *use case diagram*, *activity diagram*, *class diagram* serta *sequence diagram*.
2. Menganalisis kebutuhan data yang terdiri dari data kata istilah psikologi yang akan dimasukkan dalam *file json*, melakukan proses menggunakan Algoritma *Horspool* dan Algoritma *Raita* dalam pencarian kata pada aplikasi kamus istilah psikologi serta *output* dari aplikasi berupa hasil pencarian kata istilah beserta waktu pencarian.

4.1.2 Analisis Kebutuhan Non-fungsional

Analisis kebutuhan non-fungsional adalah sebuah langkah dimana pembangun aplikasi menganalisis sumber daya yang dibutuhkan untuk membangun aplikasi yang akan dibangun. Analisis kebutuhan nonfungsional yang dilakukan

dibagi dalam dua tahap, yaitu analisis kebutuhan perangkat keras dan analisis kebutuhan perangkat lunak. Kebutuhan perangkat keras yaitu kebutuhan perangkat atau komponen yang dibutuhkan pada sistem dan perangkat lunak yaitu kebutuhan perangkat lunak untuk membantu agar komponen perangkat keras dapat berfungsi dan dapat dijalankan pada sistem.

4.1.2.1 Kebutuhan Perangkat Keras

Untuk menerapkan rancangan yang telah dijelaskan sebelumnya, dibutuhkan beberapa perangkat keras sebagai sarana untuk mengimplementasikan aplikasi yang dibangun. Berikut ini spesifikasi perangkat keras yang dibutuhkan.

Tabel 4.1 Spesifikasi perangkat keras

No	Nama Perangkat	Spesifikasi
1.	<i>Notebook</i>	<i>Asus A455LD</i>
2.	<i>Processor</i>	<i>Intel Core i5 2.10 GHz</i>
3.	<i>Monitor</i>	<i>Monitor 14 inch</i>
4.	<i>Memori</i>	<i>RAM 6 GB DDR 3L</i>
5.	<i>Harddisk</i>	<i>500 GB HDD</i>
6.	<i>Smartphone Samsung J7 Prime</i>	<i>RAM 3 GB</i>

4.1.2.2 Kebutuhan Perangkat Lunak

Perangkat lunak yang digunakan pada pembangunan aplikasi glosarium teknologi informasi. Adapun rincian kebutuhan perangkat lunak dapat dilihat pada Tabel 4.2 berikut ini.

Tabel 4.2 Spesifikasi perangkat lunak

No	Nama Perangkat	Spesifikasi
1.	<i>Operating System</i>	<i>Windows 10 Profesional 64 bit</i>
2.	<i>Java Development Kit (JDK)</i>	<i>JDK 1.8.0</i>
3.	<i>Java Runtime Environment (JRE)</i>	<i>JRE 8</i>
4.	<i>SDK</i>	<i>28.0.3</i>
5.	<i>Ionic</i>	<i>4.11.0</i>
6.	<i>TypeScript</i>	<i>-</i>
7.	<i>Node JS</i>	<i>6.4.1</i>
8.	<i>Android OS</i>	<i>Android 8.1 Oreo</i>

4.2 Analisis Perancangan Sistem

Perancangan sistem yang akan dibangun terdiri atas perancangan *flowchart* dan perancangan UML serta perancangan *user interface*.

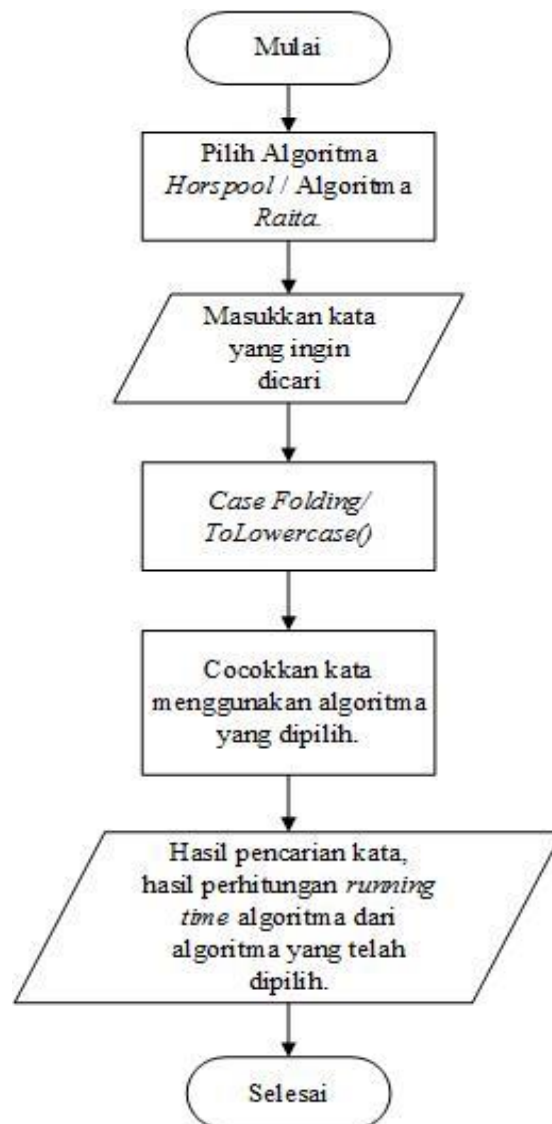
4.2.1 Perancangan *Flowchart*

Flowchart adalah simbol-simbol pekerjaan yang menunjukkan bagan aliran proses yang saling terhubung. Perancangan *flowchart* yang akan dibangun terdiri atas *flowchart* sistem, *flowchart* Algoritma *Horspool* dan *flowchart* Algoritma *Raita*.

4.2.1.1 *Flowchart* Sistem

Setelah menganalisis sistem, maka didapatkan *flowchart diagram* sistem Perbandingan Algoritma *Horspool* dan Algoritma *Raita* Pada Aplikasi Kamus Istilah Psikologi Berbasis *Android* yang ditunjukkan oleh Gambar 4.1. Adapun alur kerja *flowchart diagram* sistem adalah sebagai berikut:

- a. *User* memilih algoritma pencarian yang ingin dipakai.
- b. *User* memasukkan kata yang ingin dicari.
- c. Kata yang dimasukkan akan melalui proses *case folding/toLowerCase* untuk mengubah kata menjadi huruf kecil.
- d. Kata yang dicari akan dicocokkan menggunakan algoritma yang dipilih.
- e. Kemudian sistem akan menampilkan kata yang dicari beserta waktu pencarian dari metode tersebut.



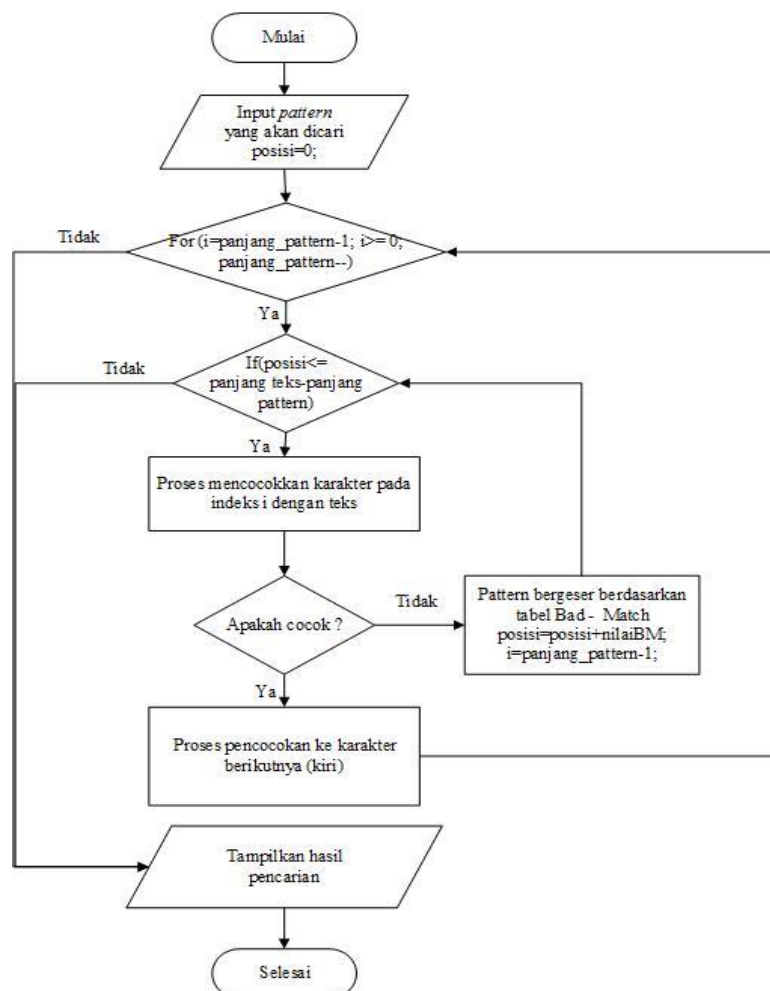
Gambar 4.1 Flowchart Sistem

4.2.1.2 Flowchart Algoritma Horspool

Setelah menganalisis sistem, maka didapatkan *flowchart diagram* Algoritma *Horspool* pada Aplikasi Kamus Istilah Psikologi Berbasis Android yang ditunjukkan oleh Gambar 4.2. Adapun alur kerja *flowchart diagram* Algoritma *Horspool* adalah sebagai berikut:

- Pertama *user* memasukkan *pattern* yang akan dicari dan posisi = 0.
- Sistem melakukan proses pencarian dengan syarat *for* $i = \text{panjang } pattern$ dimana $i \geq 0$.

- c. Kemudian mengecek posisi *pattern* terhadap teks, jika posisi \leq panjang teks – panjang *pattern* jika iya maka dilakukan proses pengecekan dan apabila tidak akan menampilkan hasil pencarian.
- d. Selanjutnya sistem melakukan proses pencocokkan karakter paling kanan *pattern* dengan teks.
- e. Apabila terjadi ketidakcocokkan maka *pattern* bergeser sebesar nilai di Tabel *BmBc*, posisi ditambah dengan nilai *BM* dan juga $i = \text{panjang_pattern} - 1$.
- f. Apabila terjadi kecocokkan, dilanjutkan dengan karakter berikutnya (kiri).
- g. Kemudian sistem mencocokkan kembali *pattern* dan teks apabila tidak cocok maka hasil pencarian selesai dan apabila terjadi kecocokkan maka sistem akan menampilkan hasil pencarian dan selesai.

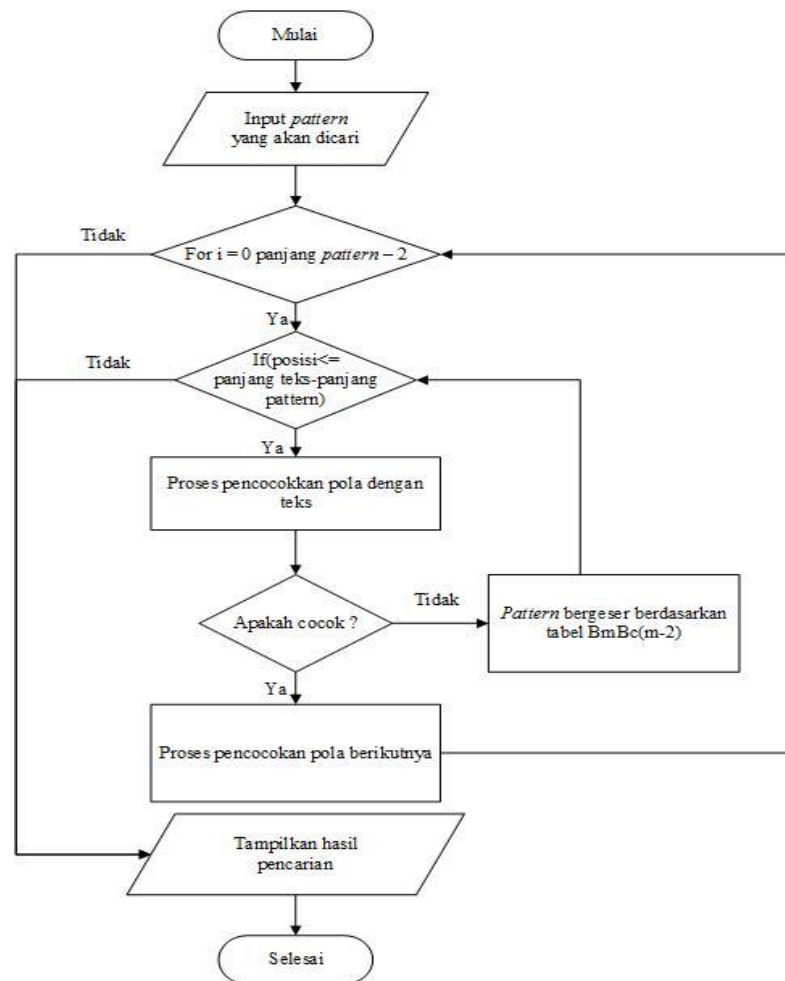


Gambar 4.2 Flowchart Algoritma Horspool

4.2.1.3 Flowchart Algoritma Raita

Setelah menganalisis sistem, maka didapatkan *flowchart diagram* Algoritma *Raita* pada Aplikasi Kamus Istilah Psikologi Berbasis Android yang ditunjukkan oleh Gambar 4.3. Adapun alur kerja *flowchart diagram* Algoritma *Raita* adalah sebagai berikut:

- a. Pertama user memulai dan memasukkan *pattern* yang akan dicari.
- b. Sistem melakukan proses pencarian dengan syarat *for i = 0 to panjang pattern - 2*.
- c. Kemudian mengecek posisi *pattern* terhadap teks, jika posisi \leq panjang teks – panjang *pattern* jika iya maka dilakukan proses pengecekan dan apabila tidak akan menampilkan hasil pencarian.
- d. Selanjutnya sistem melakukan proses pencocokkan pola dengan teks.
- e. Apabila terjadi ketidakcocokkan maka *pattern* bergeser sebesar nilai yang ada di Tabel *BmBc* dan melakukan proses pencocokkan kembali.
- f. Apabila terjadi kecocokkan antara *pattern* dan teks maka akan dicocokkan dengan pola karakter berikutnya (kiri).
- g. Kemudian sistem mencocokkan kembali *pattern* dan teks apabila tidak cocok maka hasil pencarian selesai dan apabila terjadi kecocokkan maka sistem akan menampilkan hasil pencarian dan selesai.



Gambar 4.3 Flowchart Algoritma Raita

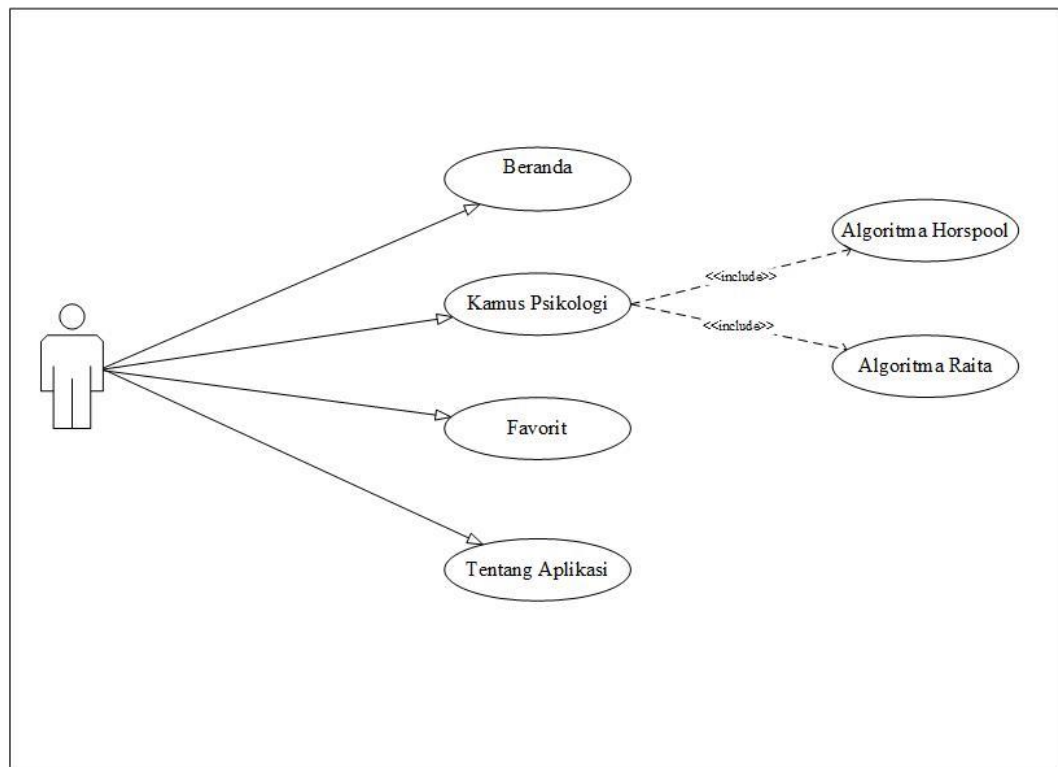
4.2.2 Unified Modeling Language (UML)

Aplikasi dibangun dengan menggunakan *Unified Modeling Language* (UML). UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram yang terdiri dari *Use Case Diagram*, *Activity Diagram*, *Class Diagram* dan *Sequence Diagram*.

4.2.2.1 Use Case Diagram

Use Case Diagram adalah sebuah diagram yang dapat merepresentasikan interaksi yang terjadi antara *user* dengan sistem. *Use Case Diagram* ini mendeskripsikan siapa saja yang menggunakan sistem dan bagaimana cara mereka

berinteraksi dengan sistem. *Use Case Diagram* dari sistem yang akan dibangun dapat ditunjukkan pada Gambar 4.4.



Gambar 4.4 *Use case diagram* aplikasi

Tabel 4.3 Keterangan *use case diagram*

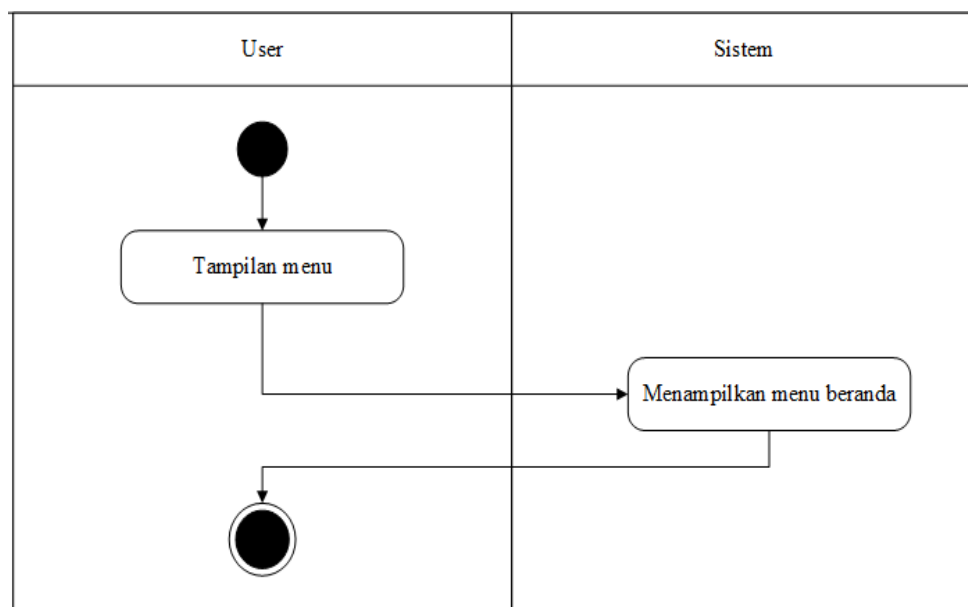
Aktor	Sistem
<i>User</i> memilih menu beranda	Sistem akan menampilkan menu beranda.
<i>User</i> memilih kamus psikologi	Sistem akan menyiapkan <i>form</i> yang berisi pencarian istilah, serta user juga dapat memilih algoritma mana yang ingin digunakan, Algoritma <i>Horspool</i> atau Algoritma <i>Raita</i> .
<i>User</i> memilih menu favorit	Sistem akan menampilkan daftar istilah-istilah psikologi yang telah difavoritkan.
<i>User</i> memilih menu tentang aplikasi	Sistem akan menampilkan deskripsi sistem dan menampilkan profil pembuat aplikasi.

4.2.2.2 Activity Diagram

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat mengtatabelkan proses paralel yang mungkin terjadi pada beberapa eksekusi. Berikut ini adalah *activity diagram* yang akan menggambarkan alir aktivitas sistem.

1. Activity Diagram Beranda

Gambar 4.5 merupakan gambar *Activity Diagram* yang menunjukkan aktivitas *user* ketika membuka aplikasi, lalu sistem akan menampilkan menu beranda yang berisikan tampilan menu beranda.

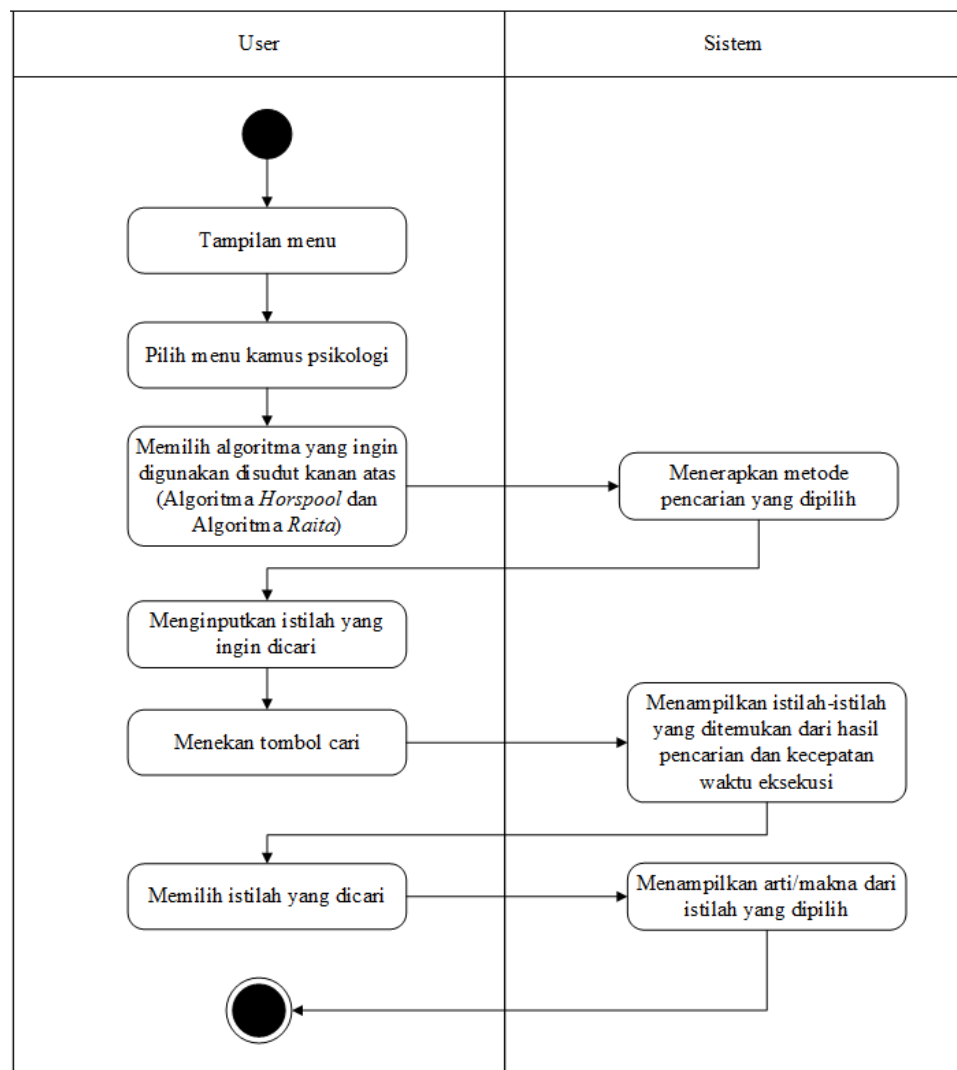


Gambar 4.5 Activity diagram beranda

2. Activity Diagram Kamus Psikologi

Gambar 4.6 merupakan gambar *Activity Diagram* yang menunjukkan aktivitas *user* ketika telah memilih menu kamus psikologi, lalu *user* akan memilih algoritma pencarian yang akan digunakan yaitu Algoritma *Horspool* dan Algoritma *Raita*. Lalu *user* memilih salah satu algoritma. Setelah memilih algoritma, *user* akan menginput istilah yang dicari lalu menekan tombol cari dan sistem akan menampilkan istilah-istilah

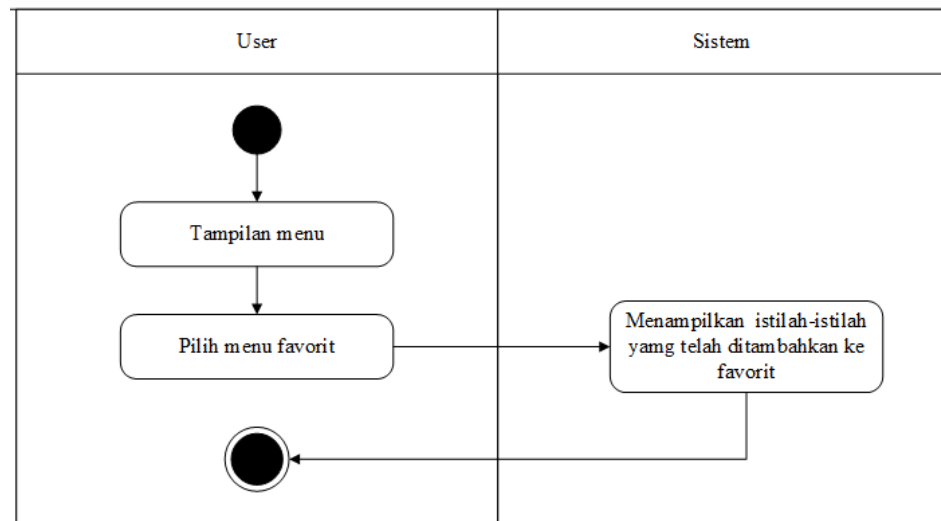
yang ditemukan dari pencarian dan menampilkan kecepatan waktu eksekusi. Setelah itu *user* akan memilih istilah yang dicari dan sistem akan menampilkan arti atau makna dari istilah yang dipilih.



Gambar 4.6 Activity diagram kamus psikologi

3. Activity Diagram Favorit

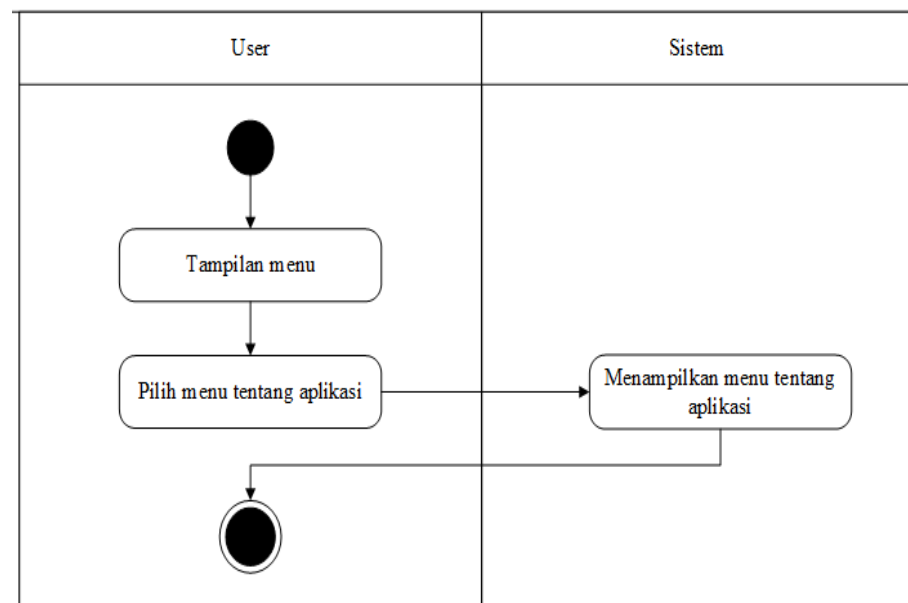
Gambar 4.7 merupakan gambar *Activity Diagram* yang menunjukkan aktivitas *user* memilih menu favorit, maka sistem akan menampilkan menu favorit. Setelah itu, menu favorit tersebut akan menampilkan istilah-istilah psikologi yang telah ditambahkan ke favorit.



Gambar 4.7 Activity diagram favorit

4. Activity Diagram Tentang Aplikasi

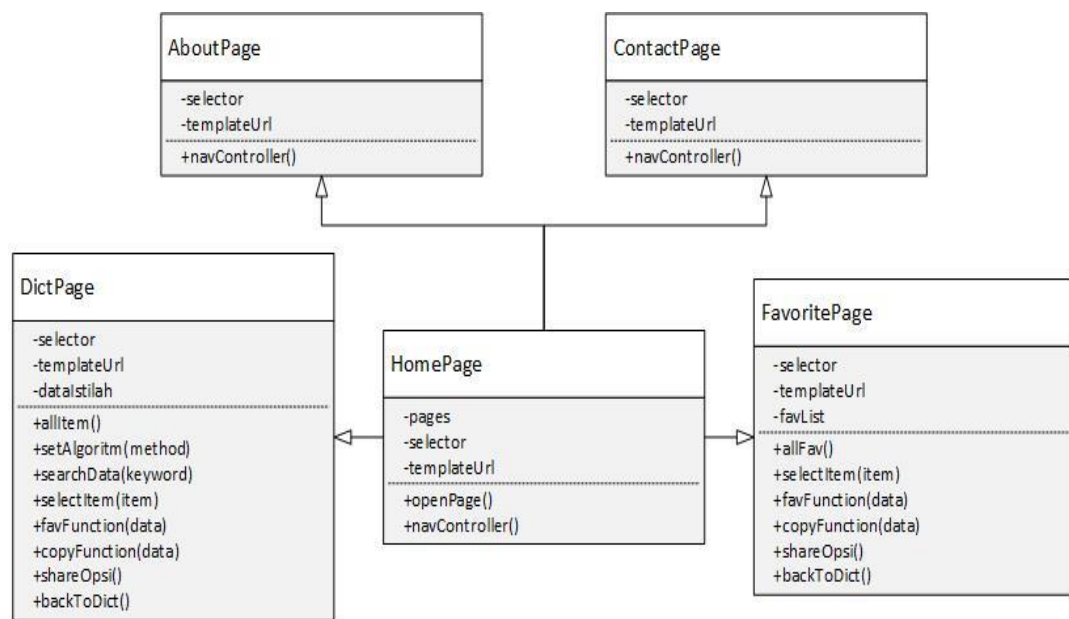
Gambar 4.8 merupakan gambar *Activity Diagram* yang menunjukkan aktivitas *user* memilih menu tentang aplikasi, maka sistem akan menampilkan menu tentang aplikasi. Setelah itu, menu tentang aplikasi tersebut akan menampilkan deskripsi sistem dan menampilkan profil pembuat aplikasi.



Gambar 4.8 Activity diagram tentang aplikasi

4.2.2.3 Class Diagram

Class diagram merupakan diagram yang selalu ada dipemodelan sistem berorientasi objek. *Class diagram* menunjukkan hubungan antar *class* dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan. Berikut ini adalah *class diagram* aplikasi kamus istilah psikologi.

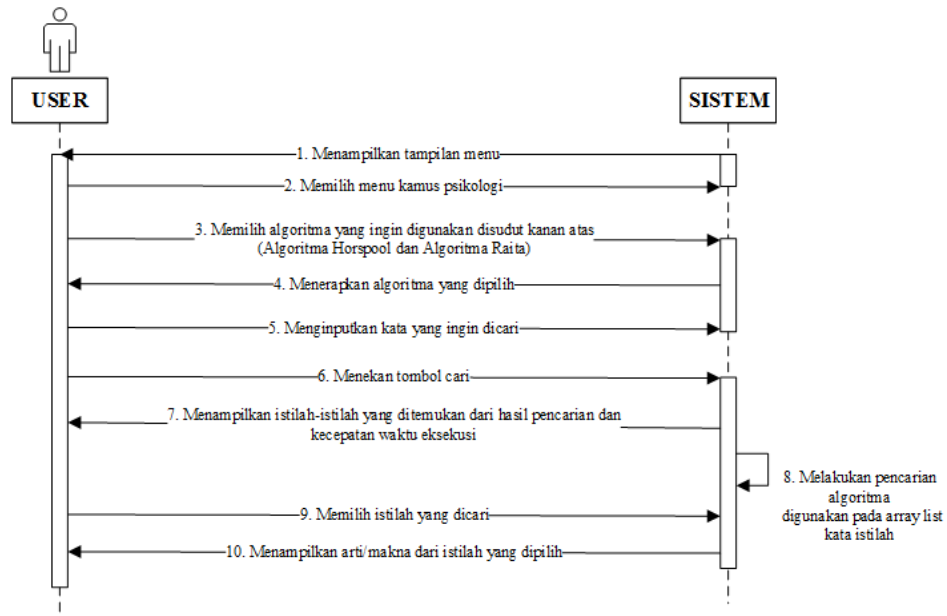


Gambar 4.9 Class diagram aplikasi istilah psikologi

4.2.2.4 Sequence Diagram

Sequence Diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem yang digambarkan terhadap waktu. Berikut ini adalah *Sequence Diagram* yang akan menggambarkan interaksi antar objek dan sistem.

1. Sequence Diagram Aplikasi



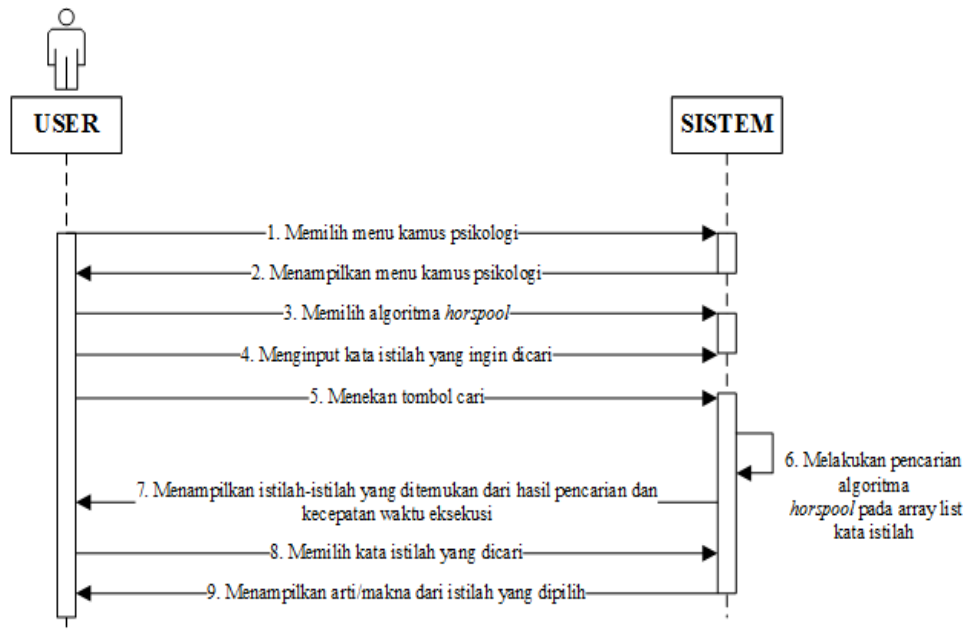
Gambar 4.10 Sequence diagram aplikasi

Gambar 4.10 adalah diagram *sequence* yang menunjukkan hubungan dari sistem terhadap *actor*, dimana sistem akan menampilkan tampilan menu, lalu *user* memilih menu kamus psikologi, lalu *user* memilih algoritma yang ingin digunakan yaitu Algoritma *Horspool* dan Algoritma *Raita*. Kemudian sistem akan menerapkan salah satu algoritma pencarian yang dipilih dan *user* menginput kata yang ingin dicari lalu menekan tombol cari. sistem kemudian akan menampilkan istilah-istilah yang ditemukan dari hasil pencarian beserta kecepatan waktu eksekusi. *User* lalu memilih istilah yang dicari dan sistem menampilkan arti/makna dari istilah yang dipilih.

2. Sequence Diagram Algoritma Horspool

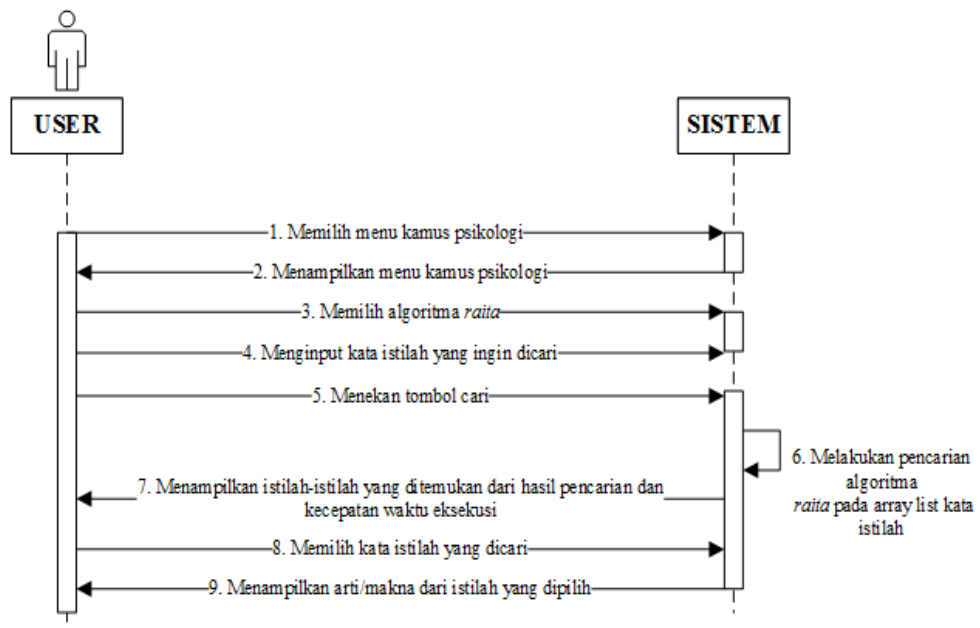
Gambar 4.11 adalah diagram *sequence* yang menunjukkan hubungan dari sistem terhadap *actor*, dimana *user* memilih menu cari istilah lalu memilih Algoritma *Horspool*. *User* lalu memasukkan kata yang akan dicari, lalu sistem melakukan proses pencarian kata dan sistem menampilkan istilah-istilah yang ditemukan dari hasil pencarian beserta

kecepatan waktu eksekusi, kemudian *user* memilih kata istilah yang dicari dan sistem akan menampilkan arti atau makna dari istilah yang dipilih.



Gambar 4.11 *Sequence diagram* Algoritma Horspool

3. *Sequence Diagram* Algoritma Raita



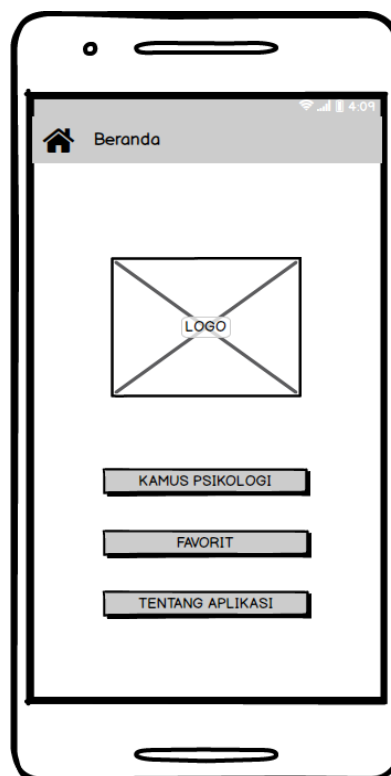
Gambar 4.12 *Sequence diagram* Algoritma Raita

Gambar 4.12 adalah diagram *sequence* yang menunjukkan hubungan dari sistem terhadap *actor*, dimana *user* memilih menu cari istilah lalu memilih Algoritma *Raita*. *User* lalu memasukkan kata yang akan dicari, lalu sistem melakukan proses pencarian kata dan sistem menampilkan istilah-istilah yang ditemukan dari hasil pencarian beserta kecepatan waktu eksekusi, kemudian *user* memilih kata istilah yang dicari dan sistem akan menampilkan arti atau makna dari istilah yang dipilih.

4.3 Perancangan *User Interface*

Perancangan *user interface* adalah tahapan pembuatan antarmuka yang akan digunakan pada pembangunan aplikasi kamus yang dibagi menjadi lima bagian yaitu perancangan tampilan menu beranda, tampilan kamus psikologi, tampilan favorit dan tampilan tentang aplikasi.

4.3.1 Tampilan Beranda



Gambar 4.13 Tampilan beranda

Tampilan beranda adalah tampilan pertama yang akan digunakan pada perangkat *mobile android*. Pada tampilan beranda berisi logo aplikasi dan menu-menu pada aplikasi.

4.3.2 Tampilan Menu Kamus Psikologi

Pada tampilan menu kamus psikologi ini terdapat satu kolom pencarian, list kata-kata istilah psikologi, dan dua tab *setting* untuk memilih metode pencarian (*Algoritma Horspool* atau *Algoritma Raita*) yang terdapat pada bagian kanan atas aplikasi, dimana pengguna yang ingin mencari kata beserta artinya juga dapat melihat hasil *running time* dari masing-masing algoritma.



Gambar 4.14 Tampilan kamus psikologi

4.3.3 Tampilan Menu Favorit

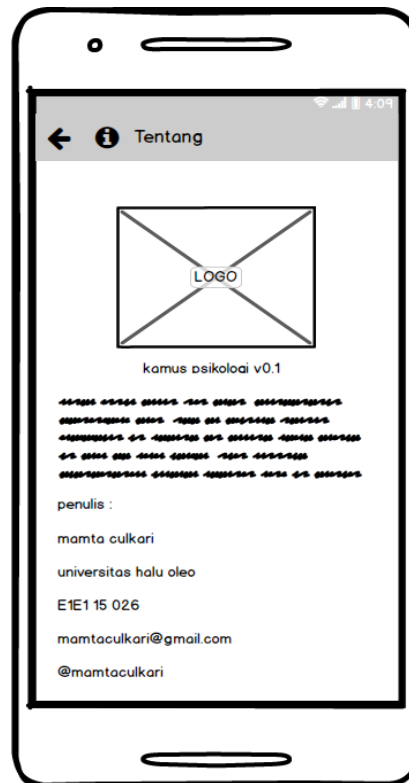
Pada saat *user* memilih menu favorit, sistem akan menampilkan istilah-istilah psikologi yang telah dipilih oleh *user* untuk difavoritkan terlebih dahulu.



Gambar 4.15 Tampilan menu favorit

4.3.4 Tampilan Menu Tentang Aplikasi

Pada saat *user* memilih menu tentang aplikasi, sistem akan menampilkan deskripsi aplikasi. Disini akan ditampilkan biodata dari pembuat aplikasi.



Gambar 4.16 Tampilan menu tentang aplikasi

4.4 Metode Pengujian

Pengujian merupakan metode yang dilakukan untuk menjelaskan mengenai kebenaran, kehandalan, dan keakurasian yang terdiri dari perangkat pengujian, metode pengujian dan pelaksanaan pengujian. Pengujian program ini menggunakan metode *black box*, pengujian 1 kata, pengujian 2 kata, pengujian 3 kata, dan pengujian kata yang tidak terdapat dalam aplikasi serta kompleksitas algoritma (*big O*).

Pengujian *black box* merupakan pengujian program berdasarkan fungsi dari program. Tujuan dari metode *black box* ini adalah untuk menemukan kesalahan fungsi pada program. Pengujian dengan metode ini dilakukan dengan cara memberikan sejumlah *input* pada program aplikasi yang kemudian diproses sesuai dengan kebutuhan fungsionalnya untuk melihat apakah program aplikasi menghasilkan keluaran yang diinginkan dan sesuai dengan fungsi dari program tersebut. Apabila dari masukan yang diberikan proses menghasilkan keluaran yang

sesuai dengan kebutuhan fungsionalnya, maka program aplikasi yang bersangkutan telah benar, tetapi jika keluaran yang dihasilkan tidak sesuai dengan kebutuhan fungsionalnya, maka masih terdapat kesalahan pada program aplikasi.

Kompleksitas dari suatu algoritma merupakan ukuran seberapa banyak komputasi yang dibutuhkan algoritma tersebut untuk menyelesaikan masalah. Secara informal, algoritma yang dapat menyelesaikan suatu permasalahan dalam waktu yang singkat memiliki kompleksitas yang rendah, sementara algoritma yang membutuhkan waktu lama untuk menyelesaikan masalahnya mempunyai kompleksitas yang tinggi. Kompleksitas algoritma terdiri dari dua macam yaitu kompleksitas waktu dan kompleksitas ruang (Azizah, 2013). Kompleksitas waktu untuk algoritma-algoritma yang dibahas akan dinyatakan dengan notasi O besar (*Big-O notation*). Notasi menyatakan *running time* dari suatu algoritma untuk kemungkinan kasus terburuk. Notasi memiliki dari beberapa bentuk. Notasi dapat berupa salah satu bentuk maupun kombinasi dari bentuk-bentuk tersebut. (Rahayuningsih 2016).

Pengujian dilakukan dengan mencoba semua kemungkinan yang terjadi. Jika dalam pengujian ditemukan kesalahan, maka akan dilakukan penelusuran dan perbaikan untuk memperbaiki kesalahan yang terjadi. Jika telah selesai melakukan perbaikan, maka akan dilakukan pengujian kembali. Pengujian dan perbaikan dilakukan secara terus menerus hingga diperoleh hasil yang terbaik.

4.5 Rencana Pengujian

Pengujian perangkat lunak berikut menggunakan data uji berdasarkan data dari masing-masing data. Rencana selengkapnya dapat dilihat pada Tabel 4.4 berikut:

Tabel 4.4 Rencana pengujian

Kelas Uji	Detail Uji	Jenis Pengujian
Pengujian Menu	Beranda	<i>Black Box</i>
	Cari Istilah	<i>Black Box</i>
	Petunjuk	<i>Black Box</i>
	Pembuat	<i>Black Box</i>
	Tentang	<i>Black Box</i>
Pengujian Algoritma <i>Horspool</i>	Pencarian data yang ada di <i>database</i>	<i>Black Box</i>
	Pencarian data yang tidak ada di <i>database</i>	<i>Black Box</i>
Pengujian Algoritma <i>Raita</i>	Pencarian data yang ada di <i>database</i>	<i>Black Box</i>
	Pencarian data yang tidak ada di <i>database</i>	<i>Black Box</i>

BAB V

IMPLEMENTASI DAN PEMBAHASAN

Implementasi merupakan tahapan pengembangan rancangan menjadi kode program. Pada awal bagian ini dijabarkan spesifikasi perangkat keras dan perangkat lunak serta implementasi yang dilakukan berdasarkan hasil pada tahap perancangan. Penjelasan pada bab ini meliputi lingkungan pengembangan yang digunakan, batasan implementasi dan proses, serta hasil implementasi dari aplikasi.

5.1 Implementasi *Interface*

Implementasi *interface* dari perangkat lunak dilakukan berdasarkan perancangan *user interface* yang telah dijelaskan, namun disesuaikan dengan komponen-komponen *view* yang tersedia pada *framework android*. Implementasi antarmuka ditampilkan dalam bentuk *screen shoot* dari *handphone android*.

1. *Splash Screen* Aplikasi Kamus Istilah Psikologi

Pada saat aplikasi dibuka yang akan muncul pertama kali adalah *splash screen*. Pada tampilan *interface* ini, *splash screen* akan menampilkan nama dan logo aplikasi.



Gambar 5.1 *Splash screen* aplikasi kamus istilah psikologi

2. Beranda Aplikasi Kamus Istilah Psikologi

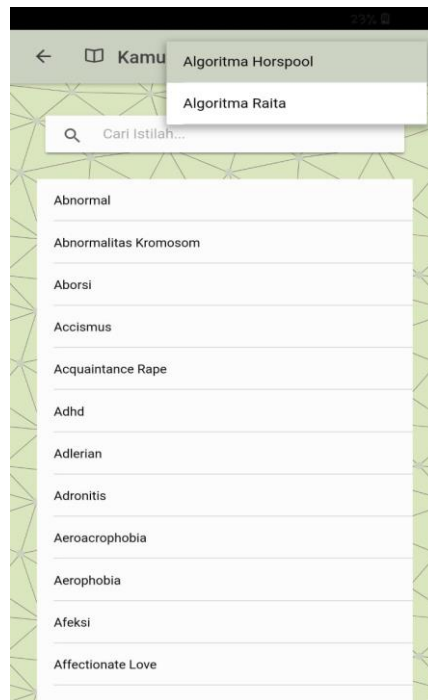


Gambar 5.2 Beranda aplikasi kamus istilah psikologi

Halaman depan aplikasi merupakan implementasi dari *interface* beranda yang telah dirancang. *Interface* ini menampilkan menu utama aplikasi Kamus Istilah Psikologi yaitu menu Kamus Istilah, menu Favorit dan menu Tentang Aplikasi. Pada menu Kamus Istilah, *user* dapat memilih salah satu metode yang dapat digunakan dalam pencarian kata istilah pada bagian kanan atas halaman.

3. *Interface* menu Kamus Psikologi

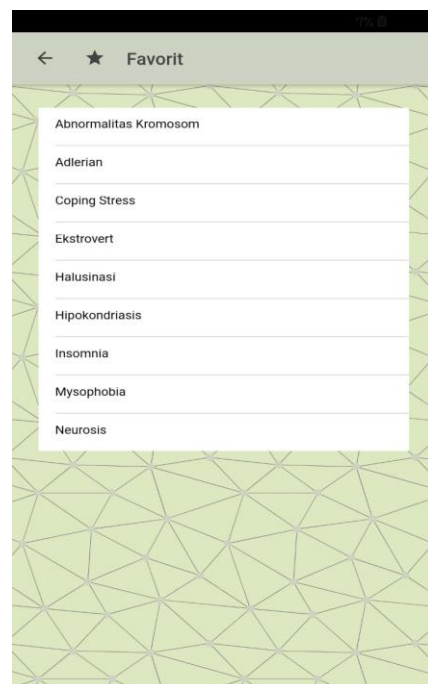
Interface menu Kamus Psikologi merupakan halaman menu pencarian kata istilah. Pada halaman ini *user* cukup memasukkan kata yang ingin dicari pada kolom pencarian dan menekan tombol cari. Pada halaman ini juga *user* dapat memilih salah satu metode pencarian yang ingin digunakan yaitu Algoritma Horspool atau Algoritma Raita.



Gambar 5.3 Interface menu kata istilah

4. **Interface menu Favorit**

Pada menu favorit, *user* dapat melihat kata-kata istilah yang telah difavoritkan terlebih dahulu oleh *user*.



Gambar 5.4 Interface menu favorit

5. *Interface* menu Tentang Aplikasi

Pada menu tentang aplikasi, sistem akan menampilkan deskripsi aplikasi dan menu ini juga menampilkan profil dari pembuat aplikasi.



Gambar 5.5 *Interface* menu tentang aplikasi

5.2 Implementasi Algoritma

5.2.1 Implementasi Algoritma *Horspool*

Gambar 5.6 merupakan *source code* Algoritma *Horspool*. *Source code* ini digunakan dalam aplikasi sebagai pencarian menggunakan Algoritma *Horspool*.

```
horspoolFunction(keyword){
  let horspoolResult: any = [];
  for(let key in this.data_kamus){
    let text =
this.data_kamus[key]['kata'].toLowerCase().replace(/[
^0-9a-z ]/gi, '').split('');
    let pattern = keyword.toLowerCase().split('');
    let index = pattern.length-1;
let bmbc = []; let bm = []; let bc = [];
    for (let i = 0; i < pattern.length; i++) {
      let bcx = pattern[i];
      let bmx = pattern.length-i-1;
      if(bmx==0) bmx = pattern.length;
```

```

        bmbc.push({'bc' : bcx, 'bm' : bmx});
        bm.push(bmx)
        bc.push(bcx)
    }

    let jump = 0;
    while(index < text.length){
        let check = 0;

    if(bmbc[Object.keys(bmbc)[Object.keys(bmbc).length-1]]['bc']!=text[index]){
        if(bc.indexOf(text[index])>-1){
            jump = bmbc[bc.indexOf(text[index))]['bm']
        }else{
            jump =
bmbc[Object.keys(bmbc)[Object.keys(bmbc).length-1]]['bm'];
        }
    }else{
        let x = 0;
        for (let key = pattern.length; key > 0; key--) {
            if(pattern[key]==text[index-x]){
                check++;
            }else{
                check = 0;
                if(bc.indexOf(text[index])>-1){
                    jump = bmbc[bc.indexOf(text[index))]['bm']
                }else{
                    jump =
bmbc[Object.keys(bmbc)[Object.keys(bmbc).length-1]]['bm']; }
                break;
            }
            x++;
        }
        if(check==pattern.length) {
            horspoolResult.push(this.data_kamus[key])
            break;
        }
        index+=jump;
    }
    }
    return horspoolResult;
}

```

Gambar 5.6 Tampilan source code Algoritma Horspool

5.2.2 Implementasi Algoritma *Raita*

Gambar 5.7 merupakan *source code* Algoritma *Raita*, dimana *source code* ini digunakan dalam pencarian Algoritma *Raita* pada aplikasi Kamus Istilah Psikologi.

```

raitaFunction(keyword){
    let raitaResult: any = [];

    for(let key in this.data_kamus){
        let text =
this.data_kamus[key]['kata'].toLowerCase().replace(/[\
^0-9a-z ]/gi, '').split('');
        let pattern = keyword.toLowerCase().split('');

        let index = pattern.length-1;

        let bmbc = [];
        let bm = [];
        let bc = [];
        for (let i = 0; i < pattern.length; i++) {
            let bcx = pattern[i];
            let bmx = pattern.length-i-1;
            if(bmx==0) bmx = pattern.length;
            bmbc.push({'bc' : bcx, 'bm' : bmx});
            bm.push(bmx)
            bc.push(bcx)
        }

        let jump = 0;
        while(index < text.length){
            let check = 0;

            if(bmbc[Object.keys(bmbc)[Object.keys(bmbc).length-1]]['bc']!=text[index]){
                if(bc.indexOf(text[index])>-1){
                    jump = bmbc[bc.indexOf(text[index))]['bm']
                }else{
                    jump =
bmbc[Object.keys(bmbc)[Object.keys(bmbc).length-1]]['bm'];
                }
            }else{
                check++;
                if(pattern.length==1){
                    raitaResult.push(this.data_kamus[key])
                }
            }
        }
    }
}

```

```

        break;
    }

    if(pattern[0]==text[index-(pattern.length-1)]){
        check++;
        let key = 0;
        let middle = 0;
        let odd = 0;
        let even = 0;

        while(key<(pattern.length)-2){
            let patternIndex;
            let textIndex;
            let oddEvenCheck = pattern.length%2==0 ?1:0;

            if(middle%2==0){
                patternIndex =
Math.floor(((pattern.length) / 2)+(middle-odd));
                textIndex = Math.floor((((pattern.length)
/ 2)-(middle-odd))-oddEvenCheck);
                odd++;

            }else{
                patternIndex =
Math.floor(((pattern.length) / 2)-(middle-even));
                textIndex = Math.floor((((pattern.length)
/ 2)+(middle-even))-oddEvenCheck);
                even++;
            }

            if(pattern[patternIndex]==text[index-
textIndex]){
                check++;
                middle++;
                key++;
            }

            else{
                check=0;
                if(bc.indexOf(text[index])>-1){
                    jump = bmbc[bc.indexOf(text[index])]['bm']
                }else{
                    jump =
bmbc[Object.keys(bmbc)[Object.keys(bmbc).length-
1]]['bm'];
                }
                break;
            }
        }
    }

```

```

    }
}

}else{
    check=0;
    jump =
bmbc[Object.keys(bmbc)[Object.keys(bmbc).length-
1]]['bm'];
    break;
}

if(check==pattern.length){
    raitaResult.push(this.data_kamus[key])
    break;
}
}

index+=jump;
}
}
return raitaResult;
}
}

```

Gambar 5.7 Tampilan *source code* Algoritma Raita

5.3 Pengujian Sistem

Pengujian merupakan tahap yang utama dalam pembuatan suatu aplikasi. Hasil dari pengujian yang didapat akan dijadikan sebagai tolak ukur dalam proses pengembangan selanjutnya. Pengujian ini dilakukan dengan memasukkan sebuah kata yang akan dicari ke dalam aplikasi untuk menguji performa dari aplikasi serta kecepatan aplikasi dalam mencari kata istilah.

5.3.1 Pengujian *Black Box*

Pengujian *black box* dilakukan dengan menguji perangkat lunak dari segi fungsionalitas perangkat lunak. Fungsionalitas perangkat lunak yang diuji sesuai dengan *use case* pada tahap desain. Peneliti membagi pengujian menjadi enam bagian. Setiap bagian diuji sesuai dengan skenario *use case* pada tahap desain.

Tabel 5.1 Pengujian *Black Box*

Input/ Event	Proses	Output	Hasil Uji
Membuka Aplikasi	Proses masuk pada aplikasi kamus istilah psikologi	Menampilkan <i>splash screen</i>	Sesuai
Memilih Menu Kamus Istilah	Menampilkan kata istilah psikologi	Kata-kata istilah psikologi ditampilkan	Sesuai
Memilih Menu Favorit	Menampilkan Kata Istilah yang telah ditambahkan ke favorit	Kata-kata istilah yang telah difavoritkan ditampilkan	Sesuai
Memilih Menu Tentang Aplikasi	Menampilkan Menu Tentang Aplikasi	Menampilkan tentang aplikasi dan biodata pembuat aplikasi	Sesuai
Pencarian Kata	Memasukkan kata yang ingin dicari dan menekan tombol enter	Menampilkan hasil pencarian	Sesuai

5.3.2 Pengujian Pencarian dengan Menggunakan 1 Kata

Pengujian ini dilakukan untuk memeriksa ketepatan dan kecepatan pencarian kata istilah psikologi yang akan dicari oleh user. Pengujian ini dilakukan dengan memasukkan satu kata yang akan dicari kedalam *form* aplikasi untuk menguji performa dari aplikasi.

Tabel 5.2 Pengujian menggunakan 1 kata (Algoritma *Horspool*)

No	Kata yang dicari	Percobaan Algoritma <i>Horspool</i> ke-										Rata-rata (ms)
		1	2	3	4	5	6	7	8	9	10	
1	Normal	16.7	14.5	14.4	16.4	14.7	15.5	15.9	15.1	14.9	14.0	15.21
2	Phobia	15.2	16.6	15.3	15.9	15.2	17.8	15.9	14.9	19.5	18.2	16.45
3	Hyper	15.6	16.5	17.8	16.7	15.5	14.9	17.7	15.0	17.9	17.0	16.46
4	Impulsif	15.6	15.6	16.8	15.6	15.3	14.9	13.1	14.8	14.4	15.6	15.17
5	Liberosis	18.9	18.5	17.9	18.1	17.1	15.5	16.9	18.4	18.8	18.2	17.83
6	Somnabulisme	12.1	12.0	14.4	14.2	12.0	12.1	13.3	13.2	14.5	14.5	13.23
7	Narkolepsi	16.7	13.0	14.6	13.2	14.4	13.6	13.1	13.3	17.8	16.6	14.63
8	Hipnotis	16.7	16.7	16.9	18.4	16.3	16.9	15.4	16.2	15.1	16.4	16.50
9	Vellichor	18.1	17.5	19.0	18.4	19.0	19.3	17.0	19.0	19.4	18.2	18.49
10	Zydis	16.4	17.7	15.9	16.6	15.4	14.6	15.6	16.4	17.5	15.2	16.13
Rata-rata :												16.01

Tabel 5.3 Pengujian menggunakan 1 kata (Algoritma *Raita*)

No.	Kata yang dicari	Percobaan Algoritma <i>Raita</i> Ke-										Rata-rata (ms)
		1	2	3	4	5	6	7	8	9	10	
1	Normal	15.2	15.3	15.6	14.6	15.2	16.6	15.3	17.2	16.4	16.6	15.80
2	Phobia	14.2	14.1	13.6	14.6	14.2	15.5	15.8	14.7	14.5	15.7	14.69
3	Hyper	14.7	15.6	15.2	15.9	16.2	17.4	16.8	16.9	16.5	15.8	16.10
4	Impulsip	12.6	13	11.8	12.4	12.4	13.3	11.3	11.8	11.1	10.7	12.04
5	Liberosis	13.4	13.6	13.8	14.7	14.3	15.4	16.4	16.8	17.2	16.6	15.22
6	Somnabulisme	10.8	10.5	10.9	11	10.7	10.9	10.3	11.1	10	11.6	10.78
7	Narkolepsi	12.3	13.7	12.8	12.6	14.3	14.8	12.7	14.5	15.8	14.1	13.76
8	Hipnotis	17.5	18.8	17.7	17.1	18.5	18.7	18.9	16.4	18.4	17.1	17.91
9	Vellichor	18.4	17.8	18.3	18.8	19.2	17.7	17.2	17.3	17.4	18.2	18.03
10	Zydis	14.4	13.6	15.8	14	14.5	13.1	14.7	13.8	15.1	15.6	14.46
Rata-rata :												14.88

Berdasarkan pengujian pencarian dengan menggunakan satu kata pada masing-masing istilah diperoleh hasil rata-rata waktu untuk Algoritma *Horspool* adalah 16.01 ms, sedangkan untuk Algoritma *Raita* adalah 14.88 ms. Berdasarkan hasil tersebut pada pengujian pencarian dengan menggunakan satu kata, Algoritma *Raita* lebih cepat dibandingkan Algoritma *Horspool* dengan selisih waktu kecepatan 1.13 ms.

5.3.3 Pengujian Pencarian dengan Menggunakan 2 Kata

Pengujian ini dilakukan untuk memeriksa ketepatan dan kecepatan pencarian kata istilah psikologi yang akan dicari oleh *user*. Pengujian ini dilakukan dengan memasukkan sepuluh kata yang akan dicari kedalam *form* aplikasi untuk menguji performa dari aplikasi.

Tabel 5.4 Pengujian menggunakan 2 kata (Algoritma *Horspool*)

No.	Kata yang dicari	Percobaan Algoritma <i>Horspool</i> Ke-										Rata-rata (ms)
		1	2	3	4	5	6	7	8	9	10	
1	Art Therapy	17.6	16.2	17.3	18.0	16.4	16.2	15.0	16.5	17.8	16.2	16.72
2	Bettelheim Bruno	12.1	12.7	12.7	13.5	12.8	12.6	12.0	12.6	12.4	13.3	12.67
3	Down Syndrome	14.8	13.4	14.8	13.8	14.8	15.3	15.1	15.3	16.0	14.4	14.77
4	Episodic Memory	15.6	13.3	13.0	15.1	12.6	13.8	13.2	13.6	13.7	12.1	13.60
5	Identitas Gender	15.5	11.4	15.7	11.9	12.5	13.6	13.2	11.1	12.7	13.9	13.15
6	Insanity Defense	13.4	14.0	12.9	12.8	11.5	13.2	12.7	11.9	12.4	12.3	12.71
7	Jet Lag	21.0	21.4	20.1	23.0	21.3	21.6	20.6	20.3	20.4	21.4	21.11
8	Usher Syndrome	15.78	14.2	14.2	14.6	17.0	14.6	14.5	14.6	14.1	15.0	14.86
9	Waking Hypnosis	14.9	13.8	13.9	13.4	13.5	16.9	13.3	13.1	12.9	17.8	14.35
10	Zeigarnik Effect	14.2	14.4	13.2	14.4	13.5	13.5	12.7	12.2	13.4	12.9	13.44
Rata-rata :												14.74

Tabel 5.5 Pengujian menggunakan 2 kata (Algoritma *Raita*)

No.	Kata yang dicari	Percobaan Algoritma <i>Raita</i> Ke-										Rata-rata (ms)
		1	2	3	4	5	6	7	8	9	10	
1	Art Therapy	12.1	13.2	13.7	15.6	14.2	15.2	13.4	14.2	12.3	12.1	13.60
2	Bettelheim Bruno	13.4	14.7	13.5	14.9	15.2	15.6	14.3	14.1	14.6	15.3	14.56
3	Down Syndrome	14.3	14.9	14.8	13.7	15.1	14.2	14.1	15.3	15.4	15.2	14.70
4	Episodic Memory	12.9	13.7	13.8	10.5	12.3	12.5	13.7	13.2	13.5	13.6	12.50
5	Identitas Gender	11.6	11.4	11.7	11.8	12.1	12	13.3	12.4	12	11.9	12.02
6	Insanity Defense	12.5	12.2	12.3	12.6	12.1	11.2	10.2	12.7	11.5	11.8	11.91
7	Jet Lag	18.7	18.6	17.2	16.6	17.3	15.1	15.8	15.78	15.8	15.8	16.67
8	Usher Syndrome	15.2	14.4	15.9	16.1	14.6	16.3	15.7	14.8	14	14.2	15.12
9	Waking Hypnosis	13.7	14.9	15.3	14.8	14.4	15.2	13.1	13.5	14.3	13.3	14.25
10	Zeigarnik Effect	13.7	12.5	11.9	13	12.9	12.5	13.3	13	13.2	12.6	12.86
Rata-rata :												13.82

Berdasarkan pengujian pencarian dengan menggunakan dua kata pada masing-masing istilah diperoleh hasil rata-rata waktu untuk Algoritma *Horspool* adalah 14.74 ms, sedangkan untuk Algoritma *Raita* adalah 13.82 ms. Berdasarkan hasil tersebut pada pengujian pencarian dengan menggunakan dua kata, Algoritma *Raita* lebih cepat dibandingkan Algoritma *Horspool* dengan selisih waktu kecepatan yang sangat kecil yaitu 0.92 ms.

5.3.4 Pengujian Pencarian dengan 3 Kata yang Sama

Pengujian ini dilakukan untuk memeriksa ketepatan dan kecepatan pencarian kata istilah psikologi yang akan dicari oleh *user*. Pengujian ini dilakukan

Tabel 5.7 Pengujian menggunakan 3 kata (Algoritma *Raita*)

No.	Kata yang dicari	Percobaan Algoritma <i>Raita</i> Ke-										Rata-rata (ms)
		1	2	3	4	5	6	7	8	9	10	
1	Battered Child Syndrome	10.0	11.8	12.0	10.2	11.7	12.3	12.7	12.7	11.5	11.2	11.61
2	Client Centered Therapy	10.4	10.9	11.9	11.2	12.7	10.5	12.0	10.3	11.4	11.1	11.24
3	Free Radical Theory	10.1	10.7	11.3	12.7	12.3	11.7	11.2	11.8	12.2	11.1	11.51
4	Maternal Blood Screening	11.7	11.5	10.8	10.8	11.3	11.1	11.5	11.1	11.5	10.2	11.15
5	Normative Life Event	11.5	12.2	12.5	13.0	12.2	11.5	10.2	11.3	10.3	11.5	11.62
6	Post Partum Depression	11.8	10.8	11.5	11.3	11.0	11.8	11.5	12.2	11.5	11.9	11.53
7	Short Term Memory	11.9	12.5	12.6	12.2	12.2	12.4	11.0	11.3	12.9	12.1	12.11
8	Stability Change Issue	11.5	11.4	12.0	11.1	11.2	11.4	10.5	11.6	10.6	11.3	11.26
9	Uncued Panic Attack	9.7	12.1	12.2	10.1	12.8	12.2	11.1	12.2	11.1	11.3	11.48
10	Weapon Fokus Effect	10.4	11.5	11.6	10.4	11.2	12.1	11.8	10.1	11.2	10.4	11.07
Rata-rata :												11.46

Berdasarkan pengujian pencarian dengan menggunakan tiga kata pada masing-masing istilah diperoleh hasil rata-rata waktu untuk Algoritma *Horspool* adalah 11.95 ms, sedangkan untuk Algoritma *Raita* adalah 11.46 ms. Berdasarkan hasil tersebut pada pengujian pencarian dengan menggunakan tiga kata, Algoritma *Raita* lebih cepat dibandingkan Algoritma *Horspool* dengan selisih waktu kecepatan 0.49 ms.

No.	Kata yang dicari	Percobaan Algoritma <i>Horspool</i> Ke-										Rata-rata (ms)
		1	2	3	4	5	6	7	8	9	10	
1	Depreszions	14.0	15.2	15.8	14.6	15.5	16.3	15.5	16.3	16.3	16.5	15.60
2	Pengobatan	16.0	16.5	16.8	15.8	15.2	17.6	16.9	16.8	16.8	15.3	16.37
3	Coding	18.2	20.1	20.2	18.4	17.2	18.5	18.4	18.3	20.6	18.3	18.82
4	Gangguan saraf	13.1	13.9	14.5	13.5	14.6	14.7	13.8	13.5	13.3	13.8	13.87
5	Saraf Trjepit	14.6	15.0	13.9	13.8	13.6	14.1	13.8	13.8	14.4	14.9	14.19
6	Gila	25.2	26.7	25.3	26.0	23.2	26.8	25.3	25.1	25.0	25.4	25.40
7	Terapi Keterbelakangan Mental	13.3	11.9	10.7	12.7	15.7	11.5	11.4	11.5	12.0	11.5	12.22
8	Obat Saraf	15.79	15.2	15.6	17.7	16.3	16.6	17.1	17.9	15.0	16.9	16.48
9	Pendeteksi Kejiwaan	12.7	12.1	12.0	11.8	12.2	12.7	12.6	12.2	13.0	12.7	12.40
10	Medical	20.3	19.3	19.4	20.7	18.0	21.0	15.9	17.1	20.6	19.6	19.19
Rata-rata:												16.45

**Tabel 5.9 Pengujian pencarian kata yang tidak terdapat pada aplikasi
(Algoritma *Raita*)**

No.	Kata yang dicari	Percobaan Algoritma <i>Raita</i> Ke-										Rata-rata (ms)
		1	2	3	4	5	6	7	8	9	10	
1	Depreszions	12.3	13.8	12.7	13.3	13.7	12.7	12.5	14.4	15.8	15.6	13.68
2	Pengobatan	18.1	18.5	18.0	16.7	17.2	17.4	17.7	16.5	17.7	17.3	17.51
3	Coding	12.5	12.4	12.7	13.9	13.6	14.7	14.8	13.2	13.6	14.5	13.59
4	Gangguan saraf	12.7	13.7	13.4	13.7	14.1	13.0	12.7	11.2	12.7	12.9	13.01
5	Saraf Terjepit	13.2	13.2	14.1	14.9	14.9	14.1	14.9	12.9	13.1	13.5	13.88
6	Gila	15.4	16.4	17.3	17.1	14.2	14.9	16.7	17.6	17.2	15.9	16.27
7	Terapi Keterbelakangan Mental	11.3	11.5	10.9	12.1	13.5	15.0	11.5	12.5	12.3	12.1	12.27
8	Obat Saraf	13.3	14.3	15.3	12.5	13.3	14.0	13.4	15.4	15.3	14.7	14.15
9	Pendeteksi Kejiwaan	14.9	14.3	13.7	12.2	11.2	12.9	12.2	12.7	13.2	12.1	12.94
10	Medical	15.5	14.6	15.3	13.4	15.7	14.8	14.0	15.6	13.7	15.9	14.85
Rata-rata:												14.22

Berdasarkan pengujian pencarian kata yang tidak terdapat pada aplikasi. Masing-masing istilah diperoleh hasil rata-rata waktu untuk Algoritma *Horspool* adalah 16.45 ms, sedangkan untuk Algoritma *Raita* adalah 14.22 ms. Berdasarkan hasil tersebut pada pengujian pencarian pada kata yang tidak terdapat pada aplikasi, Algoritma *Raita* lebih cepat dibandingkan Algoritma *Horspool* dengan selisih waktu kecepatan 2.23 ms.

Tabel 5.10 Rata- rata dari semua hasil pengujian pencarian kata (Algoritma *Horspool* dan Algoritma *Raita*)

Algoritma	Pengujian Pencarian Kata			
	1 Kata	2 Kata	3 Kata	Tidak ada kata
<i>Horspool</i>	16.01 <i>ms</i>	14.74 <i>ms</i>	11.95 <i>ms</i>	16.45 <i>ms</i>
<i>Raita</i>	14.88 <i>ms</i>	13.82 <i>ms</i>	11.46 <i>ms</i>	14.22 <i>ms</i>

Keterangan :

Kuning = Lebih tinggi

Hijau = Lebih rendah

Berdasarkan pengujian yang telah dilakukan pada pencarian 1 kata, Algoritma *Horspool* memiliki rata-rata kecepatan waktu selama 16.01 *ms* dan Algoritma *Raita* selama 14.88 *ms*. Pada pencarian 2 kata, Algoritma *Horspool* memiliki rata-rata kecepatan waktu selama 14.74 *ms* dan Algoritma *Raita* selama 13.82 *ms*. Pada pencarian 3 kata, Algoritma *Horspool* memiliki rata-rata kecepatan waktu selama 11.95 *ms* dan Algoritma *Raita* selama 11.46 *ms*, dan untuk pencarian kata yang tidak terdapat dalam aplikasi, Algoritma *Horspool* memiliki rata-rata kecepatan waktu selama 16.45 *ms* dan Algoritma *Raita* selama 14.22 *ms*. Hal ini membuktikan bahwa Algoritma *Raita* melakukan pencarian *string* lebih cepat dibandingkan dengan Algoritma *Horspool*.

Berdasarkan pengujian yang telah dilakukan, semakin banyak jumlah kata yang dicari maka kecepatannya semakin cepat, karena apabila *pattern* lebih panjang daripada kata istilah maka tidak akan dilakukan pengecekan sehingga waktu pencariannya lebih cepat.

5.3.6 Analisis Pengujian Pencarian dengan 1 Kata yang Sama

Pengujian ini dilakukan untuk menganalisa ketepatan dan kecepatan pencarian kata istilah psikologi yang akan dicari oleh *user*. Pengujian ini dilakukan

dengan memasukkan satu kata yang akan dicari kedalam *form* aplikasi untuk menguji performa dari aplikasi dan menganalisanya.

Tabel 5.11 Pengujian menggunakan 1 kata yang sama

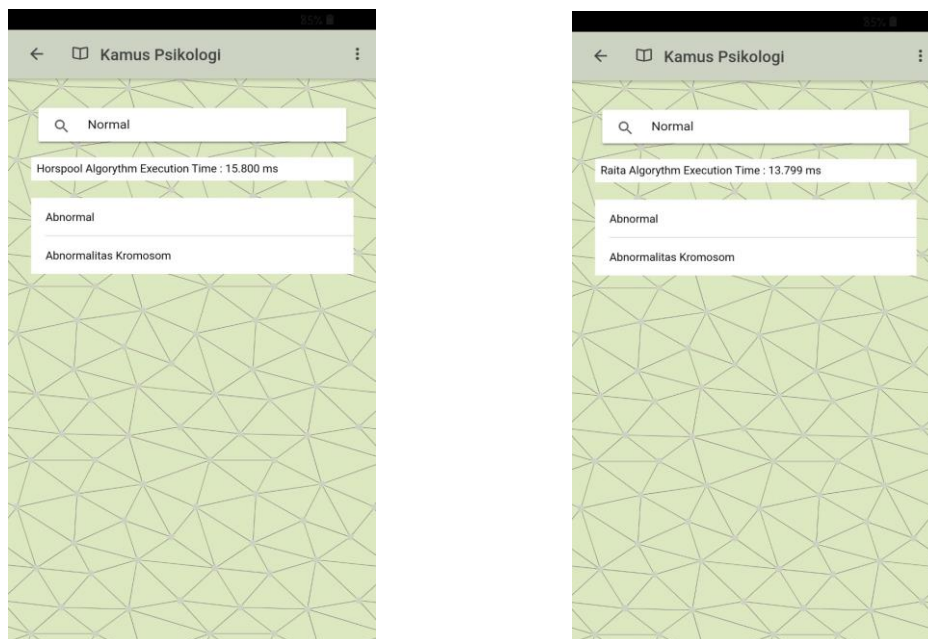
Kata (Zoophobia)	Algoritma <i>Horspool</i>	Algoritma <i>Raita</i>
Percobaan 1	11.79	10.89
Percobaan 2	11.14	11.25
Percobaan 3	11.69	10.33
Percobaan 4	12.32	11.23
Percobaan 5	11.25	10.32
Rata-Rata:	11.64	10.80

Berdasarkan pengujian dilakukan pada satu kata yang sama yaitu Zoophobia Algoritma *Raita* lebih cepat dengan nilai selisih 0.84 *ms* dari Algoritma *Horspool*. Pada kata Zoophobia, Algoritma *Horspool* akan melakukan pengecekan ke semua kata yang mengandung kata phobia dan pada aplikasi kamus istilah terdapat 22 kata yang mengandung kata phobia sedangkan untuk Algoritma *Raita* hanya melakukan pengecekan pada kata akhir dan awal saja sehingga Algoritma *Horspool* pada kata Zoophobia melakukan pencarian lebih lambat dari Algoritma *Raita*.

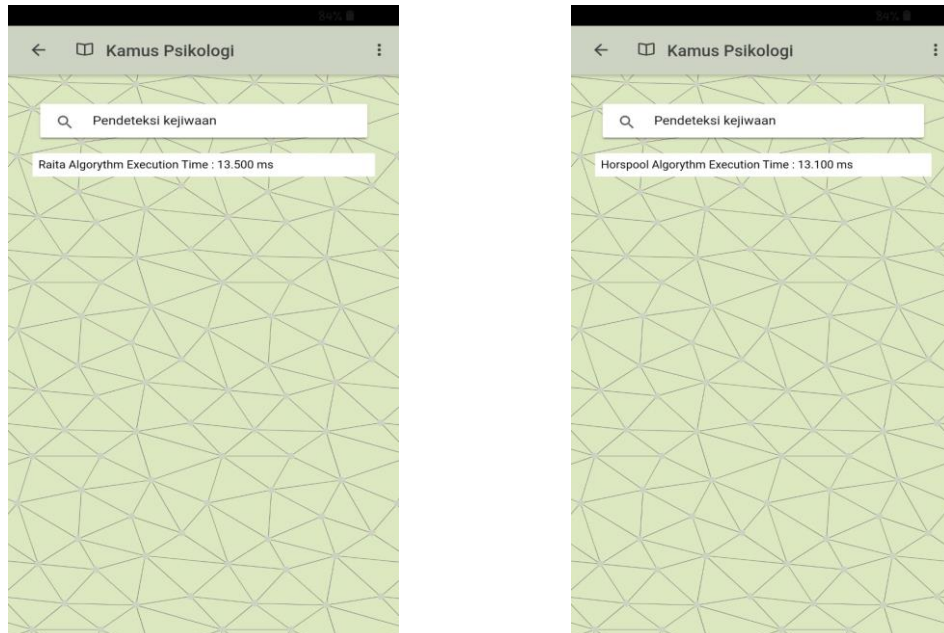
Pada Percobaan 1 sampai 5 didapatkan hasil yang dinamis (berubah-ubah) karena sistem kerja yang perangkat lakukan bukan hanya melakukan pencarian pada aplikasi ini, tetapi banyak *service-service* atau operasi sistem *android* yang berjalan walaupun semua aplikasi telah ditutup. Untuk mendapatkan hasil yang cukup stabil pengujian dilakukan dengan cara menutup semua aplikasi yang berjalan, mematikan jaringan, dan menghapus *cache* pada perangkat pada saat sebelum melakukan pengujian.

Contoh penerapan pada aplikasi :

Gambar 5.8 menunjukkan tampilan kata istilah yang mempunyai kesamaan kata yang dicari, beserta waktu pencarian.



Gambar 5.8 *Interface* hasil pencarian kata normal pada Algoritma *Horspool* dan Algoritma *Raita*



Gambar 5.9 *Interface* hasil pencarian kata yang tidak ditemukan pada Algoritma *Horspool* dan Algoritma *Raita*

Gambar 5.9 menampilkan waktu pencarian apabila tidak ditemukan kata yang cocok dengan kata yang dicari.

5.4 Pengujian Kompleksitas Algoritma

5.4.1 Kompleksitas Algoritma *Horspool*

Tabel 5.12 Kompleksitas Algoritma *Horspool*

HorspoolFunction(keyword) {	C	#	C#
let text = this.data_kamus[key]['kata'].toLowerCase().replace(/[^\0-9a-z]/gi, '').split('');	C1	1	C1
let pattern = keyword.toLowerCase().split('');	C1	1	C1
let index = pattern.length-1;	C1	1	C1
let bmbc = [];	C1	1	C1
let bm = [];	C1	1	C1
let bc = [];	C1	1	C1
for (let i = 0; i < pattern.length; i++) {	C2	N	C2N
let bcx = pattern[i];	C2	N	C2N
let bmx = pattern.length-i-1;	C2	N	C2N
if(bmx==0) bmx = pattern.length;	C2	N	C2N
bmbc.push({'bc' : bcx, 'bm' : bmx});	C2	N	C2N
bm.push(bmx)	C2	N	C2N
bc.push(bcx)	C2	N	C2N
}			
let jump = 0;	C1	1	C1
while(index < text.length){	C3	M	C3M
let check = 0;	C3	M	C3N
if (bmbc[Object.keys(bmbc)[Object.keys(bmbc).length-1]]['bc']!=text[index]){	C3	M	C3M
if(bc.indexOf(text[index])>-1){	C3	M	C3M
jump = bmbc[bc.indexOf(text[index))]['bm']	C3	M	C3M
}else{	C3	M	C3M
jump = bmbc[Object.keys(bmbc)[Object.keys(bmbc).length-1]]['bm'];	C3	M	C3M
}			

<code>}else{</code>	C3	M	C3M
<code>let x = 0;</code>	C3	M	C3M
<code>For (let key = pattern.length; key > 0; key--) {</code>	C4	NM	C4NM
<code>if (pattern[key]==text[index-x]){</code>	C4	NM	C4NM
<code>check++;</code>	C4	NM	C4NM
<code>}else{</code>	C4	NM	C4NM
<code>check = 0;</code>	C4	NM	C4NM
<code>if (bc.indexOf(text[index])>-1){</code>	C4	NM	C4NM
<code>jump = bmbc[bc.indexOf(text[index])] ['bm']</code>	C4	NM	C4NM
<code>}else{</code>	C4	NM	C4NM
<code>jump = bmbc[Object.keys(bmbc)[Object.keys(bmbc). length-1]] ['bm'];</code>	C4	NM	C4NM
<code>}</code>			
<code>break;}</code>			
<code>x++;</code>	C4	NM	C4NM
<code>}</code>			
<code>if (check==pattern.length) {</code>	C3	M	C3M
<code>horspoolResult.push(this.data_kamus[key])</code>	C3	M	C3M
<code>break;</code>			
<code>}</code>			
<code>}</code>			
<code>index+=jump;</code>	C3	M	C3M
<code>}</code>			
<code>}</code>			
<code>return horspoolResult;</code>	C1	1	C1
<code>}</code>			

Keterangan :

C = Konstanta

= Ukuran masukan

C.# = Kompleksitas waktu

N = *Pattern Length*

M = *Text Length*

$$\begin{aligned}
T\Theta &= (8C1 + 7C2N + 12C3M + 10C4NM) \\
&= 0 + N + M + NM \\
&= \Theta(NM)
\end{aligned}$$

Tabel 5.12 adalah tabel kompleksitas Algoritma *Horspool*, dimana proses pencarian kompleksitasnya menggunakan bahasa *typescript*, C sebagai konstanta, # sebagai ukuran masukan, dan C. # (C kali #) adalah untuk mencari *Theoretical Running Time* (T(n)) atau kompleksitas waktu, sehingga dapat dijumlahkan hasil dari perkalian C kali #, maka diperoleh hasil $\theta(NM)$.

5.4.2 Kompleksitas Algoritma *Raita*

Tabel 5.13 Kompleksitas Algoritma *Raita*

<code>raitaFunction(keyword) {</code>	C	#	C#
<code>let text = this.data_kamus[key]['kata'].toLowerCase() .replace(/[^\0-9a-z]/gi, '').split('');</code>	C1	1	C1
<code>let pattern = keyword.toLowerCase().split('');</code>	C1	1	C1
<code>let index = pattern.length-1;</code>	C1	1	C1
<code>let bmbc = [];</code>	C1	1	C1
<code>let bm = [];</code>	C1	1	C1
<code>let bc = [];</code>	C1	1	C1
<code>for (let i = 0; i < pattern.length; i++) {</code>	C2	N	C2N
<code>let bcx = pattern[i];</code>	C2	N	C2N
<code>let bmx = pattern.length-i-1;</code>	C2	N	C2N
<code>if(bmx==0) bmx = pattern.length;</code>	C2	N	C2N
<code>bmbc.push({'bc' : bcx, 'bm' : bmx});</code>	C2	N	C2N
<code>bm.push(bmx)</code>	C2	N	C2N
<code>bc.push(bcx)</code>	C2	N	C2N
<code>}</code>			
<code>let jump = 0</code>	C1	1	C1

<code>while(index < text.length){</code>	C3	M	C3M
<code>let check = 0;</code>	C3	M	C3M
<code>if(bmbc[Object.keys(bmbc)[Object.keys(bmbc).length-1]]['bc']!=text[index]){</code>	C3	M	C3M
<code>if(bc.indexOf(text[index])>-1){</code>	C3	M	C3M
<code>Jump = bmbc[bc.indexOf(text[index))]['bm']</code>	C3	M	C3M
<code>}else{</code>	C3	M	C3M
<code>jump = bmbc[Object.keys(bmbc)[Object.keys(bmbc).length-1]]['bm'];</code>	C3	M	C3M
<code>}</code>	C3	M	C3M
<code>}else{</code>	C3	M	C3M
<code>check++;</code>	C3	M	C3M
<code>if(pattern.length==1){</code>	C3	M	C3M
<code>raitaResult.push(this.data_kamus[key])</code>	C3	M	C3M
<code>break;</code>			
<code>}</code>			
<code>if(pattern[0]==text[index-(pattern.length-1)]){</code>	C3	M	C3M
<code>check++;</code>	C3	M	C3M
<code>let key = 0;</code>	C3	M	C3M
<code>let middle = 0;</code>	C3	M	C3M
<code>let odd = 0;</code>	C3	M	C3M
<code>let even = 0;</code>	C3	M	C3M
<code>while(key<(pattern.length)-2){</code>	C4	M(N-2)	C4M(N-2)
<code>let textIndex;</code>	C4	M(N-2)	C4M(N-2)
<code>let oddEvenCheck = pattern.length%2==0 ? 1:0;</code>	C4	M(N-2)	C4M(N-2)
<code>if(middle%2==0){</code>	C4	M(N-2)	C4M(N-2)
<code>patternIndex = Math.floor((pattern.length) / 2)+(middle-odd));</code>	C4	M(N-2)	C4M(N-2)

textIndex = Math.floor(((pattern.length) / 2) - (middle-odd)) - oddEvenCheck);	C4	M(N-2)	C4M(N-2)
odd++;	C4	M(N-2)	C4M(N-2)
}else{	C4	M(N-2)	C4M(N-2)
patternIndex = Math.floor(((pattern.length) / 2) - (middle-even));	C4	M(N-2)	C4M(N-2)
textIndex = Math.floor(((pattern.length) / 2) + (middle-even)) - oddEvenCheck);	C4	M(N-2)	C4M(N-2)
even++;	C4	M(N-2)	C4M(N-2)
}			
if (pattern[patternIndex] == te xt[index-textIndex]) {	C4	M(N-2)	C4M(N-2)
, check++;	C4	M(N-2)	C4M(N-2)
middle++;	C4	M(N-2)	C4M(N-2)
key++;	C4	M(N-2)	C4M(N-2)
}else{	C4	M(N-2)	C4M(N-2)
check=0;	C4	M(N-2)	C4M(N-2)
jump = bmbc[Object.keys(bmbc)[Object.keys(bmb c).length-1]]['bm'];	C4	M(N-2)	C4M(N-2)
break;			
}			
}else{	C4	M(N-2)	C4M(N-2)
check=0;	C4	M(N-2)	C4M(N-2)
jump = bmbc[Object.keys(bmbc)[Object.keys(bmb c).length-1]]['bm'];	C4	M(N-2)	C4M(N-2)
break;			
}			
if (check == pattern.length) {	C3	M	C3M
raitaResult.push(this.data_ka mus[key])	C3	M	C3M
break;			

}			
index+=jump;	C3	M	C3M
}			
}			
}			
return raitaResult;	C1	1	C1
}			

Keterangan :

C = Konstanta

= Ukuran masukan

C . # = Kompleksitas waktu

N = *Pattern Length*

M = *Text Length*

$$T\Theta = (8C1 + 7C2N + 21C3M + 21C4M(N-2))$$

$$= 0 + N + M + C4M(N-2)$$

$$= \Theta(M(N-2))$$

Tabel 5.13 adalah tabel kompleksitas Algoritma *Raita* , dimana proses pencarian kompleksitasnya menggunakan bahasa *typescript*, C sebagai konstanta, # sebagai ukuran masukan, dan C . # (C kali #) adalah untuk mencari *Theoretical Running Time* (T(n)) atau kompleksitas waktu, sehingga dapat dijumlahkan hasil dari perkalian C kali #, maka diperoleh hasil $\theta(M(N-2))$ untuk seluruh proses algoritmanya.

Berdasarkan pengujian kompleksitas algoritma antara Algoritma *Horspool* dan Algoritma *Raita*, Algoritma *Horspool* memiliki hasil kompleksitas yaitu $T(n)=\Theta(MN)$ sedangkan Algoritma *Raita* memiliki hasil kompleksitas yaitu $T(n)=\Theta(M(N-2))$. Dari hasil kompleksitas algoritma tersebut, Algoritma *Raita* lebih baik dibandingkan Algoritma *Horspool* pada pencarian kata karena dalam hal komputasi Algoritma *Raita* lebih pendek dalam menyelesaikan pencarian pada kasus terburuk.

DAFTAR PUSTAKA

- Azizah, Nur Ulfah, 2013, Perbandingan Detektor Tepi Prewit dan Detektor Tepi Laplacian Berdasarkan Kompleksitas Waktu dan Citra Hasil, Universitas Pendidikan Indonesia.
- Badan Pengembangan dan Pendidikan Bahasa Kementrian Pendidikan dan Kebudayaan, 2010, Glosarium, <http://badanbahasa.kemdikbud.go.id/lamanbahasa/produk/890>, diakses tanggal 9 November 2018.
- Baeza-Yates, R.A. & Regnier, M. 1992. Average running time of the boyer-moorehorspool algorithm. *Journal Theoretical Computer Science* Vol. 92, No. 1: 1931.
- Charras, C. & Lecroq, T. 1997. *Handbook of Exact String Matching Algorithms*. Oxford University Press: United Kingdom.
- Danarjati, D.P., Murtiadi, A. & Ekawati, A.R., 2013. Pengantar Psikologi Umum, Graha Ilmu, Yogyakarta.
- Ewolf, Community, 2011, *Indeks Lengkap Syntax*, MediaKom, Yogyakarta.
- Hidayat , Aep Saeful_. 2016. Mengenal TypeScript. Diakses pada : 28 Juni 2017
<https://www.codepolitan.com/mengenal-typescript>
- Horspool, R.N, 1980, Practical fast searching in strings, *Journal Software Practice and Experience*, Vol. 10, Hal 501-506.
- Kadir Abdul. 2002. *Dasar Pemrograman Web Dinamis Menggunakan PHP*. Yogyakarta: Andi.
- Kusnadi, A. & Wicaksono, A.K. (2017), Perbandingan Algoritma Horspool dan Algoritma Zhu-Takaoka dalam Pencarian String Berbasis Desktop, Teknologi Informasi dan Komunikasi, Universitas Multimedia Nusantara, Gading Serpong.
- Maulana, Irfan (2017). Angular 4 - Instalasi dan Tutorial Angular CLI. <https://www.codepolitan.com/angular-4-instalasi-dan-tutorial-angular-cli-5943489a10cc0>. Diakses pada tanggal 3 Februari 2019
- Moreno, Z. 2015. *AngularJS Deployment Essentials*. Packt Publishing.
- Mulyadi, A. 2010. Membangun Aplikasi Android. Yogyakarta Multimedia Center Publishing.

- Mutiawani, V. 2011 Aplikasi Kamus DwiBahasa Aceh-Indonesia Berbasis Java Untuk Telpn Genggam. Infonnatika Unsyiah.
- Pratiwi, H., 2014, Sistem Pendukung Keputusan Penentuan Karyawan Berprestasi Menggunakan Metode Multifactor Evaluation Process, *Jurnal Sistem Informasi*, No. 2, Vol. 5, Hal. 95 – 101
- Pressman, Roger. 2002. Rekayasa Perangkat Lunak Pendekatan Praktisi (Buku I), Yogyakarta: Penerbit Andi & McGraw-Hill Book Co.
- Rayahuningsih, P.A. 2016. Analisis Perbandingan Kompleksitas Algoritma Pengurutan Nilai (Sorting). *Jurnal Evolusi* Vol. 4, No 2 : 2338-8161.
- RI, K. K. (2013). Riset Kesehatan Dasar 2013. 2013: Kementrian Kesehatan RI
- Roeslandy 2012, Pengertian Android, <http://tabloidmakalahindofokus.com.html>. diakses 10 November 2018.
- Sembiring, Jhoni Pranata, 2013, Perancangan Aplikasi Kamus Bahasa Indonesia – Karo Online Berbasis Web dengan Metode Sequential Search. *Jurnal Teknik Informatika 2013*. STMIK Budi Darma, Medan.
- Singh, R. & Verma, H.N., 2011, A fast string matching algorithm. *International Journal of Computer Technology and Applications*, No. 6, Vol. 2, Hal. 1877-1883.
- Sukamto, Rosa A., dan Shalahuddin, M., 2013, *Rekayasa perangkat Lunak Terstruktur dan Berorientasi Objek*, Informatika, Bandung.
- Sukamto, Rosa A., dan Shalahuddin, M., 2011, *Modul Pembelajaran Rekayasa Perangkat Lunak*, Modula, Bandung.
- Suwitri, 2017, Perbandingan Algoritma Raita Dan Algoritma Quick Search Pada Aplikasi Kamus Bahasa Indonesia – Bahasa Prancis, *Skripsi*, Jurusan Ilmu Komputer Universitas Sumatera Utara, Medan.
- Syah, M. 2003. Psikologi Be/ajar. Jakarta: PT Raja Grafindo Persada.
- Tambun, E.D., 2010, Perbandingan Penggunaan Algoritma Boyer-Moore Dan Algoritma Horspool Pada Pencarian String Dalam Bahasa Medis, *Skripsi*, Institut Teknologi Bandung.
- Valba, Ryan R., 2012, Perbandingan Algoritma Horspool Dan Algoritma Raita Pada Aplikasi Kamus Bahasa Indonesia – Jerman Berbasis Web, *Skripsi*, Jurusan Ilmu Komputer Universitas Sumatera Utara, Medan.