```
procedure CountSORT (input/output T : TabInt, input N : integer)
{ Mengurut tabel integer [1..N] dengan pencacahan }

KAMUS LOKAL
{ ValMin dan ValMax adalah batas minimum dan Maximum harga yg tersimpan
dalam T, harus diketahui }
   TabCount : array [ValMin..Valmax] of integer [0..NMax]
   i : integer { indeks untuk traversal tabel }
   K : integer { jumlah elemen T yang sudah diisi pada proses pembentukan
                 kembali }

ALGORITMA
   { Inisialisasi TabCount }
   i traversal [ValMin..ValMax]
      TabCount_i ← 0
   { Counting }
   i traversal [1..N]
      TabCount_Ti ← TabCount_Ti + 1
   { Pengisian kembali : T_1 ≤ T_2 ... ≤ T_N }
   K ← 0
   i traversal [ValMin..ValMax]
      if (TabCount_i ≠ 0) then
         repeat TabCount_i times
            K ← K + 1
            T_K ← i
```

```
procedure MAXSORT (input/output T : TabInt, input N : integer)
{ Mengurut tabel integer [1..N] terurut mengecil dengan maksimum suksesif }

KAMUS LOKAL
   i : integer     { indeks untuk traversal tabel }
   Pass : integer  { tahapan pengurutan }
   Temp : integer  { memorisasi harga untuk penukaran }
   IMax : integer  { indeks, di mana T [1..pass] bernilai maksimum }

ALGORITMA
   Pass traversal [1..N-1]
      { Tentukan Maximum [Pass..N] }
      IMax ← pass
      i traversal [pass+1..N]
         if (T_IMax < T_i ) then
            IMax ← i
      { T_IMax adalah maximum T[pass..N] }
      { Tukar T_IMax dengan T_Pass }
      Temp ← T_IMax
      T_IMax ← T_Pass
      T_Pass ← Temp
      { T [1..Pass] terurut: T_1 ≥ T_2 ≥ T_3 ≥ ... ≥ T_Pass }
   { Seluruh tabel terurut, T_1 ≥ T_2 ≥ T_3 ..... ≥ T_N }
```

<---- Descending (besar ke kecil)

ascending (kecil ke besar)

```
Pass traversal [1...N-1]
    IMin ← Pass
    i traversal [Pass+1...N]
      if (T_IMin > T_i) then
          IMin ← i

    Temp ← T_IMin
    T_IMin ← T_Pass
    T_Pass ← Temp
```

```
procedure InsertionSORT (input/output T : TabInt, input N : integer)
{ Mengurut tabel integer [1..N] dengan insertion sort }

KAMUS LOKAL
   i : integer       { indeks untuk traversal tabel }
   Pass : integer    { tahapan pengurutan }
   Temp : integer

ALGORITMA
   { T_1 adalah terurut}
   Pass traversal [2..N]
      Temp ← T_Pass        { Simpan  harga T_Pass
                             Supaya tidak tertimpa krn pergeseran  }
      { Sisipkan elemen ke Pass dalam T [1..Pass-1] sambil menggeser: }
      i ← Pass-1
      { Cari i, Temp < T_i and i > 1 }
      while (Temp < T_i) and (i > 1) do
         T_i+1 ← T_i       { Geser }
         i ← i - 1         { Berikutnya }
      { Temp ≥ T_i (tempat yg tepat) or i = 1 (sisipkan sbg. elmt. pertama) }
      depend on (T, i, Temp)
         Temp ≥ T_i   : T_i+1 ← Temp     { Menemukan tempat yg tepat }
         Temp < T_i   : T_i+1 ← T_i
                        T_i ← Temp       { sebagai elemen pertama }
      { T [1..Pass] terurut membesar: T_1 ≤ T_2 ≤ ... ≤ T_Pass }
   { Seluruh tabel terurut, karena Pass = N : T_1 ≤ T_2 ≤ T_3 ≤ ... ≤ T_N }
```

<--- ascending (kecil ke besar)

Descending (besar ke kecil)

```
Pass traversal [2...N]
Temp ← T_pass
i ← Pass - 1

while (Temp > T_i) and (i > 1) do
      T_i+1 ← T_i
        i ← i-1

depend on (T, i, Temp)
   Temp ≤ T_i : T_i+1 ← Temp
   Temp > T_i : T_i+1 ← T_i
                T_i ← Temp
```

```
procedure InsertionSORTWithSentinel(input/output T: TabInt,
                                    input N : integer)
{ mengurut tabel integer [1..N] dgn insertion sort, pencarian dengan
sentinel}
```

```
KAMUS LOKAL
    i : integer      { indeks untuk traversal tabel }
    Pass : integer   { tahapan pengurutan }
    Temp : integer   { Menyimpan harga Tab_Pass spy tidak tertimpa krn
                       pergeseran }
```

```
ALGORITMA
    { T_1 adalah terurut }
    Pass traversal  [2..N]
        { Simpan dulu harga T(Pass) supaya tidak tertimpa karena pergeseran }
        Temp ← T_Pass
        T_0 ← Temp
        { Sisipkan elemen ke Pass dalam T[1..Pass-1] sambil menggeser }
        i ← Pass-1
        { Cari i, Temp < T_i and i > 1 }
        while (Temp < T_i) do
            T_i+1 ← T_i        { Geser}
            i ← i - 1          { Berikutnya }
        { Temp ≥ T_i }
        T_i+1 ← Temp           { Sisipkan }
        { T[1..Pass] terurut: T_1 ≤ T_2 ≤ T_3 ≤ ... ≤ T_Pass }
    { Seluruh tabel terurut, karena Pass = N : T_1 ≤ T_2 ≤ T_3 ≤ ... ≤ T_N }
```

```
procedure BubbleSort (input/output T : TabInt, input N : integer)
{ Mengurut tabel integer [1..N] dengan bubble sort }
```

```
KAMUS LOKAL
    i, K : integer  { indeks untuk traversal tabel }
    Pass : integer  { tahapan pengurutan }
    Temp : integer  { Memorisasi untuk pertukaran harga }
```

```
ALGORITMA
    Pass traversal [1..N-1]
        K traversal [N..Pass+1]
            if (T_K < T_K-1) then
                Temp ← T_K
                T_K ← T_K-1
                T_K-1 ← Temp
    { T [1..Pass] terurut: T_1 ≤ T_2 ≤ T_3 ≤ ... ≤ T_Pass }
{ Seluruh tabel terurut, karena Pass = N: T_1 ≤ T_2 ≤ T_3 ≤ ... ≤ T_N }
```

```
procedure MAX1 (input T : TabInt, input N : integer, output MAX : integer)
{ Pencarian harga Maksimum: }
{ I.S. Tabel tidak kosong, karena jika kosong maka harga maksimum tidak
terdefinisi, N >= 0 }
{ F.S. Menghasilkan harga maksimum MAX pada tabel T [1..N] secara sekuensial
mulai dari indeks 1..Nmax }
```

```
KAMUS LOKAL
    i : integer  { indeks untuk pencarian }
```

```
ALGORITMA
    MAX ← T_1
    i ← 2
    while (i ≤ N) do
        if (MAX < T_i) then
            MAX ← T_i
        i ← i + 1
    { i > N : semua elemen sudah selesai diperiksa }
```

```
procedure BinSearch1 (input T : TabInt, input N : integer,
                      input X : integer, output Found : boolean)
{ Mencari harga X dalam Tabel T [1..N] secara dikhotomik }
{ Hasilnya adalah sebuah boolean Found, true jika ketemu }
{ Nilai elemen tabel terurut membesar: T1 ≤ T2 ≤ T3 ≤ ... ≤ TN }

KAMUS LOKAL
    Atas, Bawah, Tengah : integer { indeks atas, bawah, tengah:
                                    batas pemeriksaan }

ALGORITMA
    Atas ← 1; Bawah ← N      { Batas atas dan bawah seluruh tabel }
    Found ← false            { Mula-mula belum ketemu }
    while (Atas ≤ Bawah) and (not Found) do
        Tengah ← (Atas + Bawah) div 2
        depend on (T, Tengah, X)
            X = TTengah : Found ← true; IX = Tengah
            X < TTengah : Bawah ← Tengah - 1
            X > TTengah : Atas ← Tengah + 1
    { Atas > Bawah or Found, harga Found menentukan hasil pencarian }
```

```
procedure SEQSearchX2 (input T : TabInt, input N : integer,
                       input X : integer, output IX : integer,
                       output Found : boolean )
{ Mencari harga X dalam Tabel T[1..N] secara sekuensial mulai dari T1,
  Hasilnya adalah indeks IX di mana Ti=X (i terkecil),
  IX = 0 jika tidak ketemu dan sebuah boolean Found (true jika ketemu). }

KAMUS LOKAL
    i : integer [1..N+1]  { indeks untuk pencarian }

ALGORITMA
    Found ← false  { awal pencarian, belum ketemu }
    i ← 1
    while (i ≤ N) and (not Found) do
        if (Ti = X) then
            Found ← true
        else
            i ← i + 1
    { i > N or Found }
    if (Found) then
        IX ← i
    else
        IX ← 0
```

```
procedure SEQSearchX1 (input T : TabInt, input N : integer,
                       output IX : integer, input X : integer)
{ Mencari harga X dalam Tabel T [1..N] secara sekuensial mulai dari T1.
  Hasilnya adalah indeks IX di mana TIX = X (i terkecil) }
{ IX = 0 jika tidak ketemu. }

KAMUS LOKAL
    i : integer [1..Nmax]     { indeks untuk pencarian }

ALGORITMA
    i ← 1
    while (i < N) and (Ti ≠ X) do
        i ← i + 1
    { i = N or Ti = X }
    if (Ti = X) then
        IX ← i
    else   { Ti ≠ X }
        IX ← 0
```

```
procedure SEQSearchX1a (input T : TabInt, input N : integer,
                        output Found : boolean)
{ Mencari harga X dalam Tabel T [1..N] secara sekuensial mulai dari T1
Hasilnya adalah sebuah nilai boolean Found (true jika ketemu, false jika
tidak. Pada versi ini informasi yang diperlukan adalah ketemu atau tidak
ketemu, namun kita tidak mendapatkan informasi di mana indeks nilai yang
dicari diketemukan }

KAMUS LOKAL
    i : integer [1..N]  { indeks untuk pencarian }

ALGORITMA
    i ← 1
    while (i < N) and (Ti ≠ X) do
        i ← i + 1
    { i = N or Ti = X }
    Found ← (Ti = X)
```

```
Program NILAIRATA_RATA
{ Model proses sekuensial dengan mark,
  dengan penanganan kasus kosong }

KAMUS
   type rekaman : < NIM : integer, nilai : integer[0..100] >
   ArsipMhs : SEQFILE of
              (*) RekMhs : rekaman { setiap mahasiswa punya 1 rekaman }
              (1) <9999999, 99>
   SumNil : integer  { jumlah nilai }
   JumMhs : integer  { jumlah mahasiswa }

ALGORITMA
   OPEN(ArsipMhs, RekMhs)              { First_Elmt }
   if (RekMhs.NIM = 9999999) then
      output ("Arsip kosong")
   else
      SumNil ← 0; JumMhs ← 0           { Inisialisasi }
      repeat
         SumNil ← Sumnil + RekMhs.nilai; JumMhs ← JumMhs + 1     { Proses }
         READ(ArsipMhs,RekMhs)         { Next_Elmt }
      until (RekMhs.NIM = 9999999)     { EOP }
      output (Sum/JumMhs)              { Terminasi }
   CLOSE(ArsipMhs)
```

```
Program KONSOLIDASITanpaSeparator
{ Dengan penanganan kasus kosong }
{ Input : sebuah arsip sekuensial berisi NIM dan nilai mahasiswa }
{ Proses : Mengelompokkan setiap kategori dan memrosesnya : menghitung nilai
rata-rata setiap mahasiswa dan nilai rata-rata seluruh populasi mahasiswa }
{ Output : NIM Nilai rata-rata setiap mahasiswa, dan nilai rata-rata seluruh
mahasiswa }

KAMUS
   type Keytype : integer
   type Valype  : integer [0..100]
   type rekaman : < NIM   : keytype,    { kunci }
                    Nilai : valtype     { harga lain yang direkam } >
   ArsipMhs : SEQFILE of    { input, terurut menurut kunci }
                (*) RekMhs : rekaman
                (1) <9999999, 0>
   Current_NIM : integer    { identifikasi kategori yg sedang diproses }
   SumNil : integer         { Jumlah nilai seluruh matakuliah seorg mhs }
   NKuliah : integer        { Jumlah matakuliah seorang mahasiswa }
   NilRata : integer        { Nilai rata-rata seorang mahasiswa }
   SumNilTot : integer      { Jumlah nilai seluruh matakuliah seorg mhs }
   NMhs : integer           { Jumlah matakuliah seluruh mahasiswa }

ALGORITMA
   OPEN(ArsipMhs ,RekMhs)               { First_Elmt }

   if (RekMhs.NIM = 9999999) then       { EOP }
      output ("Arsip kosong")
   else

      repeat

         { Proses satu kategori = 1 NIM }
         SumNil ← 0; NKuliah ← 0       { Init_Categ }
         Current_NIM ← RekMhs.NIM
         repeat
            SumNil ← SumNil + RekMhs.Nilai   { Proses }
            NKuliah ← NKuliah + 1            { Proses_Current_Categ }
            READ(ArsipMhs, RekMhs)
         until (Current_NIM ≠ RekMhs.NIM)
         { NIM ≠ Current_NIM, RekMhs.NIM = elemen pertama Next_Categ }
         NilRata ← SumNil/NKuliah
         SumNilTot ← SumNilTot + NilRata
         NMhs ← NMhs + 1
         output (Current_NIM,NilRata)         { Terminasi_Categ }

      until (RekMhs.NIM = 9999999)            { EOP }

      output (SumNilTot/NMhs)                 { terminasi seluruh file }

   CLOSE (ArsipMhs)
```

```
Program KATATERPANJANG
{ Input  : sebuah arsip sequential, yang mewakili sebuah teks, }
{          setiap rekaman adalah sebuah karakter }
{ Proses : Menghitung kata terpanjang dalam teks }
{ Output : Panjang kata maksimum }
```

KAMUS

```
    { Keytype : type dari elemen arsip yg menentukan apakah elemen tsb.
                separator atau bukan}
    { Valtype adalah type dari harga rekaman, di sini tidak ada }
    KeyType : character
    constant mark  : character = '.'
    constant blank : character = ' '
    type rekaman : Keytype
    ArsipIn : SEQFILE of  { input, terurut menurut kunci }
              (*) CC : rekaman   { CC : sebuah karakter yang direkam }
              (1) <'.'>          { mark }
    PanjangKata : integer        { Panjang kata yang sedang dalam proses }
    MaxLength : integer          { Panjang kata maksimum }
    procedure Inisialisasi
    procedure Init_Categ         { Inisialisasi untuk satu kategori }
    procedure Proses_Current_Categ       { Proses terhadap sebuah elemen
                                           kategori }
    procedure Terminasi_Categ    { Terminasi sebuah  kategori }
    function Separator (K : Keytype) → boolean
    { True jika K adalah separator, di sini adalah blank : ' ' }
```

ALGORITMA

```
    OPEN(ArsipIn, CC)                              { First_Elmt }
    if (CC = mark) then
       output ("Arsip kosong")
    else
       Maxlength ← 0     { Diandaikan kata minimum terdiri dari 1 huruf }
       repeat
          { Skip separator, jika ada}
          while (CC ≠ mark) and (CC = blank) do
             { not EOP & Separator(Keyin) }
             READ (ArsipIn, CC)
             { CC bukan Separator, CC adalah huruf pertama dari sebuah kata
               atau Mark }
          { CC = mark or CC ≠ blank }
          PanjangKata ← 0
          while (CC ≠ mark) and (CC ≠ blank ) do
             { not Separator(KeyIn) and not EOP }
             PanjangKata ← PanjangKata + 1
             { Proses_Current_Categ }
             READ (ArsipIn, CC)
          { CC = blank or CC = mark }
          if (MaxLength < PanjangKata)  then
             MaxLength ← Panjang Kata          { Terminasi_Categ }
       until (CC = mark)                       { EOP }
       { Terminasi proses, Maxlength = 0 berarti Arsip hanya berisi blank }
       output (MaxLength)
    CLOSE(ArsipIn)
```

```
Program MERGING2
{ Input  : Dua arsip sequential, terurut menaik menurut kunci, sejenis, dan
           semua harga <Mark> }
{ Proses : Menggabung kedua arsip menjadi sebuah arsip yg terurut }
{          VERSI  OR }
{ Output : Sequential file baru yang terurut }
```

KAMUS

```
    { Keytype adalah suatu type dari kunci rekaman,
      Valtype adalah type dari harga rekaman }
    { Keytype dan Valtype harus terdefinisi
      Akhir arsip ditandai oleh mark dan Val }
    type rekaman : < Key : keytype,      { kunci }
                     Val : valtype       { harga lain yang direkam } >
    ArsipIn1 : SEQFILE of    { input, terurut menurut kunci }
                  (*) RekIn1 : rekaman
                  (1) <mark, Val>
    ArsipIn2 : SEQFILE of    { input, terurut menurut kunci }
                  (*) RekIn2 : rekaman
                  (1) <mark, Val>
    ArsipOut : SEQFILE of    { output, terurut menurut kunci }
                  (*) RekOut : rekaman
                  (1) <mark, Val>
```

ALGORITMA

```
    OPEN(ArsipIn1, RekIn1)          { First_Elmt of Arsip1 }
    OPEN(ArsipIn2, RekIn2)          { First_Elmt of Arsip2 }
    REWRITE(ArsipOut)               { Menyiapkan arsip hasil : Arsip3 }
    while (RekIn1.Key ≠ mark) or (RekIn2.Key ≠ mark) do
       depend on (RekIn1.Key, RekIn2.Key)
          RekIn1.Key ≤ RekIn2.Key : WRITE(ArsipOut, RekIn1)
                                    READ(ArsipIn1, RekIn1)
          RekIn1.Key > RekIn2.Key : WRITE(ArsipOut, RekIn2)
                                    READ(ArsipIn2, RekIn2)
    { RekIn1.Key = mark and RekIn2.Key = mark }
    WRITE(ArsipOut, <mark,Val>)
    CLOSE(ArsipIn1)
    CLOSE(ArsipIn2)
    CLOSE(ArsipOut)
```