POLITEKNIK STATISTIKA STIS

For Better Official Statistics

SEARCHING

Ibnu Santoso S.S.T., M.T., Nori Wilantika, S.S.T., M.T.I.

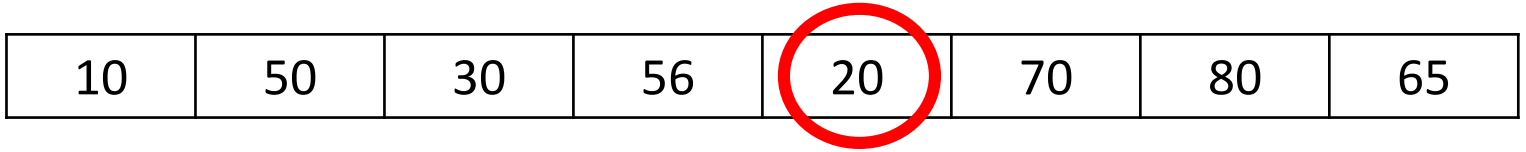
MATERI

- Konsep *searching*
- *Sequential /Linear Search*
  - *Sequential search* pada array tidak berurut
  - *Sequential search* pada array berurut
  - *Sequential search* menggunakan sentinel
- *Binary Search*
- *Sequential Search vs Binary Search*
- Latihan

KONSEP *SEARCHING*

- Pencarian (*searching*) adalah proses yang fundamental dalam pemrograman
- Proses pencarian adalah **menemukan data tertentu di dalam sekumpulan data yang bertipe sama**
- Pada kuliah ini akan dikhususkan pada data yang berstruktur array

Cari ’20’



TINJAUAN SINGKAT ARRAY

- Array merupakan tipe data terstruktur.
- Setiap elemen array dapat dirujuk melalui indeksnya
- Karena elemen disimpan secara berurutan, indeks array harus berupa tipe yang juga memiliki keterurutan (ada *successor* dan *predecessor*) misalnya integer, character, atau enumeration
  - Jika indeks array adalah integer maka keterurutan indeks sesuai dengan urutan integer
  - Jika indeks array adalah character maka keterurutan indeks sesuai dengan urutan character
  - Jika indeks array berupa enumeration, maka keterurutan indeks sesuai dengan urutan elemen dalam enumeration

TINJAUAN SINGKAT ARRAY

```
D: array [1..11] of integer
```

21	36	8	7	10	36	68	32	12	10	36
1	2	3	4	5	6	7	8	9	10	11

```
Kar: array [1..8] of character
```

k	m	t	a	f	m	*	#
1	2	3	4	5	6	7	8

```
Const
  N : integer =  5 {jumlah siswa}
Type
  Data = record <Nama: string, Usia: integer>
Var
  DataSiswa : array[1..N] of Data
```

1	Ali	18
2	Tono	24
3	Amir	30
4	Tuti	21
5	Yani	22

PERSOALAN PENCARIAN 1

- Misal terdapat suatu **Array A** yang sudah terdefinisi elemen-elemennya. **X** adalah suatu elemen yang bertipe sama dengan elemen **Array A**. Tentukan apakah **X** terdapat di dalam **Array A**. Jika ditemukan, tulis pesan "`X ditemukan`", sebaliknya jika tidak ditemukan,tulis pesan "`X tidak ditemukan`".

- Contoh:

21	36	8	7	10	36	68	32	12	10	36
1	2	3	4	5	6	7	8	9	10	11

  - Misal `X = 68`, maka output yang dihasilkan adalah "`68 ditemukan`"
  - Bila `X = 100`, maka output yang dihasilkan adalah "`100 tidak ditemukan`".

PERSOALAN PENCARIAN 2

- Misal terdapat suatu Array `A` yang sudah terdefinisi elemen-elemennya. `X` adalah suatu elemen yang bertipe sama dengan elemen Array `A`. Tentukan indeks Array `A` yang elemennya sama dengan `X`. Simpan indeks tersebut dalam variabel `Y`. Jika `X` tidak terdapat di dalam Array `A`, isikan `Y` dengan nilai `0`.

- Contoh:

21	36	8	7	10	36	68	32	12	10	36
1	2	3	4	5	6	7	8	9	10	11

  - Misal `X = 68`, maka `Y = 7`.
  - Bila `X = 100`, maka `Y = 0`.

PERSOALAN PENCARIAN 3

- Misal terdapat suatu Array `A` yang sudah terdefinisi elemen-elemennya. `X` adalah suatu elemen yang bertipe sama dengan elemen Array `A`. Tentukan apakah `X` terdapat di dalam Array `A`. Jika `X` ditemukan, maka variabel boolean misalnya `ketemu` diisi nilai `true`. Jika tidak, `ketemu` diisi nilai `false`.

- Contoh:

21	36	8	7	10	36	68	32	12	10	36
1	2	3	4	5	6	7	8	9	10	11

  - Misal `X = 68`, maka `ketemu = true`
  - Bila `X = 100`, maka `ketemu = false`

BAGAIMANA JIKA ADA DUPLIKASI ELEMEN?

- Apabila X yang dicari jumlahnya lebih dari satu di dalam Array A, maka hanya X yang pertama kali ditemukan yang dirujuk. **Proses pencarian dihentikan setelah X pertama ditemukan atau X yang dicari tidak ada**.

- Contoh:

21	36	8	7	10	36	68	32	12	10	36
1	2	3	4	5	6	7	8	9	10	11

  - Terdapat tiga buah nilai 36
  - Bila `X = 36`, maka:
    - Persoalan 1, hasilnya "`36 ditemukan`"
    - Persoalan 2, hasilnya `Y = 2`
    - Persoalan 3, hasilnya `ketemu = true`

Kalau yang ditanya hanay ada atau tidak, maka diambil yang pertama kali ditemukan. kalau ditanya frekuensinya sudah beda lagi.

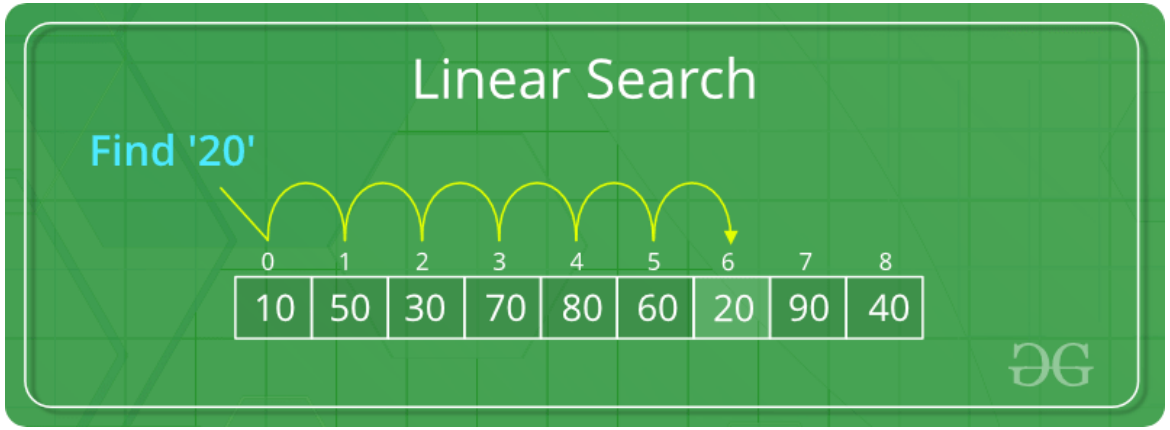
SEQUENTIAL SEARCH

SEQUENTIAL /LINEAR SEARCH

- Disebut juga Pencarian Beruntun
- *Sequential Search* membandingkan setiap elemen array satu per satu secara beruntun, mulai dari elemen pertama, sampai elemen yang dicari ditemukan atau sampai seluruh elemen sudah diperiksa.
- Terdapat 2 macam *sequential search*:
  - *sequential search* pada array tidak terurut
  - *sequential search* pada array terurut

SEQUENTIAL SEARCH PADA ARRAY TIDAK TERURUT

- Pencarian dilakukan dengan memeriksa setiap elemen Array mulai dari elemen pertama sampai elemen yang dicari **ketemu** atau sampai **seluruh elemen telah diperiksa**.

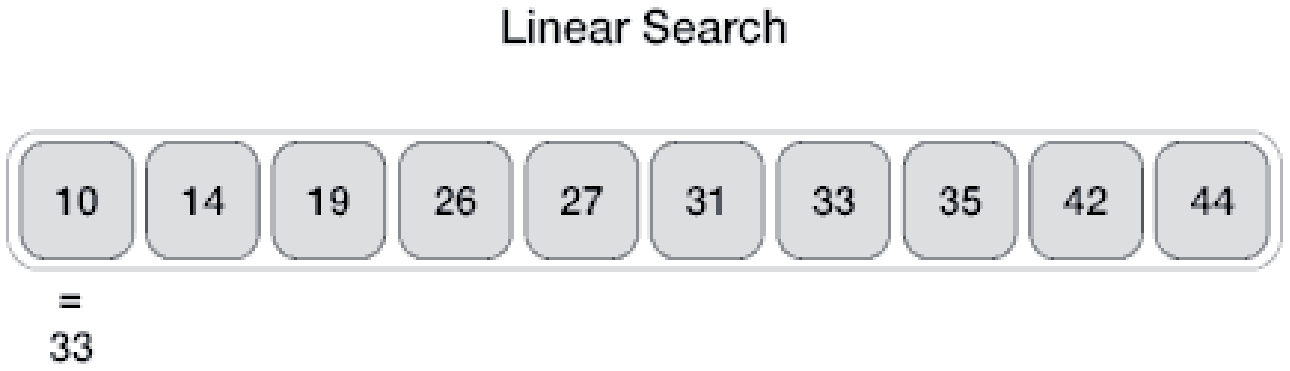


- Misal nilai yang dicari adalah: `X = 20`. maka elemen yang diperiksa: `10,50,30,70,80,60,20 (ditemukan!)`. Indeks Array yang dikembalikan: `Y = 6`
- Misal nilai yang dicari adalah: `X = 10`. maka elemen yang diperiksa: `10(ditemukan!).` Indeks Array yang dikembalikan: `Y = 0`
- Misal nilai yang dicari adalah: `X = 15`. maka elemen yang diperiksa: `10,50,30,70,80,60,20,90,40(tidak ditemukan!).` Indeks Array yang dikembalikan: `Y = -1`

ILUSTRASI SEQUENTIAL SEARCH

Mencari elemen “33” dengan memeriksa elemen satu per satu.

Sehingga ditemukan bahwa elemen “33” berada pada indeks ke-7.



IMPLEMENTASI PROGRAM

```
Type Larik = array [1..100] of integer;
Procedure SequentialSearch1(A:Larik; N:integer; X:integer;  var Y:integer);
Var
    i:integer;
Begin
    i:=1;
    while (i<N) and (A[i] <> X) do i:= i + 1;
    if (A[i] <> X) then Y:=0
    else Y:=i;
End;
```

```
A= 13 16 14 21 76 21
N= 6
X= 14
Y=

i      i<6?      A[i]<>14?
1       T            T
2       T            T
3       T            F

A[3]<>14? F -> Y=3
```

```
A= 13 16 14 21 76 21
N= 6
X= 100
Y=

i      i<6?      A[i]<>100?
1       T            T
2       T            T
3       T            T
4       T            T
5       T            T
6       F

A[6]<>100? T -> Y=0
```

IMPLEMENTASI PROGRAM VERSI BOOLEAN

```
Type Larik = array [1..100] of integer;
Procedure SequentialSearch2(A:Larik; N:integer; X:integer;  var Y:integer);
Var
    i:integer;
    ketemu:boolean;
Begin
    i:=1;
    ketemu:=false;
    while (i<=N) and (not ketemu) do
        if A[i] = X then ketemu:=true else i := i + 1;

    if ketemu then Y:=i else Y:=0;
End;
```

```
A= 13 16 14 21 76 21
N= 6
X= 14
Y=

i     i<=6?    not ketemu?
1       T      T, A[1]=14?
2       T      T, A[2]=14?
3       T      T, A[3]=14?, ketemu=T

Y=3
```

```
A= 13 16 14 21 76 21
N= 6
X= 100
Y=

i     i<=6?    not ketemu?
1       T      T, A[1]=100?
2       T      T, A[2]=100?
3       T      T, A[3]=100?
4       T      T, A[4]=100?
5       T      T, A[5]=100?
6       T      T, A[6]=100?
7       F

Y=0
```

Sequential search pada array terurut

- Apabila array sudah terurut, maka proses pencarian dapat dibuat **lebih efisien**.
  - terurut dari nilai terkecil ke nilai terbesar, yakni untuk setiap

    ```
    I = 1..N, Nilai [I-1] < Nilai [I]
    ```

  - terurut dari nilai terbesar ke nilai terkecil, yakni untuk setiap

    ```
    I = 1..N, Nilai [I-1] > Nilai[I]
    ```

- Caranya yaitu dengan menghilangkan langkah pencarian yang tidak perlu yaitu **bila nilai elemen array yang diperiksa sudah melewati nilai X yang dicari** maka pencarian bisa dihentikan

Sequential search pada array terurut (dari kecil ke besar)

```
Type Larik = array [1..100] of integer;
Procedure Sequentialsearch3(A:Larik; N:integer; X:integer;  var Y:integer);
Var
    i:integer;
Begin
    i:=1;
    while (i<N) and (A[i] < X) do i:= i + 1;
    if (A[i] = X) then Y:=i
    else Y:=0;
End;
```

```
A= 13 14 16 21 21 76
N= 6
X= 14
Y=

i     i<6?     A[i]<14?
1      T          T
2      T          F

A[2]=14? T -> Y=2
```

```
A= 13 14 16 21 21 76
N= 6
X= 15
Y=

i     i<6?     A[i]<15?
1      T          T
2      T          T
3      T          F

A[3]=15? F -> Y=0
```

SEQUENTIAL SEARCH MENGGUNAKAN SENTINEL

- Merupakan pengembangan dari *sequential search*
- Sentinel adalah elemen fiktif yang ditambahkan sesudah elemen terakhir dari array
  - Jadi jika elemen terakhir array adalah `A[N]`, maka sentinel dipasang pada elemen `A[N+1]`

- Sentinel **nilainya sama dengan nilai data yang dicari**
  - Sehingga proses pencarian selalu menemukan data yang dicari
  - Periksa kembali letak data tersebut ditemukan, apakah:
    - Di antara elemen array sesungguhnya (dari `A[1]` sampai `A[N]`)
    - Pada elemen fiktif `A[N+1]` yang berarti sesungguhnya **X tidak ada di dalam array A**

Pengecekaan kondisi akan lebih memberatkan komputasi daripada looping yang banyak. jadi penggunaan looping yang banyak bisa lebih efektif daripada pengecekan kondisi.

SEQUENTIAL SEARCH MENGGUNAKAN SENTINEL

13	16	14	21	76	21
1	2	3	4	5	6

- Data yang dicari adalah `X=21`.
- Tambahkan `21` sebagai sentinel di `A[N+1]`.

13	16	14	21	76	21	21
1	2	3	4	5	6	7

Programmer harus hati-hati dengan pendefinisian batas indeks array, tidak boleh menambahkan data melebihi rentang indeks.

- Telusuri array seperti sequential search tanpa sentinel, jika data ditemukan pada **sentinel** maka data yang dicari **tidak ada/tidak ditemukan**. Tetapi, jika data yang dicari ditemukan **bukan pada sentinel** maka data **ditemukan**.
- Indeks array yang dikembalikan = `4`. karena `4 ≠ N+1`, berarti `X = 21` terdapat pada Array semula, maka data **ditemukan**.

SEQUENTIAL SEARCH MENGGUNAKAN SENTINEL

13	16	14	21	76	21
1	2	3	4	5	6

- Data yang dicari adalah `X=15`.
- Tambahkan `15` sebagai sentinel di `A[N+1]`.

13	16	14	21	76	21	15
1	2	3	4	5	6	7

- Telusuri array seperti sequential search tanpa sentinel.
- Indeks array yang dikembalikan = `7`. karena `7 = N+1`, berarti `X = 15` tidak terdapat pada Array semula, maka data **tidak ditemukan**.

IMPLEMENTASI PROGRAM

```
Type Larik = array [1..100] of integer;
Procedure SequentialSearchSentinel(A:Larik; N:integer; X:integer;  var Y:integer);
Var
    i:integer;
Begin
    A[N+1]:=X;
    i:=1;
    while (A[i]<>X) do i:= i + 1;
    if (i < N+1) then Y:=i else Y:=0;
End;
```

```
A= 13 16 14 21 76 21
N= 6
X= 14
Y=

A= 13 16 14 21 76 21 14

i    A[i]<>14?
1        T
2        T
3        F

3<7? T -> Y=3
```

```
A= 13 16 14 21 76 21
N= 6
X= 100
Y=

A= 13 16 14 21 76 21 100

i    A[i]<>100?
1        T
2        T
3        T
4        T
5        T
6        T
7        F

7<7? F -> Y=0
```

DENGAN SENTINEL VS TANPA SENTINEL

Dengan Sentinel

```
Begin
    A[N+1]:=X;
    i:=1;
    while (A[i]<>X) do i:= i + 1;
    if (i < N+1) then Y:=i else Y:=0;
End;
```

A= 13 16 14 21 76 21
N= 6
X= 14
Y=

A= 13 16 14 21 76 21 14

i	A[i]<>14?
1	T
2	T
3	F

3<7? T -> Y=3

Tanpa Sentinel

```
Begin
    i:=1;
    while (i<N) and (A[i] <> X) do i:= i + 1;
    if (A[i] <> X) then Y:=0
    else Y:=i;
End;
```

A= 13 16 14 21 76 21
N= 6
X= 14
Y=

i	i<6?	A[i]<>14?
1	T	T
2	T	T
3	T	F

A[3]<>14? F -> Y=3

DENGAN SENTINEL VS TANPA SENTINEL

Dengan Sentinel

```
Begin
    A[N+1]:=X;
    i:=1;
    while (A[i]<>X) do i:= i + 1;
    if (i < N+1) then Y:=i else Y:=0;
End;
```

A= 13 16 14 21 76 21
N= 6
X= 100
Y=

A= 13 16 14 21 76 21 100

i	A[i]<>100?
1	T
2	T
3	T
4	T
5	T
6	T
7	F

N+1

1

7<7? F -> Y=0

Tanpa Sentinel

```
Begin
    i:=1;
    while (i<N) and (A[i] <> X) do i:= i + 1;
    if (A[i] <> X) then Y:=0
    else Y:=i;
End;
```

A= 13 16 14 21 76 21
N= 6
X= 100
Y=

N

N-1

i	i<6?	A[i]<>100?
1	T	T
2	T	T
3	T	T
4	T	T
5	T	T
6	F	

A[6]<>100? T -> Y=0

1

CATATAN TENTANG *SEQUENTIAL SEARCH*

- Seringkali algoritma *Sequential search* tidak dinyatakan sebagai prosedur, namun direalisasikan sebagai fungsi boolean atau fungsi integer yang mengembalikan indeks array.
- *Sequential search* menggunakan sentinel sangat efektif terutama bila pencarian dilakukan untuk menyisipkan elemen (X) yang belum terdapat pada array. Jika X tidak ditemukan, maka sentinel tersebut sekaligus sudah ditambahkan.
- Secara umum, *Sequential search* lambat. Waktu pencarian sebanding dengan jumlah elemen array. Pada kasus X tidak terdapat dalam Array, kita harus memeriksa seluruh elemen array.
  - Misalkan array berukuran 100.000 elemen.
  - Bila satu pemeriksaan elemen array membutuhkan waktu 0,01 detik, maka untuk 100.000 kali pemeriksaan membutuhkan waktu 1000 detik atau 16,7 menit!
- Algoritma *sequential search* tidak praktis untuk data berukuran besar
- Algoritma yang lebih cepat dari *sequential search* adalah algoritma *binary search*

25

BINARY SEARCH

BINARY SEARCH

- Disebut juga Pencarian Biner.
- Pencarian biner **hanya bisa** diterapkan pada sekumpulan data yang sudah terurut (terurut menaik atau menurun).
- Contoh data yang sudah terurut banyak ditemukan pada kehidupan sehari-hari:
  - Data kontak telepon di HP terurut dari nama A sampai Z
  - Data pegawai diurut berdasarkan nomor induk pegawai dari kecil ke besar
  - Data mahasiswa diurutkan berdasarkan NIM
  - Kata-kata di dalam kamus diurut dari A sampai Z
- Keuntungan data yang terurut adalah memudahkan pencarian.

ANALOGI *BINARY SEARCH*

- Misal untuk mencari arti kata tertentu dalam kamus Bahasa Inggris, kita tidak perlu membuka kamus itu dari halaman awal sampai akhir satu per satu:
  - Pertama Kamus tersebut kita bagi dua di tengah-tengah.
  - Jika yang dicari tidak ada di pertengahan, kita cari lagi di belahan kiri atau kanan dengan membagi dua lagi belahan tersebut.
  - Begitu seterusnya sampai kata yang dicari ketemu.

ILUSTRASI *BINARY SEARCH* 1

- Misal ada array A dengan 8 elemen yang terurut menaik. Data yang dicari adalah X = 16.

7	10	13	16	18	21	76	81
L=1	2	3	4	5	6	7	H=8

- **Langkah 1**: L=1 dan H=8. maka indeks elemen tengah M = (1+8) div 2 = 4

kiri

7	10	13	16	18	21	76	81
L=1	2	3	4	5	6	7	H=8

kanan

- **Langkah 2**: A[4] = 16? Ya! (X ditemukan, pencarian dihentikan)

ILUSTRASI *BINARY SEARCH* 2

- Misal ada array L dengan 8 elemen yang terurut menaik. Data yang dicari adalah X = 18.

7	10	13	16	18	21	76	81
L=1	2	3	4	5	6	7	H=8

- **Langkah 1**: L=1 dan H=8. maka indeks elemen tengah M = (1+8) div 2 = 4

kiri

7	10	13	16	18	21	76	81
L=1	2	3	4	5	6	7	H=8

kanan

- **Langkah 2**: A[4] = 18? Tidak! Putuskan pencarian akan dilakukan di bagian kiri atau kanan dengan pemeriksaan:
  - A[4] < 18 ? Ya! Lakukan pencarian di larik bagian kanan

- L = M+1 = 5 dan H = 8 (tetap)

18	21	76	81
L=5	6	7	H=8

- **Langkah 1**: Indeks elemen tengah M = (5+8) div 2 = 6

kiri

18	21	76	81
L=5	6	7	H=8

kanan

- **Langkah 2**: A[6] = 18? Tidak! Putuskan pencarian akan dilakukan di bagian kiri atau kanan dengan pemeriksaan:
  - A[6] < 18 ? Tidak! Lakukan pencarian di larik bagian kiri.

- L = 5 (tetap) dan H= M-1 = 5

18
L = H = 5

- **Langkah 1**: Indeks elemen tengah M = (5+5) div 2 =5

18
L = H = M= 5

- **Langkah 2**: M[5] = 18? Ya! (X ditemukan, pencarian dihentikan)

ALGORITMA *BINARY SEARCH*

- Kita memerlukan dua buah indeks array yaitu indeks terkecil dan indeks terbesar (Misal L dan H). Umumnya L=1 dan H=N.

- **Langkah 1**: bagi dua elemen array pada elemen tengah. Elemen tengah adalah elemen dengan indeks M = (L+H) DIV 2
  - Elemen tengah array A[M] membagi array membagi dua bagian, yaitu bagian kiri Array A[A..M-1] dan bagian kanan Array A[M+1..H]

- **Langkah 2**: periksa apakah A[M] = X. Jika Ya, maka pencarian dihentikan (data ditemukan).
  - Jika tidak, jika A[M] < X maka pencarian dilakukan pada larik bagian kanan.
  - Jika A[M] > X maka pencarian dilakukan pada larik bagian kiri.

- **Langkah 3**: Ulangi langkah 1-2 sampai X ditemukan atau L > H (ukuran array sudah nol!)

ILUSTRASI *BINARY SEARCH 3*

- Misal ada array L dengan 8 elemen yang terurut menaik. Data yang dicari adalah X = 100.

7	10	13	16	18	21	76	81
L=1	2	3	4	5	6	7	H=8

- **Langkah 1**: L=1 dan H=8. maka indeks elemen tengah M = (1+8) div 2 = 4

kiri

7	10	13	16	18	21	76	81
L=1	2	3	4	5	6	7	H=8

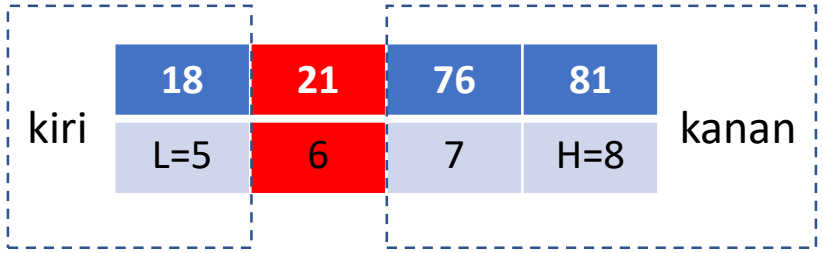
kanan

- **Langkah 2**: A[4] = 100? Tidak! Putuskan pencarian akan dilakukan di bagian kiri atau kanan dengan pemeriksaan:
  - A[4] < 100 ? Ya! Lakukan pencarian di larik bagian kanan

- L = M+1 = 5 dan H = 8 (tetap)

18	21	76	81
L=5	6	7	H=8

- **Langkah 1**: Indeks elemen tengah M = (5+8) div 2 = 6



- **Langkah 2**: L[6] = 100? Tidak! Putuskan pencarian akan dilakukan di bagian kiri atau kanan dengan pemeriksaan:
  - A[6] < 100 ? Ya! Lakukan pencarian di larik bagian kanan.

- L = M+1 = 7 dan H = 8 (tetap)

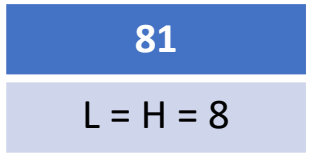
76	81
L=7	H=8

- **Langkah 1**: Indeks elemen tengah M = (7+8) div 2 =7

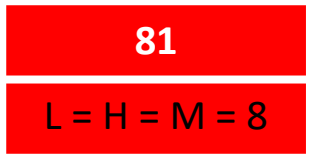
76	81
L = M = 7	H=8

- **Langkah 2**: M[7] = 100? Tidak! Putuskan pencarian akan dilakukan di bagian kiri atau kanan dengan pemeriksaan:
  - A[7] < 100 ? Ya! Lakukan pencarian di larik bagian kanan.

- L = M+1 = 8 dan H = 8 (tetap)



- Langkah 1: Indeks elemen tengah M = (8+8) div 2 = 8



- Langkah 2: A[8] = 100? Tidak! Putuskan pencarian akan dilakukan di bagian kiri atau kanan dengan pemeriksaan:
  - A[8] < 100 ? Ya! Lakukan pencarian di larik bagian kanan dengan L=M+1=9 dan H= 8
  - Karena L > H maka tidak ada lagi bagian array yang tersisa,dengan demikian, X tidak ditemukan di dalam array. Pencarian dihentikan.

BINARY SEARCH PADA ARRAY TERURUT MENURUN

- Misal yang dicari adalah X = 16

81	76	21	18	16	13	10	7
L=1	2	3	4	5	6	7	H=8

81	76	21	18	16	13	10	7
L=1	2	3	M=4	5	6	7	H=8

81	76	21	18	16	13	10	7
1	2	3	4	L = 5	6	7	H=8

81	76	21	18	16	13	10	7
1	2	3	4	L = 5	M=6	7	H=8

81	76	21	18	16	13	10	7
1	2	3	4	L = H = 5	6	7	H

81	76	21	18	16	13	10	7
1	2	3	4	L = H = M = 5	6	7	H

Jika A[M] < X, maka pencarian dilakukan pada array bagian kiri.

Jika A[M] > X maka pencarian dilakukan pada array bagian kanan.

CATATAN TENTANG *BINARY SEARCH*

- Pada setiap kali pencarian, array dibagi dua menjadi dua bagian yang berukuran sama (atau beda selisih 1 elemen)
- Pada setiap pembagian, elemen tengah K diperiksa apakah sama dengan X.
  - Pada *worst case scenario*, yaitu pada kasus X tidak terdapat di dalam array, array dibagi sejumlah $^{2}\log(N)$ kali, sehingga jumlah pemeriksaan yang dilakukan juga sebanyak $^{2}\log(N)$ kali.
  - Pada contoh x=100, pembagian array yang dilakukan adalah $^{2}\log(8)$ =3 kali. Jumlah pemeriksaan elemen array juga 3 kali.
- Untuk data terurut, algoritma *binary search* lebih cepat dibandingkan *sequential search*

SEQUENTIAL SEARCH VS BINARY SEARCH

- Misal untuk kasus nilai X tidak ditemukan dalam array:
  - Untuk array berukuran 256 elemen:
    - *sequential search* melakukan pembandingan elemen array sebanyak 256 kali,
    - *binary search* melakukkan pembandingan elemen array sebanyak $^{2}\log(256) = 8$ kali.
  - Untuk array berukuran 1024 elemen:
    - *sequential search* melakukan pembandingan elemen array sebanyak 1024 kali,
    - *binary search* melakukkan pembandingan elemen array sebanyak $^{2}\log(1024) = 10$ kali.
  - Untuk array berukuran N elemen:
    - *sequential search* melakukan pembandingan elemen array sebanyak N kali,
    - *binary search* melakukan pembandingan elemen array sebanyak $^{2}\log(N)$ kali.
- Karena $^{2}\log(N) < N$ untuk N yang besar, maka algoritma *binary search* lebih cepat daripada algoritma *sequential search*
  - Sehingga Algoritma *binary search* lebih disukai untuk mencari data pada array terurut
- Namun untuk data yang tidak terurut, hanya dapat menggunakan algoritma *sequential search*

POLITEKNIK STATISTIKA STIS

For Better Official Statistics

TERIMA KASIH