Laporan Praktikum 2  
Praktikum Analisis Algoritma



Imron Madani (140810170061)

Fakultas Matematika dan Ilmu Pengetahuan Alam  
Program Studi Teknik Informatika  
Universitas Padjajaran  
2019

# Studi Kasus 1: Pencarian Nilai Maksimal

Buatlah programnya dan hitunglah kompleksitas waktu dari algoritma berikut:

**Algoritma Pencarian Nilai Maksimal**

|  |
| --- |
| procedure CariMaks(input x1, x2, …, xn: integer, output maks: integer)  { Mencari elemen terbesar dari sekumpulan elemen larik integer x1, x2, …, xn. Elemen terbesar akan disimpan di dalam maks  Input: x1, x2, …, xn  Output: maks (nilai terbesar)  }  **Deklarasi**  i : integer  **Algoritma**  maks 🡨 x1  i 🡨 2  while i ≤ n do  if xi > maks then  maks 🡨 xi  endif  i 🡨 i + 1  endwhile |

Jawaban:

|  |
| --- |
| #include <iostream>  using namespace std;  int main(){  int max,n,x[100],i;  cout<<"Banyak elemen : "; cin>>n;  for (int i=0; i<n;){  cout<<"Input elemen ke -"<<++i<<" : "; cin>>x[i];  }  max=x[1];  i=2;  do{  if(x[i]>max){  max=x[i];  }  i=i+1;  }while(i<=n);  cout<<"Nilai maksimum : "<<max;  } |

**PEMBAGIAN KOMPLEKSITAS WAKTU**

Hal lain yang harus diperhatikan dalam menghitung kompleksitas waktu suatu algoritma adalah parameter yang mencirikan ukuran input. Contoh pada algoritma pencarian, waktu yang dibutuhkan untuk melakukan pencarian tidak hanya bergantung pada ukuran larik () saja, tetapi juga bergantung pada nilai elemen () yang dicari.

Misalkan:

* Terdapat sebuah larik dengan panjang elemen 130 dimulai dari
* Asumsikan elemen-elemen larik sudah terurut. Jika , maka waktu pencariannya lebih cepat 130 kali dari pada atau tidak ada di dalam larik.
* Demikian pula, jika , maka waktu pencariannya ½ kali lebih cepat daripada

Oleh karena itu, kompleksitas waktu dibedakan menjadi 3 macam:

1. : kompleksitas waktu untuk kasus terbaik (***best case***)

merupakan kebutuhan waktu minimum yang diperlukan algoritma sebagai fungsi dari .

1. : kompleksitas waktu untuk kasus rata-rata (***average case***)

merupakan kebutuhan waktu rata-rata yang diperlukan algoritma sebagai fungsi dari . Biasanya pada kasus ini dibuat asumsi bahwa semua barisan input bersifat sama. Contoh pada kasus *searching* diandaikan data yang dicari mempunyai peluang yang sama untuk tertarik dari larik.

1. : kompleksitas waktu untuk kasus terburuk (***worst case***)

merupakan kebutuhan waktu maksimum yang diperlukan algoritma sebagai fungsi dari .

# Studi Kasus 2: *Sequential Search*

Diberikan larik bilangan bulan yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan rata-rata dari algoritma pencarian beruntun (*sequential search*). Algoritma *sequential search* berikut menghasilkan indeks elemen yang bernilai sama dengan y. Jika y tidak ditemukan, indeks 0 akan dihasilkan.

|  |
| --- |
| procedure SequentialSearch(input : integer, y : integer, output idx : integer)  { Mencari di dalam elemen . Lokasi (indeks elemen) tempat ditemukan diisi ke dalam idx. Jika tidak ditemukan, makai idx diisi dengan 0.  Input:  Output: idx  }  **Deklarasi**  i : integer  found : boolean {bernilai true jika y ditemukan atau false jika y tidak ditemukan}  **Algoritma**  i 🡨 1  found 🡨 false  while (i ≤ n) and (not found) do  if xi = y then  found 🡨 true  else  i 🡨 i + 1  endif  endwhile  {*i < n or found*}  If found then {*y ditemukan*}  idx 🡨 i  else  idx 🡨 0 {y tidak ditemukan}  endif |

Jawaban:

|  |
| --- |
| #include <iostream>  using namespace std;  main(){  int i, x[99],y,idx,n;  bool found=false;  cout<<"Input Jumlah Array : ";cin>>n;  i=1;  while (i<=n){  cout<<"Input Bilangan kedalam Array : ";cin>> x[i];  i++;  }  i=1;  cout<<"Input Bilangan Yang Dicari : ";cin>>y;  while(i<n && found==false)  {  if (x[i] == y)  {  found=true;  }  else  i=i+1;  }  if (found==true)  idx=i;  else  idx=0;  cout<<"Bilangan berada di index ke-"<<idx;  } |

# Studi Kasus 3: *Binary Search*

Diberikan larik bilangan bulan yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan rata-rata dari algoritma pencarian bagi dua (*binary search*). Algoritma *binary search* berikut menghasilkan indeks elemen yang bernilai sama dengan y. Jika y tidak ditemukan, indeks 0 akan dihasilkan.

|  |
| --- |
| procedure BinarySearch(input : integer, x : integer, output : idx : integer)  { Mencari y di dalam elemen . Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx. Jika y tidak ditemukan makai dx diisi dengan 0.  **Input:**  **Output: idx**  }  **Deklarasi**  i, j, mid : integer  found : Boolean  **Algoritma**  i 🡨 1  j 🡨 n  found 🡨 false  while (not found) and ( i ≤ j) do  mid 🡨 (i + j) div 2  if xmid = y then  found 🡨 true  else  if xmid < y then {*mencari di bagian kanan*}  i 🡨 mid + 1  else {*mencari di bagian kiri*}  j 🡨 mid – 1  endif  endif  endwhile  {*found or i > j* }  If found then  Idx 🡨 mid  else  Idx 🡨 0  Endif |

Jawaban:

|  |
| --- |
| #include <iostream>  using namespace std;  int main(){  int i,j,mid,n,x[100],y,idx;  bool found=false;    cout<<"Banyak elemen : ";cin>>n;  for (int i=0; i<n;){  cout<<"Input elemen ke -"<<++i<<" : ";cin>>x[i];  }  i=1;j=n;  cout<<"Cari elemen : ";cin>>y;  do{  mid=(i+j)/2;  if (x[mid]==y)  found=true;  else if (x[mid]<y)  i=mid+1;  else  j=mid-1;  }while(found==false&&i<=j);    if (found==true)  idx=mid;  else  idx=0;    cout<<"Bilangan berada di index ke - "<<idx;  } |

# Studi Kasus 4: Insertion Sort

1. Buatlah program insertion sort dengan menggunakan bahasa C++
2. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma insertion sort.
3. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma insertion sort.

|  |
| --- |
| procedure InsertionSort(input/output : integer)  { Mengurutkan elemen-elemen dengan metode insertion sort.  Input:  OutputL (sudah terurut menaik)  }  **Deklarasi**  i, j, insert : integer  **Algoritma**  for i 🡨 2 to n do  insert 🡨 xi  j 🡨 i  while (j < i) and (x[j-i] > insert) do  x[j]🡨 x[j-1]  j🡨j-1  endwhile  x[j] = insert  endfor |

Jawaban:

|  |
| --- |
| #include <iostream>  #include <conio.h>  using namespace std;  int data[10],data2[10];  int n;  void tukar(int a, int b)  {  int t;  t = data[b];  data[b] = data[a];  data[a] = t;  }  void insertion\_sort()  {  int temp,i,j;  for(i=1;i<=n;i++){  temp = data[i];  j = i -1;  while(data[j]>temp && j>=0){  data[j+1] = data[j];  j--;  }  data[j+1] = temp;  }  }  int main()  {  cout<<"Masukkan Jumlah Data : "; cin>>n;  for(int i=1;i<=n;i++)  {  cout<<"Masukkan data ke- "<<i<<" : ";  cin>>data[i];  data2[i]=data[i];  }    insertion\_sort();  cout<<"Data Setelah di Sort : ";  for(int i=1; i<=n; i++)  {  cout<<" "<<data[i];  }  cout<<"\n\nSorting dengan insertion sort Selesai";  getch();  } |

# Studi Kasus 5: Selection Sort

1. Buatlah program selection sort dengan menggunakan bahasa C++
2. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma selection sort.
3. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma insertion sort.

|  |
| --- |
| procedure SelectionSort(input/output : integer)  { Mengurutkan elemen-elemen dengan metode selection sort.  Input:  OutputL (sudah terurut menaik)  }  **Deklarasi**  i, j, imaks, temp : integer  **Algoritma**  for i 🡨 n downto 2 do {*pass sebanyak n-1 kali*}  imaks 🡨 1  for j 🡨 2 to i do  if xj > ximaks then  imaks 🡨 j  endif  endfor  {pertukarkan ximaks dengan xi}  temp 🡨 xi  xi 🡨 ximaks  ximaks 🡨 temp  endfor |

Jawaban:

|  |
| --- |
| #include <iostream>  using namespace std;  int data[10],data2[10];  int n;  void tukar(int a, int b)  {  int t;  t = data[b];  data[b] = data[a];  data[a] = t;  }  void selection\_sort()  {  int pos,i,j;  for(i=1;i<=n-1;i++)  {  pos = i;  for(j = i+1;j<=n;j++)  {  if(data[j] < data[pos]) pos = j;  }  if(pos != i) tukar(pos,i);  }  }  int main()  {  cout<<"===PROGRAM SELECTION SORT==="<<endl;    cout<<"Masukkan Jumlah Data : ";cin>>n;  for(int i=1;i<=n;i++)  {  cout<<"Masukkan data ke-"<<i<<" : ";  cin>>data[i];  data2[i]=data[i];  }    selection\_sort();  cout<<"Data Setelah di Sort : ";  for(int i=1; i<=n; i++)  {  cout<<" "<<data[i];  }  cout<<"\n\nSorting dengan selection sort Selesai";  getch();  } |