Laporan Praktikum II

Analisis Algoritma



|  |  |
| --- | --- |
| Disusun Oleh :  Dzikri Algiffari 140810180053 | |
|  |  |

**Worksheet 2**

# 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 ß x1 i ß 2  while i ≤ n do  if xi > maks then  maks ß xi  endif  i ß i + 1 endwhile |

Jawaban Studi Kasus 1

T(n) = 2(n – 2) + (n – 2) + 2

= 3 n - 4

Studi Kasus 2: *Sequential Search*

Diberikan larik bilangan bulan *x1, x2, …, xn* yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan ratarata 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 *x1, x2, …, xn* : integer, y : integer, output idx : integer)  { Mencari di dalam elemen *x1, x2, …, xn*. Lokasi (indeks elemen) tempat ditemukan diisi ke dalam idx. Jika tidak ditemukan, makai idx diisi dengan 0.  Input *x1, x2, …, xn*  Output: idx  } |
| **Deklarasi**  i : integer  found : boolean {bernilai true jika y ditemukan atau false jika y tidak ditemukan} **Algoritma**  i ß 1  found false  while (i ≤ n) and (not found) do  if xi = y then  if xi = y then  found ß true  else  i ß i + 1 endif  endwhile  {*i < n or found*}    If found then {*y ditemukan*}  idx ß i  else  idx ß 0 {y tidak ditemukan} endif |

|  |
| --- |
| Jawaban Studi Kasus 2  1*. Kasus terbaik*: ini terjadi bila *a*1 = *x.*  *T*min(*n*) =1  2. *Kasus terburuk*: bila *an* = *x* atau *x* tidak ditemukan.    *T*max(*n*) = *n*  3. *Kasus rata-rata*: Jika *x* ditemukan pada posisi ke-*j*, maka operasi perbandingan (ak = x)akan dieksekusi sebanyak *j* kali.  *T*avg(*n*) = |

Studi Kasus 3: *Binary Search*

Diberikan larik bilangan bulan *x1, x2, …, xn* yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan ratarata 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**   1. ß 1 2. ß n   found ß false  while (not found) and ( i ≤ j) do mid ß (i + j) div 2  if xmid = y then  found ß true else | |
| if xmid < y then | | {*mencari di bagian kanan*} | |
| 1. ß mid + 1   else   1. ß mid – 1   endif  endif  endwhile  {*found or i > j* }  If found then  Idx ß mid  else  Idx ß 0  endif | | {*mencari di bagian kiri*} | |

|  |
| --- |
| Jawaban Studi Kasus 3  1*. Kasus terbaik*  *T*min(*n*) = 1  2. *Kasus terburuk*:  *T*max (*n*) = 2log *n* |

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 *x1, x2, …, xn* : integer)  { Mengurutkan elemen-elemen *x1, x2, …, xn* dengan metode insertion sort.  Input: *x1, x2, …, xn*  OutputL *x1, x2, …, xn* (sudah terurut menaik)  }  **Deklarasi**  i, j, insert : integer **Algoritma**  for i ß 2 to n do  insert ß xi  j ß i  while (j < i) and (x[j-i] > insert) do  x[j]ß x[j-1]  jßj-1 endwhile  x[j] = insert endfor |

|  |
| --- |
| Jawaban Studi Kasus 4  Loop sementara dijalankan hanya jika i> j dan arr [i] <arr [j]. Jumlah total iterasi loop sementara (Untuk semua nilai i) sama dengan jumlah inversi.  Kompleksitas waktu keseluruhan dari jenis penyisipan adalah O (n + f (n)) di mana f (n) adalah jumlah inversi.Jika jumlah inversi adalah O(n), maka kompleksitas waktu dari jenis penyisipan adalah O(n).  Dalam kasus terburuk, bisa ada inversi n \* (n-1) / 2. Kasus terburuk terjadi ketika array diurutkan dalam urutan terbalik. Jadi kompleksitas waktu kasus terburuk dari jenis penyisipan adalah O (n2). |

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 *x1, x2, …, xn* : integer)  { Mengurutkan elemen-elemen *x1, x2, …, xn* dengan metode selection sort.  Input *x1, x2, …, xn*  OutputL *x1, x2, …, xn* (sudah terurut menaik)  }  **Deklarasi**  i, j, imaks, temp : integer **Algoritma**  for i ß n downto 2 do {*pass sebanyak n-1 kali*}  imaks ß 1  for j ß 2 to i do if xj > ximaks then  imaks ß j endif endfor  {pertukarkan ximaks dengan xi}  temp ß xi xi ß ximaks ximaks ß temp endfor |

|  |
| --- |
| Jawaban Studi Kasus 5   1. Jumlah operasi perbandingan element. Untuk setiap *pass* ke-*i*,   *i*= 1 –>  jumlah perbandingan  = *n* – 1  *i* = 2 –>  jumlah perbandingan = *n* – 2  *i*= 3  –> jumlah perbandingan = *n* – 3  :  *i* = *k* –>  jumlah perbandingan = *n* – *k*  :  *i*= *n* – 1  –> jumlah perbandingan = 1  Jumlah seluruh operasi perbandingan elemen-elemen larik adalah    *T*(*n*) = (*n* – 1) + (*n* – 2) + … + 1  Ini adalah kompleksitas waktu untuk kasus terbaik dan terburuk, karena algoritma Urut tidak bergantung pada  batasan apakah data masukannya sudah terurut atau acak.   1. Jumlah operasi pertukaran   Untuk setiap*i* dari 1 sampai *n* – 1, terjadi satu kali pertukaran elemen, sehingga jumlah operasi pertukaran seluruhnya   adalah  *T*(*n*) = *n* – 1.  Jadi, algoritma pengurutan maksimum membutuhkan *n*(*n* – 1 )/2 buah operasi perbandingan elemen dan *n* – 1  buah operasi pertukaran. |