PERTEMUAN 12

Teknik Searching
dan
Pengantar Analisis Algoritma

TEKNIK SEARCHING

1. Pengertian Teknik Searching (Pencarian)

Teknik dalam memilih dan menyeleksi sebuah elemen dari beberapa elemen yang ada.

2. Teknik Searching (Pencarian)

a. Teknik Linear/Sequential Search

b. Teknik StraitMAXMIN

TEKNIK LINEAR/SEQUENTIAL SEARCH

a. **Linear/Sequential Search (Untuk data yang belum terurut/yang sudah terurut)**
   Pencarian yang dimulai dari record-1 diteruskan ke record selanjutnya yaitu record-2, ke-3,..., sampai diperoleh isi record sama dengan informasi yang dicari (Nilai X).

Kecepatan Waktu Pencarian tergantung pada: **Jumlah elemen data dan posisi data yang dicari**

TEKNIK LINEAR/SEQUENTIAL SEARCH (Lanjutan)

Algoritma :

1. Tentukan I = 1
2. Ketika Nilai (I) <> X Maka Tambahkan I = I +1
3. Ulangi langkah No. 2 sampai Nilai(I) = X atau I > N
4. Jika I = N+1 Maka Cetak “Pencarian Gagal” selain itu Cetak “ Pencarian Sukses “

TEKNIK LINEAR/SEQUENTIAL SEARCH (Lanjutan)

Contoh 1:

Data A = { 10, 4, 9, 1, 15, 7 }

Dicari **15**

Langkah pencariannnya:

Langkah 1: A[1] = 10

Langkah 2: 10 <> 15, maka A[2] = 4

Langkah 3: ulangi langkah 2

Langkah 2: 4 <> 15, maka A[3] = 9

Langkah 2: 9 <> 15, maka A[4] = 1

Langkah 2: 1 <> 15, maka A[5] = 15

Langkah 2: 15 = 15

Langkah 4: “Pencarian Sukses”

TEKNIK LINEAR/SEQUENTIAL SEARCH (Lanjutan)

Contoh 2

Apabila ditemukan kondisi:

Nilai (I) = N + 1, maka pencarian tidak ditemukan atau **gagal.**

Dikarenakan jumlah elemen adalah N, N + 1 artinya data yang dicari bukan merupakan elemen data dari N.

TEKNIK LINEAR/SEQUENTIAL SEARCH (Lanjutan)

Kodingan Program Teknik Sequential/Linier Search

```
def seqSearch(data, key):
    i=0
    pos = i + 1
    while(i<len(data)):
        if data[i] == key:
            break
        i+=1
        pos=i+1
    if pos <= len(data):
        print('data', key, 'ditemukan di posisi’, pos)
    else:
        print('data tidak ditemukan')
    return pos
data=[10,  4,  9,  1,  15,  7]
seqSearch(data,15)
```

Hasil Program:

```
data 15 ditemukan di posisi 5
```

Teknik Pencarian MAXMIN

b. Teknik STRAITMAXMIN

- Menentukan/mencari elemen max&min. Pada Himpunan yang berbentuk array linear.

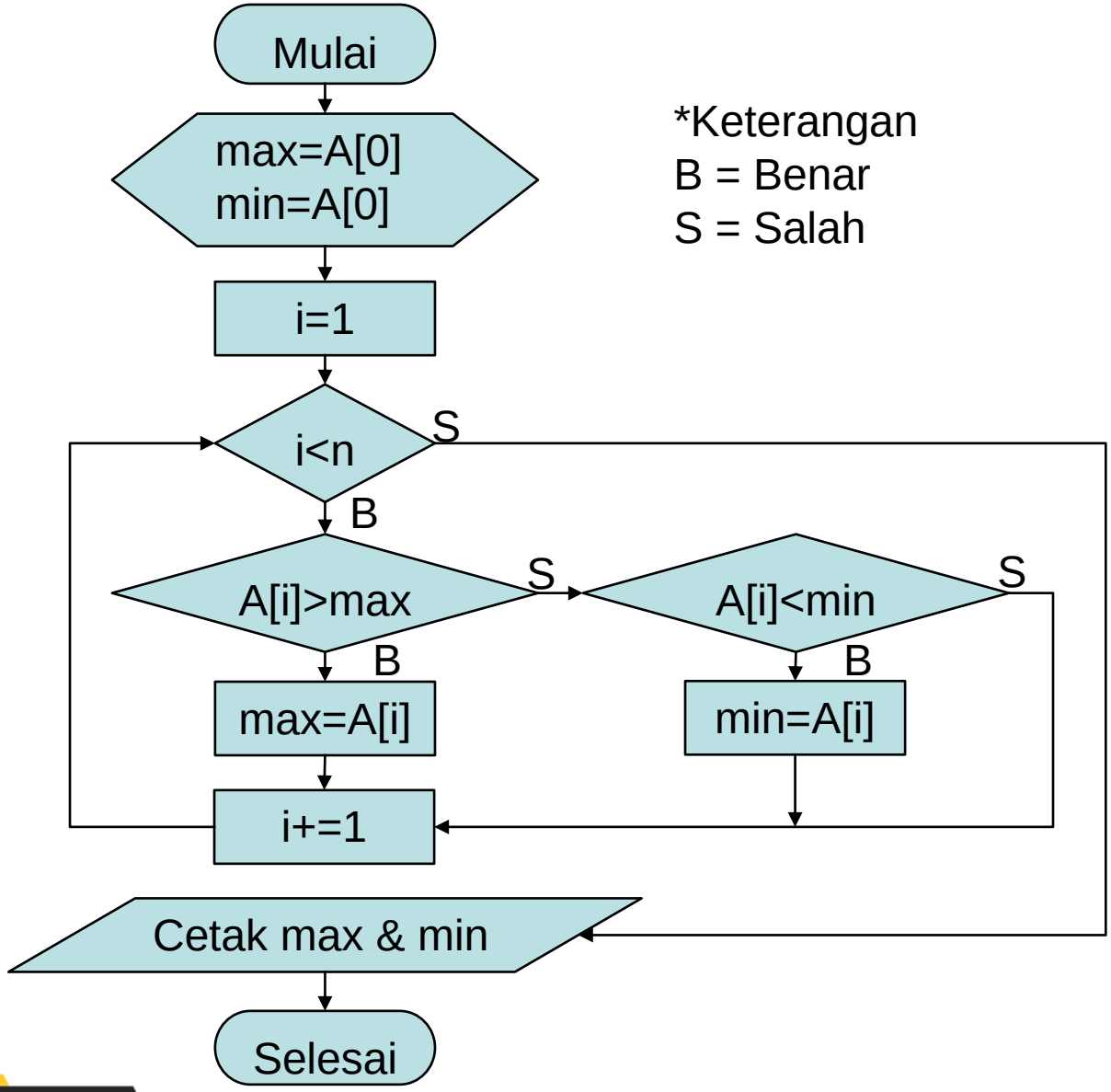
- Analisis waktu tempuh/*time complexity* yang digunakan untuk menyelesaikan pencarian hingga mendapatkan tingkat kompleksitas yang terbagi atas ***best case, worst case*** *dan* ***average case***

Teknik STRAITMAXMIN *(Lanjutan)*

Algoritma untuk mencari elemen MaxMin dalam pemrograman Python :

```
def STRAITMAXMIN(A,n):
    max = A[0]
    min = A[0]
    for i in range(1,n):
        if A[i] > max:
            max = A[i]
        else
            if A[i] < min:
            min = A[i]
    print("Max =", max,", Min =",min)
```

Teknik STRAITMAXMIN *(Lanjutan)*



*Keterangan
B = Benar
S = Salah

BEST CASE

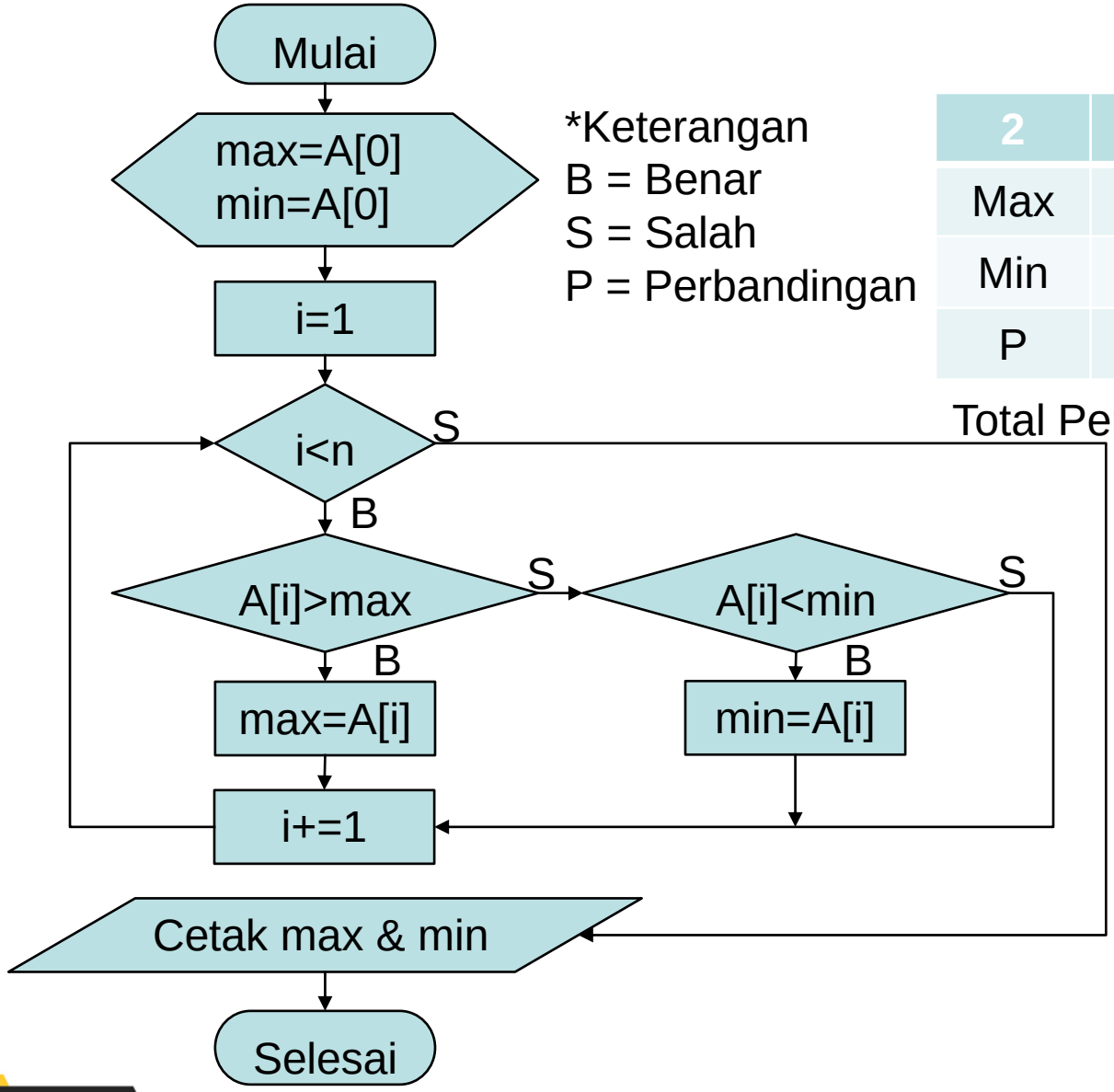
- Pada kasus ini keadaan tercapai jika elemen pada himpunan A disusun secara *increasing* (menaik).
- Dengan perbandingan waktu ***n - 1*** kali satuan operasi.
- Kondisi pencarian yang tercepat/terbaik

Contoh :

Terdapat himpunan A yang berisi 4 buah bilangan telah disusun secara *increasing* dengan A[0]=2, A[1]=4, A[2]=5, A[3]=10.

Tentukan/cari Bilangan Max&Min serta jumlah operasi perbandingan yang dilakukan.

BEST CASE (Lanjutan)



*Keterangan
B = Benar
S = Salah
P = Perbandingan

2	4	5	10
Max	4	5	10
Min	2	2	2
P	1	1	1

Total Perbandingan = 3

BEST CASE (Lanjutan)

Penyelesaian Best Case:

- Untuk masalah tersebut dapat digunakan procedure STRAITMAXMIN yang menghasilkan bilangan Min=2 & bilangan Max=10,

- Operasi perbandingan data mencari bilangan MaxMin dari himpunan tersebut (**n-1**)=3 kali operasi perbandingan.

WORST CASE

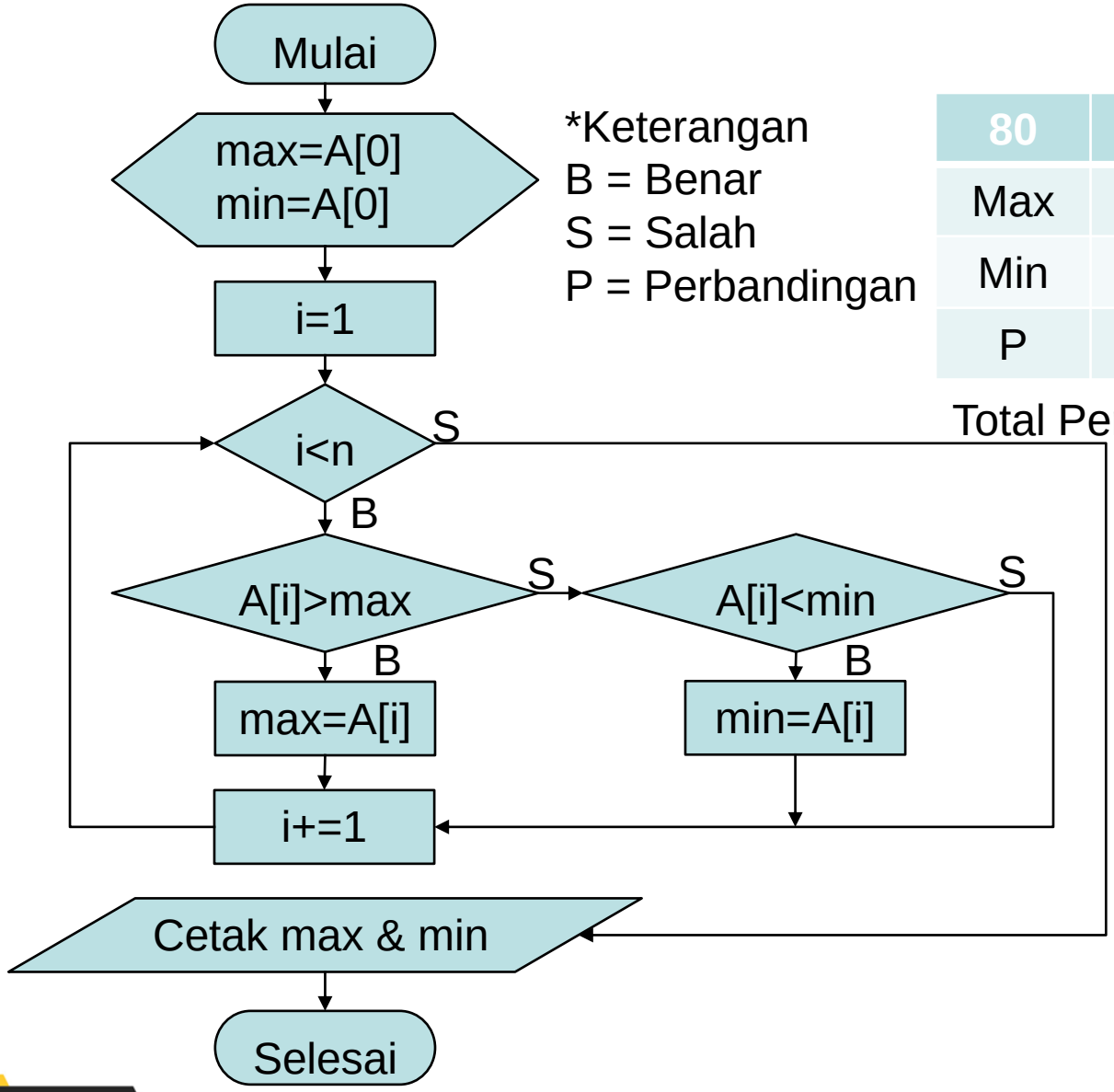
- Pada kasus ini terjadi jika elemen dalam himpunan nilai yang terbesar berada di paling depan atau disusun secara *decreasing* (menurun).
- Dengan Oprasi perbandingan sebanyak ***2(n-1)*** kali satuan operasi.
- Kondisi Pencarian Terlama/Terburuk.

Contoh :

Mencari elemen MaxMin & jumlah oprasi perbandingan yang dilakukan terhadap himpunan A yang disusun *decreasing*.

A[0]=80, A[1]=21, A[2]=6, A[3]=-10

WORST CASE (Lanjutan)



*Keterangan
B = Benar
S = Salah
P = Perbandingan

80	21	6	-10
Max	80	80	80
Min	21	6	-10
P	2	2	2

Total Perbandingan = 6

WORST CASE (Lanjutan)

Penyelesaian Worst Case

- Untuk masalah tersebut dengan proses STRAITMAXMIN adalah elemen max=80 & elemen min=-10,

- Operasi perbandingan untuk elemen Maxmin tersebut adalah 2(4-1) = 6 kali satuan operasi.

AVERAGE CASE

- Digunakan untuk memprediksi jumlah langkah/operasi jika pencarian elemen MaxMin dilakukan pada elemen dalam himpunan yang tersusun secara acak (tidak decreasing/tidak increasing).
- Jumlah prediksi operasi Perbandingan yang dilakukan adalah rata-rata waktu tempuh *best case* & *worst case*, yaitu:

$$\frac{1}{2}[(n-1)+2(n-1)]$$

$$=\frac{1}{2}(n-1+2n-2)$$

$$=\frac{1}{2}(3n-3)=\frac{3}{2}n-\frac{3}{2} \quad \text{Kali}$$

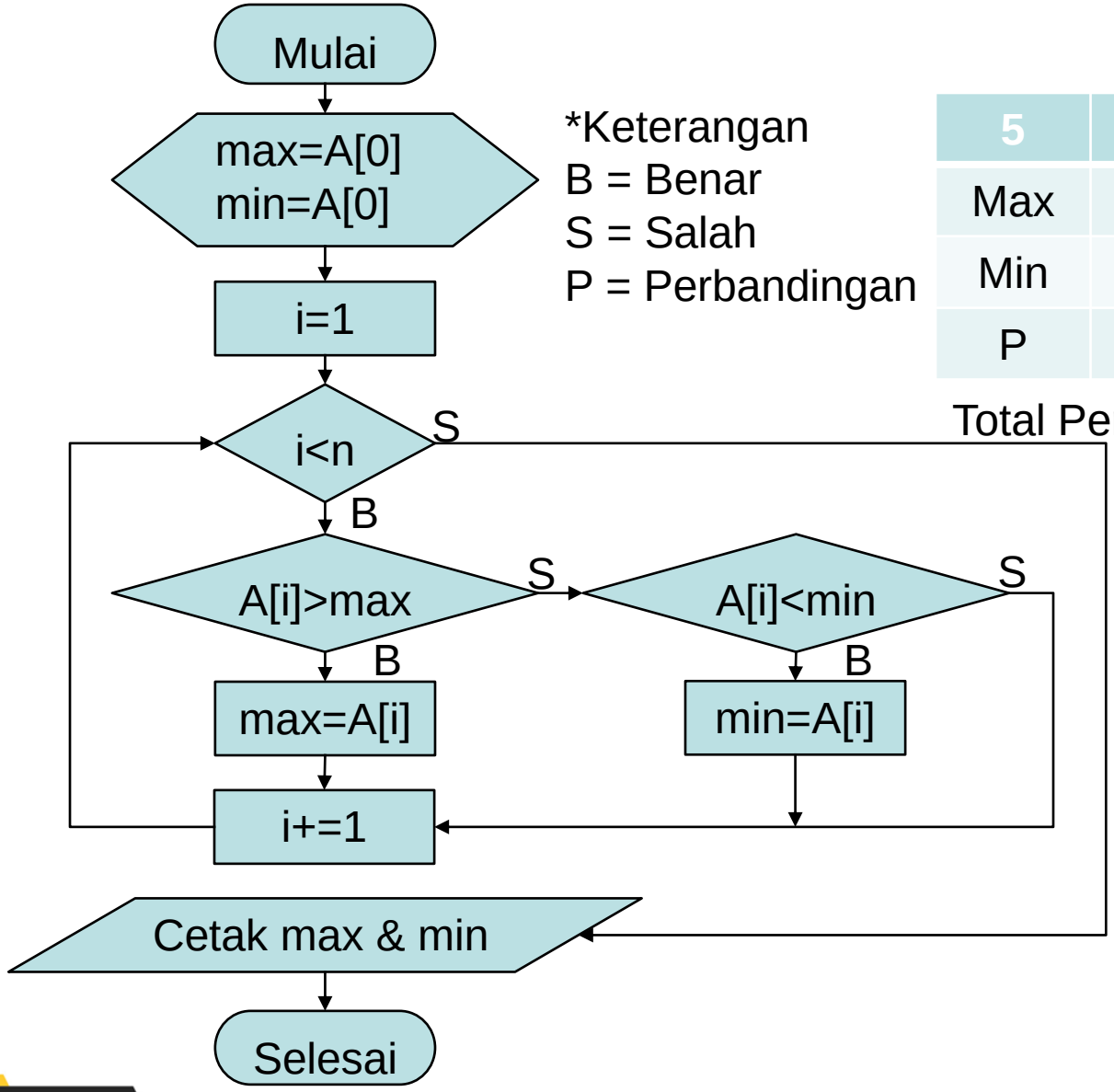
- Kondisi Pencarian acak

Contoh:

Pada himpuan A yg berisi { 5,-4, 9, 7 } dilakukan pencarian elemen max & min dengan menggunakan proses STRAITMAXMIN.

Berapa elemen maxmin yang didapatkan & jumlah oprasi perbandingan yang dilakukan.

AVERAGE CASE (Lanjutan)



*Keterangan
B = Benar
S = Salah
P = Perbandingan

5	-4	9	7
Max	5	9	9
Min	-4	-4	-4
P	2	1	2

Total Perbandingan = 5

AVERAGE CASE (Lanjutan)

Penyelesaian kasus acak

Contoh: A = { 5, -4, 9, 7 }

Operasi Perbandingan : 5

***Penyelesaian kasus acak tidak bisa menggunakan rumus dikarenakan hasil perbandingan akan berada diantara Best Case dan Worst Case**

Best Case < Average Case < Worst Case

Dimana menggunakan rumus:

3/2 n - 3/2 = 3/2*4 - 3/2 = 4,5

*note

Penggunaan average case digunakan untuk membandingkan algoritma dimana jika terjadi perbedaan nilai yang lebih baik antara Best Case dan Worst Case seperti merge sort dan quick sort. Dimana best case lebih baik quick sort namun worst case lebih baik merge sort sehingga untuk menentukan algoritma terbaik dilakukan perbandingan average case (Tidak dibahas karena merupakan analisis algoritma yang lebih mendalam)

Kesimpulan Analisis Algoritma

- Setiap Algoritma bisa dianalis langkah operasi yang dilakukan untuk menentukan kompleksitasnya
- Untuk menyatakan suatu algoritma lebih baik bisa dilakukan dengan membandingkan kompleksitas algoritma tersebut secara menyeluruh
- Jika terjadi perbedaan nilai yang lebih baik antara Best Case dan Worst Case, maka untuk menentukan algoritma terbaik bisa dilakukan perbandingan Average Case