

## Tugas Searching

# PRAKTIKUM ALGORITMA DAN STRUKTUR DATA



Nama : Muhammad Daffa  
NRP : 3124510006  
Prodi : D3 PJJ Teknik Informatika

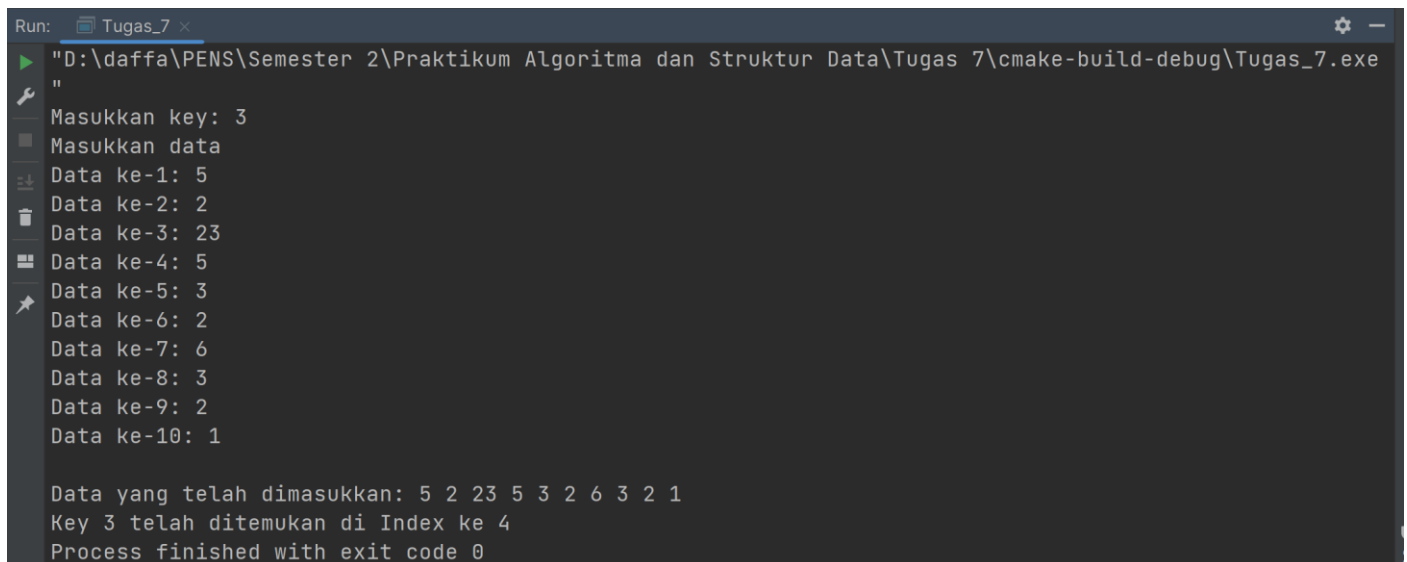
PRAKTIKUM ALGORITMA DAN STRUKTUR DATA  
POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

2024

## Sequential Search:

```
1  #include "stdio.h"
2
3  int sequentialSearch(int array[], int size, int key) {
4      int index = 0;
5      int indexAkhir = size - 1;
6
7      while (index ≤ indexAkhir) {
8          if (array[index] == key) {
9              return index;
10         } else {
11             index += 1;
12         }
13     }
14
15     return -1;
16 }
17
18 int main () {
19
20     int size = 10;
21     int array[size];
22     int key;
23
24     /*===== Input =====*/
25     printf( format: "Masukkan key: ");
26     scanf( format: "%d", &key);
27
28     printf( format: "Masukkan data\n");
29     for (int i = 0; i < size; ++i) {
30         printf( format: "Data ke-%d: ", i + 1);
31         scanf( format: "%d", &array[i]);
32     }
33     /*===== Input =====*/
34
35
36     /*===== Output =====*/
37     printf( format: "\nData yang telah dimasukkan: ");
38     for (int i = 0; i < size; i++) {
39         printf( format: "%d ", array[i]);
40     }
41
42     int index = sequentialSearch(array, size, key);
43     if (index ≠ -1) {
44         printf( format: "\nKey %d telah ditemukan di Index ke %d", key, sequentialSearch(array, size, key);
45     } else {
46         printf( format: "\nKey %d tidak di temukan", key);
47     }
48     /*===== Output =====*/
49
50     return 0;
51 }
```

## Output:



```
Run: Tugas_7
"D:\daffa\PENS\Semester 2\Praktikum Algoritma dan Struktur Data\Tugas 7\cmake-build-debug\Tugas_7.exe"
Masukkan key: 3
Masukkan data
Data ke-1: 5
Data ke-2: 2
Data ke-3: 23
Data ke-4: 5
Data ke-5: 3
Data ke-6: 2
Data ke-7: 6
Data ke-8: 3
Data ke-9: 2
Data ke-10: 1

Data yang telah dimasukkan: 5 2 23 5 3 2 6 3 2 1
Key 3 telah ditemukan di Index ke 4
Process finished with exit code 0
```

## Penjelasan:

Program di atas mengimplementasikan Sequential Search (pencarian berurutan) pada sebuah array berukuran tetap (size = 10). Setelah membaca nilai kunci (key) dan mengisi elemen-elemen array melalui scanf, fungsi sequentialSearch dipanggil. Fungsi ini memulai pencarian dari indeks 0 hingga indeks terakhir (size-1), membandingkan setiap elemen dengan key. Jika ditemukan kecocokan, fungsi langsung mengembalikan posisi indeks tersebut; jika tidak, ia terus menelusuri hingga akhir dan mengembalikan -1 saat key tidak ada.

Pembahasan Kompleksitas dan Perilaku Sequential Search memiliki kompleksitas waktu rata-rata dan terburuk sebesar  $O(n)$ , di mana  $n$  adalah jumlah elemen array. Pada kasus terbaik (jika key ada di posisi pertama), pencarian selesai cepat dalam satu perulangan, sedangkan pada kasus terburuk (jika key tidak ada atau di akhir array), semua elemen diperiksa. Program ini juga mencetak ulang hasil pencarian menggunakan pemanggilan fungsi kedua—sebaiknya cukup sekali dengan menyimpan hasil di variabel index untuk efisiensi.

## Kesimpulan:

1. **Kesederhanaan:** Sequential Search mudah dipahami dan diimplementasikan, cocok untuk array berukuran kecil atau data yang hampir selalu berisi elemen yang dicari di awal.

2. **Keterbatasan:** Kurang efisien untuk array besar karena harus memeriksa setiap elemen hingga akhir—pada data besar, metode seperti Binary Search (pada array terurut) jauh lebih cepat.
3. **Peningkatan:** Untuk optimasi ringan, hindari memanggil fungsi pencarian dua kali—gunakan variabel hasil yang sama. Jika data sering diubah, pertimbangkan struktur data lain (hash table) atau sort + binary search untuk pencarian yang repetitif.

## Binary Search:

```
1  #include "stdio.h"
2
3  void bubbleSort(int array[], int size) {
4      for(int i = 0; i < size-1; i++) {
5          int swapped = 0;
6          for(int j = 0; j < size-i-1; j++) {
7              if(array[j] > array[j+1]) {
8                  int temp = array[j];
9                  array[j] = array[j+1];
10                 array[j+1] = temp;
11                 swapped = 1;
12             }
13         }
14         if(!swapped) break;
15     }
16 }

18 int binarySearch(int array[], int size, int key) {
19     int indexAwal = 0;
20     int indexAkhir = size - 1;
21
22     while (indexAwal ≤ indexAkhir) {
23         int indexTengah = (indexAwal + indexAkhir) / 2;
24
25         if (array[indexTengah] == key) {
26             return indexTengah;
27         } else if (array[indexTengah] < key) {
28             indexAwal = indexTengah + 1;
29         } else {
30             indexAkhir = indexTengah - 1;
31         }
32     }
33
34     return -1;
35 }
```

```

38 int main() {
39     int size = 10;
40     int array[size];
41     int key;
42
43     /*===== Input =====*/
44     printf( format: "Masukkan key: ");
45     scanf( format: "%d", &key);
46
47     printf( format: "Masukkan Data \n");
48     for (int i = 0; i < size; i++) {
49         printf( format: "Data ke-%d: ", i + 1);
50         scanf( format: "%d", &array[i]);
51     }
52     /*===== Input =====*/
53
54
55     /*===== Output =====*/
56     bubbleSort(array, size);
57     printf( format: "\nData yang telah dimasukkan dan disorting: ");
58     for (int i = 0; i < size; i++) {
59         printf( format: "%d ", array[i]);
60     }
61
62     int index = binarySearch(array, size, key);
63     if (index != -1) {
64         printf( format: "\nKey %d telah ditemukan di Index ke %d", key, binarySearch(array, size, key));
65     } else {
66         printf( format: "\nKey %d tidak di temukan", key);
67     }
68
69     /*===== Output =====*/
70
71     printf( format: "\n");
72     return 0;
73 }
74

```

## Output:

```

Run: Tugas_7 x
Masukkan key: 52
Masukkan Data
Data ke-1: 61
Data ke-2: 26
Data ke-3: 1
Data ke-4: 2
Data ke-5: 6
Data ke-6: 6
Data ke-7: 7
Data ke-8: 3
Data ke-9: 52
Data ke-10: 5

Data yang telah dimasukkan dan disorting: 1 2 3 5 6 6 7 26 52 61
Key 52 telah ditemukan di Index ke 8

Process finished with exit code 0

```

## Penjelasan:

Program ini menggabungkan dua algoritma klasik: Bubble Sort untuk mengurutkan array berukuran tetap (size = 10) dan Binary Search untuk mencari suatu nilai kunci (key) di dalam

array yang sudah terurut. Setelah pengguna memasukkan 10 bilangan bulat, fungsi bubbleSort akan menata elemen-elemen array dari nilai terkecil ke terbesar dengan melakukan perbandingan berulang dan pertukaran jika elemen bersebelahan tidak dalam urutan yang benar. Begitu array terurut, program mencetak hasil pengurutan, lalu memanggil fungsi binarySearch untuk memeriksa apakah key ada di dalam array—mengembalikan indeks jika ditemukan, atau -1 jika tidak.

### Pembahasan Kompleksitas dan Keandalan

- **Bubble Sort** memiliki kompleksitas waktu terburuk dan rata-rata sebesar  $O(n^2)$ , sehingga tidak efisien untuk data berukuran besar, meski pada array kecil ( $n = 10$ ) overhead-nya masih dapat ditoleransi. Implementasi dengan flag swapped membuatnya *adaptive*: jika array sudah terurut sebelum iterasi terakhir, loop akan berhenti lebih awal.
- **Binary Search** bekerja dalam  $O(\log n)$  karena setiap kali membagi ruang pencarian menjadi dua. Namun, ia mengharuskan data **sudah terurut** terlebih dahulu—oleh karena itu, pengurutan Bubble Sort menjadi langkah wajib sebelum mencari. Pastikan pula untuk menyimpan hasil binarySearch dalam variabel (seperti index) agar tidak memanggil fungsi dua kali dan menghindari perhitungan ganda.

### Kesimpulan:

1. Kombinasi Bubble Sort + Binary Search cocok untuk array kecil dimana kemudahan implementasi lebih penting daripada performa mutlak.
2. Untuk data besar, sebaiknya gunakan algoritma pengurutan lebih cepat (e.g. Quick Sort atau Merge Sort) dan lalu lakukan Binary Search, agar keseluruhan *pipeline* pencarian menjadi lebih efisien.
3. Praktik baik: simpan sekali hasil binarySearch ke variabel untuk kemudian dipakai di *output*, serta pertimbangkan validasi atau penanganan kesalahan masukan pengguna sebelum memanggil algoritma.