

SOFTWARE TESTING  
PENGUJIAN APLIKASI MENGGUNAKAN *BLACK BOX TESTING*

(Studi Kasus : Program Selection Sort (Descending Sort))



Disusun oleh :

Putri Ayu Lestari 065115224()  
Wardah Maulida (065115254)  
Ela Susilawati (065115276)  
Irfany Al Fathwah (065115280)

PROGRAM STUDI ILMU KOMPUTER  
FAKULTAS MIPA DAN ILMU PENGETAHUAN ALAM  
UNIVERSITAS PAKUAN  
2017



## **I. PENDAHULUAN**

### **Latar Belakang**

Program Selection Sort (Descending Sort) merupakan program yang digunakan untuk mengurutkan bilangan dari nilai terbesar ke nilai terkecil. Program ini dibuat menggunakan DEV C++ yang dapat menulis program atau source code, mengkompilasi program (compile), melakukan pengujian program (debugging), mengartikan Object dan Library ke program (linking), dan menjalankan program (running).

Program Selection Sort ini dapat mengurutkan bilangan dari nilai terbesar ke nilai terkecil sesuai dengan nilai masukan (input) yang diberikan.

Setelah program selesai dibuat, maka perlu dilakukan pengujian untuk memastikan semua proses berjalan sesuai dengan yang diinginkan. Pengujian adalah suatu proses pelaksanaan suatu program dengan tujuan menemukan suatu kesalahan. Suatu kasus test yang baik adalah apabila test tersebut mempunyai kemungkinan menemukan sebuah kesalahan yang tidak terungkap. Suatu test yang sukses adalah bila test tersebut membongkar suatu kesalahan yang awalnya tidak ditemukan. Tujuan utama dari pengujian adalah untuk mendesain test yang secara sistematis membongkar jenis kesalahan dengan usaha dan waktu minimum.

### **Maksud dan Tujuan**

#### **1.1. Maksud**

Berdasarkan permasalahan yang dibahas, maka maksud dari pembahasan ini adalah untuk menganalisis program Selection Sort menggunakan metode black box testing

#### **1.2. Tujuan**

- Pengujian black box testing diharapkan mampu menemukan kesalahan dalam kategori fungsi-fungsi yang tidak benar atau hilang
- Menganalisis kesalahan interface

## II. TINJAUAN PUSTAKA

### Pengujian *Software*

Pengujian *software* sangat diperlukan untuk memastikan *software/aplikasi* yang sudah/sedang dibuat dapat berjalan sesuai dengan fungsionalitas yang diharapkan. Pengembang atau penguji *software* harus menyiapkan sesi khusus untuk menguji program yang sudah dibuat agar kesalahan ataupun kekurangan dapat dideteksi sejak awal dan dikoreksi secepatnya. Pengujian atau testing sendiri merupakan elemen kritis dari jaminan kualitas perangkat lunak dan merupakan bagian yang tidak terpisahkan dari siklus hidup pengembangan *software* seperti halnya analisis, desain, dan pengkodean. (Shi, 2010)

Pengujian *software* haruslah dilakukan dalam proses rekayasa perangkat lunak atau *software engineering*. Sejumlah strategi pengujian *software* telah diusulkan dalam literatur. Semuanya menyediakan template untuk pengujian bagi pembuat *software*.

Dalam hal ini, semuanya harus memiliki karakteristik umum berupa (Bhat and Quadri, 2015) :

1. Testing dimulai pada level modul dan bekerja keluar ke arah integrasi pada sistem berbasis komputer
2. Teknik testing yang berbeda sesuai dengan poin-poin yang berbeda pada waktunya
3. Testing diadakan oleh pembuat/pengembang *software* dan untuk proyek yang besar oleh group testing yang independent
4. *Testing* dan *Debugging* adalah aktivitas yang berbeda tetapi *debugging* harus diakomodasikan pada setiap strategi *testing*

Pengujian *software* adalah satu elemen dari sebuah topik yang lebih luas yang sering diartikan sebagai Verifikasi dan Validasi (V&V)

- ☐ Verifikasi: menunjuk kepada kumpulan aktifitas yang memastikan bahwa *software* telah mengimplementasi sebuah fungsi spesifik.
- ☐ Validasi: menunjuk kepada sebuah kumpulan berbeda dari aktivitas yang memastikan bahwa *software* yang telah dibangun dapat ditelusuri terhadap kebutuhan customer.

Definisi V&V meliputi banyak aktifitas SQA (*software quality assurance*), termasuk review teknis formal, kualitas dan audit konfigurasi, monitor performance.

Terdapat beberapa tipe yang berbeda dalam pengujian software yang meliputi studi kelayakan dan simulasi. (Bhat and Quadri, 2015):

1. Metode *software engineering* menyediakan dasar dari mutu yang mana yang akan dipakai.
2. Metode *Analysis, design and Construction* berupa tindakan untuk meningkatkan kualitas dengan menyediakan teknik yang seragam dan hasil yang sesuai dengan keinginan.
3. Metode *Formal Technical Reviews* menolong untuk memastikan kualitas kerja produk merupakan hasil konsekuensi dari setiap langkah *software engineering*.
4. Metode *Measurement* diberlakukan pada setiap elemen dari konfigurasi software
5. Metode *Standards and Procedures* membantu untuk memastikan keseragaman dan formalitas dari SQA untuk menguatkan dasar “filosofi kualitas total”
6. Metoda *Testing* menyediakan cara terakhir dari tingkat kualitas mana yang dapat dicapai dan dengan praktis dapat mengetahui letak error.

Dauids menyarankan satu set prinsip pengujian:

1. Semua test harus dapat dilacak ke kebutuhan pelanggan.
2. Test harus direncanakan dengan baik sebelum pengujian mulai.
  - a. Prinsip Pareto berlaku untuk pengujian
  - b. 80% dari semua kesalahan yang terungkap selama pengujian akan mudah dapat dilacak dari 20% semua modul program.
3. Pengujian seharusnya mulai “dari yang kecil” dan pengujian perkembangan ke arah “yang besar”.
4. Pengujian menyeluruh adalah tidak mungkin. Paling efektif, pengujian harus diselenggarakan oleh suatu pihak ketiga mandiri.

Langkah-langkah pengujian software ada 4 yaitu:

1. *Unit testing*-testing per unit yaitu mencoba alur yang spesifik pada struktur modul kontrol untuk memastikan pelengkapan secara penuh dan pendeteksian error secara maksimum

2. *Integration testing* – testing per penggabungan unit yaitu pengalamatan dari isu-isu yang diasosiasikan dengan masalah ganda pada verifikasi dan konstruksi program
3. *High-order test* yaitu terjadi ketika software telah selesai diintegrasikan atau dibangun menjadi satu –tidak terpisah-pisah
4. *Validation test* yaitu menyediakan jaminan akhir bahwa software memenuhi semua kebutuhan fungsional, kepribadian dan performa.

Ada beberapa jenis pengujian perangkat lunak, antara lain (Khan, 2011):

1. Pengujian *white box* adalah pengujian yang didasarkan pada pengecekan terhadap detail perancangan, menggunakan struktur kontrol dari desain program secara prosedural untuk membagi pengujian ke dalam beberapa kasus pengujian. Secara sekilas dapat diambil kesimpulan *white box testing* merupakan petunjuk untuk mendapatkan program yang benar secara 100%,
2. *Black-Box Testing* merupakan pengujian yang berfokus pada spesifikasi fungsional dari perangkat lunak, tester dapat mendefinisikan kumpulan kondisi input dan melakukan pengetesan pada spesifikasi fungsional program.

### ***White Box Testing***

*White Box Testing* adalah salah satu cara untuk menguji suatu aplikasi atau *software* dengan cara melihat modul untuk dapat meneliti dan menganalisa kode dari program yang di buat ada yang salah atau tidak. Kalau modul yang telah dan sudah di hasilkan berupa output yang tidak sesuai dengan yang di harapkan maka akan dikompilasi ulang dan di cek kembali kode-kode tersebut hingga sesuai dengan yang diharapkan (Nidhra and Dondetti, 2012).

Kasus yang sering menggunakan *white box testing* akan di uji dengan beberapa tahapan yaitu:

1. Pengujian seluruh keputusan yang menggunakan logikal.
2. Pengujian keseluruhan loop yang ada sesuai batasan-batasannya.
3. Pengujian pada struktur data yang sifatnya internal dan yang terjamin validitasnya.

Kelebihan *White Box Testing* antara lain (Nidhra and Dondetti, 2012) :

1. Kesalahan Logika

Menggunakan syntax 'if' dan syntax pengulangan. Langkah selanjutnya metode *white box testing* ini akan mencari dan mendeteksi segala kondisi yang di percaya tidak sesuai dan mencari kapan suatu proses pengulangan di akhiri.

## 2. Ketidaksesuaian Asumsi

Menampilkan dan memonitor beberapa asumsi yang diyakini tidak sesuai dengan yang diharapkan atau yang akan diwujudkan, untuk selanjutnya akan dianalisa kembali dan kemudian diperbaiki.

## 3. Kesalahan Pengetikan

Mendeteksi dan menaribahasa-bahasa pemograman yang di anggap bersifat *case sensitif*.

Kelemahan *White Box Testing* adalah pada perangkat lunak yang jenisnya besar, metode *white box testing* ini dianggap boros karena melibatkan banyak sumberdaya untuk melakukannya. (Nidhra and Dondetti, 2012)

## ***Black Box Testing***

*Black Box Testing* berfokus pada spesifikasi fungsional dari perangkat lunak. *Tester* dapat mendefinisikan kumpulan kondisi input dan melakukan pengetesan pada spesifikasi fungsional program.

*Black Box Testing* bukanlah solusi alternatif dari *White Box Testing* tapi lebih merupakan pelengkap untuk menguji hal-hal yang tidak dicakup oleh *White Box Testing*.

Black Box Testing cenderung untuk menemukan hal-hal berikut:

1. Fungsi yang tidak benar atau tidak ada.
2. Kesalahan antarmuka (*interface errors*).
3. Kesalahan pada struktur data dan akses basis data.
4. Kesalahan performansi (*performance errors*).
5. Kesalahan inisialisasi dan terminasi.

Pengujian didesain untuk menjawab pertanyaan-pertanyaan berikut:

1. Bagaimana fungsi-fungsi diuji agar dapat dinyatakan valid?
2. Input seperti apa yang dapat menjadi bahan kasus uji yang baik?
3. Apakah sistem sensitif pada input-input tertentu?

4. Bagaimana sekumpulan data dapat diisolasi?
5. Berapa banyak rata-rata data dan jumlah data yang dapat ditangani sistem?
6. Efek apa yang dapat membuat kombinasi data ditangani spesifik pada operasi sistem?

Saat ini terdapat banyak metoda atau teknik untuk melaksanakan Black Box Testing, antara lain:

1. *Equivalence Partitioning*
2. *Boundary Value Analysis/Limit Testing*
3. *Comparison Testing*
4. *Sample Testing*
5. *Robustness Testing*
6. *Behavior Testing*
7. *Requirement Testing*
8. *Performance Testing*
9. Uji Ketahanan (*Endurance Testing*)
10. Uji Sebab-Akibat (*Cause-Effect Relationship Testing*)

### ***Boundary Value Analysis / Limit Testing***

*Boundary value analysis* adalah salah satu teknik *black box testing* yang melakukan pengujian pada batas atas dan batas bawah nilai yang diisikan pada aplikasi. Beberapa prinsip yang mendasari pada *boundary value analysis* (BVA) yaitu :

1. Banyak kesalahan terjadi pada kesalahan masukan.
2. BVA mengijinkan untuk menyeleksi kasus uji yang menguji batasan nilai input.
3. BVA merupakan komplemen dari *equivalence partitioning*. Lebih pada memilih elemen-elemen di dalam kelas ekivalen pada bagian sisi batas dari kelas.
4. Contoh:
  - a. Untuk rentang yang dibatasi a dan b maka uji (a-1), a, (a+1), (b-1), b, (b+1).
  - b. Jika kondisi input mensyaratkan sejumlah n nilai maka uji dengan sejumlah (n-1), n dan (n+1) nilai.
  - c. Aplikasikan dua aturan sebelumnya pada kondisi output (buat table pengujian hasil outputnya untuk nilai maksimal dan minimal).



d. Jika struktur data internal dari program

memiliki cakupan (misal: ukuran buffer, batas array) gunakan data input yang menguji batas cakupan.

Secara umum, aplikasi BVA dapat dikerjakan secara generic. Bentuk dasar implementasi BVA adalah untuk menjaga agar satu variable berada pada nilai nominal (normal atau rata-rata) dan mengijinkan variable lain diisikan dengan nilai ekstrimnya.

Sebagai contoh, misalnya akan dientrikan data tanggal. Data tanggal memiliki tiga variable yaitu tanggal, bulan dan tahun. Maka untuk ketiga variable tersebut, dapat diambil kondisi berikut :

$$1 \leq \text{tanggal} \leq 31$$

$$1 \leq \text{bulan} \leq 12$$

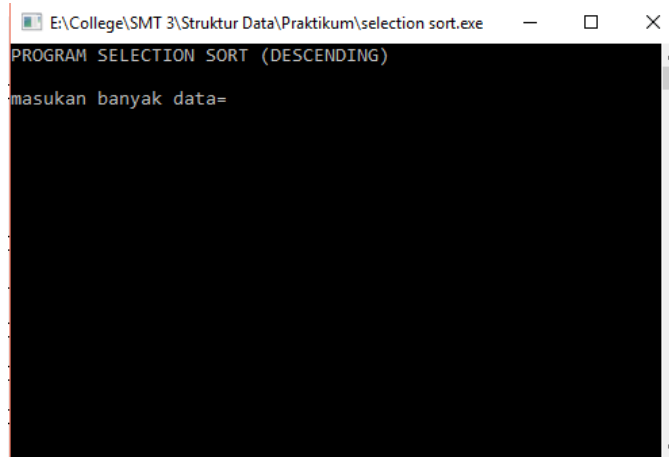
$$1812 \leq \text{tahun} \leq 2016$$

Maka untuk setiap entri data di luar angka di atas akan menampilkan pesan “Tanggal yang anda isikan salah”.

### III. PEMBAHASAN

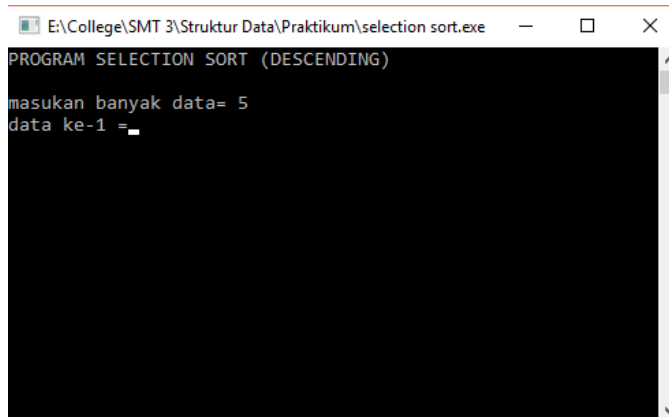
#### Pengujian Black Box Testing

Berdasarkan penjelasan sebelumnya, maka diterapkan teknik Blackbox Testing untuk menguji program yaitu “Program Selection Sort (Descending Sort)”. Program ini sendiri memiliki fungsionalitas yang akan dibahas hasil pengujiannya pada artikel ini. Fungsi ini terdiri atas satu panel entri data seperti pada gambar 1. Pada program ini terdapat field entri data yaitu berapa maksimum data yang akan dimasukan.

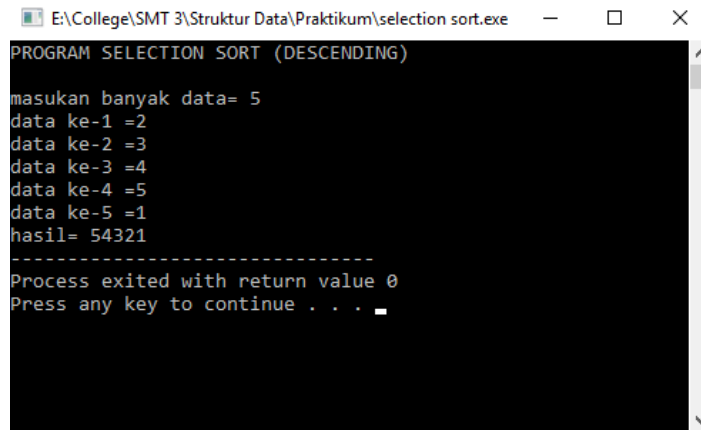


**Gambar 1. Maksimum masukan data**

Berdasarkan program di atas, kemudian dilakukan pengujian dengan menyiapkan beberapa data uji. Dari bentuk program di atas, contoh pengujian akan dilakukan dengan mengisi banyaknya masukan data. Banyak data yang dimasukan sebanyak 5 angka, sehingga terdapat 5 data yang akan diuji. Untuk data-data tersebut, akan dibuat scenario pengujian dan hasil ujinya seperti pada gambar 3 hingga gambar 4.



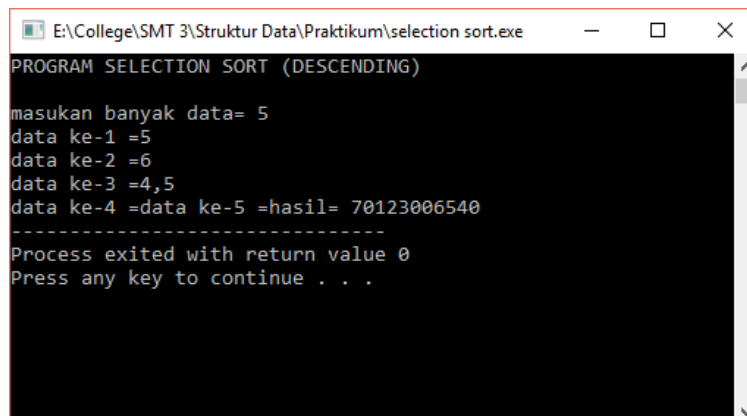
**Gambar 2. Maksimum 5 masukan data**



```
PROGRAM SELECTION SORT (DESCENDING)
masukan banyak data= 5
data ke-1 =2
data ke-2 =3
data ke-3 =4
data ke-4 =5
data ke-5 =1
hasil= 54321
-----
Process exited with return value 0
Press any key to continue . . .
```

**Gambar 3. Hasil uji pertama**

Aturan entri data pertama : harus terdiri dari 5 angka yang bukan bilangan pecahan (integer). Data pertama merupakan angka 2, data kedua merupakan angka 3, data ketiga merupakan angka 4, data keempat merupakan angka 5, dan data kelima



```
PROGRAM SELECTION SORT (DESCENDING)
masukan banyak data= 5
data ke-1 =5
data ke-2 =6
data ke-3 =4,5
data ke-4 =data ke-5 =hasil= 70123006540
-----
Process exited with return value 0
Press any key to continue . . .
```

merupakan angka 1, sehingga hasil yang diperoleh adalah 54321 karena program ini mengurutkan angka dari nilai tertinggi hingga nilai terendah.

**Gambar 4. Hasil uji kedua**

Aturan entri data kedua : dari 5 data yang dimasukan terdapat 1 angka yang merupakan bilangan pecahan (float). Data pertama merupakan angka 5; data kedua merupakan angka 6; dan data ketiga merupakan angka 4,5 sehingga hasil yang diperoleh adalah 70123006540 (error / tidak terurut). Karena data ketiga yang dimasukan berupa bilangan pecahan jadi data keempat serta kelima tidak bisa dimasukan dan program langsung memunculkan hasilnya.

Berdasarkan hasil uji pada program di atas, dapat disiapkan beberapa kasus data uji. Pada contoh di atas, terdapat dua aturan entri data yang akan diuji. Untuk aturan uji pertama, hasil yang ditampilkan merupakan hasil yang valid. Tetapi untuk aturan uji yang kedua, hasil yang ditampilkan tidak valid atau error. Hal ini dapat

memberikan gambaran tipe data uji yang harus disiapkan untuk melakukan blackbox testing. Hasil pengujian memperlihatkan bahwa program ini masih memiliki beberapa kekurangan yaitu belum jelasnya tipe data apa saja yang harus dimasukan sehingga masih perlu penyempurnaan dengan menambah tipe-tipe data yang berkaitan dengan macam-macam bilangan yang ada.

#### **IV. KESIMPULAN**

Berdasarkan hasil analisis maka dapat ditarik kesimpulan bahwa:

1. Metode Blackbox Testing merupakan salah satu metode yang mudah digunakan karena hanya memerlukan batas bawah dan batas atas dari data yang di harapkan,
2. Estimasi banyaknya data uji dapat dihitung melalui banyaknya field data entri yang akan diuji, aturan entri yang harus dipenuhi serta kasus batas atas dan batas bawah yang memenuhi.
3. Setelah melakukan pengujian diketahui bahwa fungsionalitas masih bisa berjalan namun masih dapat menerima masukan data yang tidak diharapkan sehingga dapat menyebabkan data yang disimpan kurang valid.

## **DAFTAR PUSTAKA**

- Shi, Mingtao, 2010, Software Functional Testing from the Perspective of Business Practice Computer and Information Science, [www.ccsenet.org/cis](http://www.ccsenet.org/cis)
- Bhat, A, and Quadri, S.M.K, 2015, Equivalence Class Partitioning and Boundary Value Analysis = A review, 2nd International Conference on  
Computing for Sustainable Global Development (INDIACom)
- Khan, Mohd Ehmer, 2011, Different Approach to Blackbox Testing Technique for Finding Error, International Journal of Software Engineering  
& Applications (IJSEA), Vol.2, No.4, October 2011
- Nidhra, Srinivas, and Dondeti, Jagruthi, 2012, Blackbox and Whitebox Testing Techniques - A Literature Review, International Journal of Embedded Systems and Applications (IJESA) Vol.2, No.2, June 2012