

FORM 5 PASCAL PROGRAMMING

Unit 11: Searching, Sorting and Merging

March, 01

11.1 SEARCHING (搜尋)

- refers to the problem of finding a particular element from a list

Example:

```
program SearchingProblem;
const MAX = 10; (* Number of students *)
var
  Name : array[1..Max] of string;
  Score : array[1..Max] of integer;
  Target : string;
  Found : boolean;
  Index : integer;

procedure ReadData;
var
  K : integer;
  InFile : text;
begin
  assign( InFile, 'Students.dat' );
  reset( InFile );
  for K := 1 to MAX do
    begin
      readln( InFile, Name[K] );
      readln( InFile, Score[K] );
    end;
  close( InFile );
end;

procedure Search;
:
:
:
begin (* Main program *)
  ReadData; (* Store data in arrays *)
  write( 'Enter name of student : ' );
  readln( Target );
  Search; (* Search for Target in array Name *)
  if Found then
    writeln( 'The score of ', Target, ' is ', Score[Index] )
  else
    writeln( 'Student not found !' );
end.
```

11.1.1 Linear (Sequential) Search and Ordered Linear Search

- examines the data items in order they appear in the list until the desired value is located or the end of the list is reached;
- Comparison

	Flowchart	Program code
Linear Search	<pre> graph TD ENTER([ENTER]) --> Init[K <- 1 Found <- false] Init --> Dec1{Name[K] = Target?} Dec1 -- T --> SetIndex[Index <- K Found <- true] Dec1 -- F --> IncK[K <- K + 1] IncK --> Dec2{K > MAX?} Dec2 -- T --> RETURN([RETURN]) Dec2 -- F --> Dec1 SetIndex --> RETURN </pre>	<pre> (*Linear search*) procedure Search; var K : integer; begin K := 1; Found := false; (*not yet found*) repeat if Name[K] = Target then begin Index := K; Found := true; end else K := K + 1 until Found or (K > MAX) end; </pre>
Ordered Linear Search	<pre> graph TD ENTER([ENTER]) --> Init[K <- 1 Found <- false] Init --> Dec1{Name[K] = Target?} Dec1 -- T --> SetIndex1[Index <- K Found <- true] Dec1 -- F --> Dec2{Name[K] > Target?} Dec2 -- T --> SetIndex2[Index <- K Found <- true] Dec2 -- F --> IncK[K <- K + 1] IncK --> Dec3{K > MAX?} Dec3 -- T --> RETURN([RETURN]) Dec3 -- F --> Dec1 SetIndex1 --> RETURN SetIndex2 --> RETURN </pre>	<pre> (*Ordered linear search*) procedure Search; var K : integer; begin K := 1; Found := false; (*not yet found*) repeat if Name[K] = Target then begin Index := K; Found := true; end else if Name[K] > Target then K := MAX + 1 (*force to exit the loop*) else K := K + 1 until Found or (K > MAX) end; </pre>

11.1.2 Binary Search

- the list is cut in half repeatedly until the target is found or the list is fully searched.
- program code:

```

procedure Search;

(* Binary search :
   Name[] must be already in alphabetical order. *)

var Top, Bottom, Middle : integer;
begin
  Top := 1;
  Bottom := MAX;
  repeat
    Middle := (Top + Bottom) div 2;
    if Target < Name[Middle] then
      Bottom := Middle - 1
    else if Target > Name[Middle] then
      Top := Middle + 1
    else
      begin
        Found := true;
        Index := Middle
      end;
  until Found or (Top > Bottom)
  (* Target is found or list is empty *)
end;

```

11.1.3 Comparison of searching algorithms:

Sequential search	Binary search
Simple algorithm and easy to implement	Algorithm rather complex
The list need not be sorted	The list must be sorted.
Requires many comparisons	Requires very few comparisons
Slow in locating a target	Very fast in locating a target

11.2 SORTING (排序)

- refers to the problem of arranging a given list of elements in order

11.2.1 Brute Force Sort

11.2.2 Insertion Sort

11.2.3 Selection Sort

11.2.4 Bubble Sort ***

11.3 BUBBLE SORT (冒泡分類法)

- starts at the top of the array and compares two adjacent elements. If they are not in the correct order, they are swapped. Then the next pair is compared until this is done for the entire array;

■ Example 1:

Using bubble sort, the numbers 8, 6, 1, 9, 4 will be sorted in this way:

First pass	The working list is: 8, 6, 1, 9, 4				
8 6	1	9	4		Swap 8 and 6 (because $8 > 6$)
6 8	1	9	4		Swap 8 and 1 (because $8 > 1$)
6 1	8	9	4		No swap is needed (because $8 < 9$)
6 1	8	9	4		Swap 9 and 4 (because $9 > 4$)
6 1	8	4	9		The largest element, 9, "bubbles" to the last position

Second pass	The working list is: 6, 1, 8, 4				
6 1	8	4	9		Swap 6 and 1 (because $6 > 1$)
1 6	8	4	9		No swap is needed (because $6 < 8$)
1 6	8	4	9		Swap 8 and 4 (because $8 > 4$)
1 6	4	8	9		Two elements are in proper positions

Third pass	The working list is: 1, 6, 4				
1 6	4	8	9		No swap is needed (because $1 < 6$)
1 6	4	8	9		Swap 6 and 4 (because $6 > 4$)
1 4	6	8	9		Three elements are in proper positions

Fourth pass	The working list is: 1, 4				
1 4	6	8	9		No swap is needed (because $1 < 4$)
1 4	6	8	9		Four elements are in proper positions

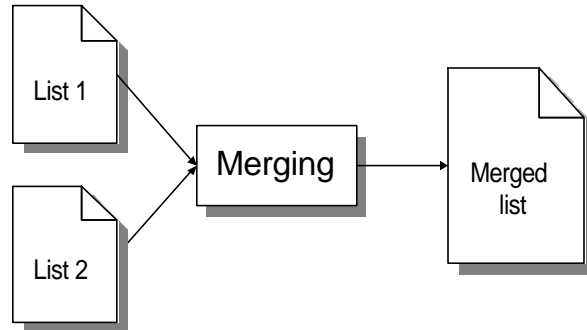
Note that 4 passes are needed. After the fourth pass, the largest 4 elements are in their proper position. In general, to sort a list of N elements, at most N - 1 passes are needed.

■ Program Codes:

Normal Bubble Sort	Enhanced Bubble Sort
<pre> program Sorting; const Max = 100; type ElementType = integer; ArrayType = array[1..Max] of ElementType; var i : integer; List : ArrayType; procedure BubbleSort(var List : ArrayType; N : integer); var Pass, i : integer; Temp : ElementType; begin for Pass := 1 to N - 1 do for i := 1 to N - Pass do if List[i] > List[i+1] then begin Temp := List[i]; List[i] := List[i+1]; List[i+1] := Temp; end end; end; begin for i := 1 to 10 do List[i] := random(100); BubbleSort(List, 10); for i := 1 to 10 do writeln(List[i]) end. </pre>	<pre> program ImprovedSorting; const Max = 100; type ElementType = integer; ArrayType = array[1..Max] of ElementType; var i : integer; List : ArrayType; procedure BubbleSort(var List : ArrayType; N : integer); var Pass, i : integer; Temp : ElementType; NoExchange : boolean; begin Pass := 1; repeat NoExchange := true; for i := 1 to N - Pass do if List[i] > List[i+1] then begin Temp := List[i]; List[i] := List[i+1]; List[i+1] := Temp; NoExchange := false; end; Pass := Pass + 1; until NoExchange = true; end; begin for i:=1 to 10 do List[i] := random(100); BubbleSort(List, 10); for i:=1 to 10 do writeln(List[i]) end. </pre>

11.4 MERGING (合併)

- refers to the problem of combining two or more sorted lists into one single sorted list so that the elements in the new list are also in the correct order.



- Example: To merge elements from $A[]$ and $B[]$ into a new array $C[]$, the algorithm is as follows:

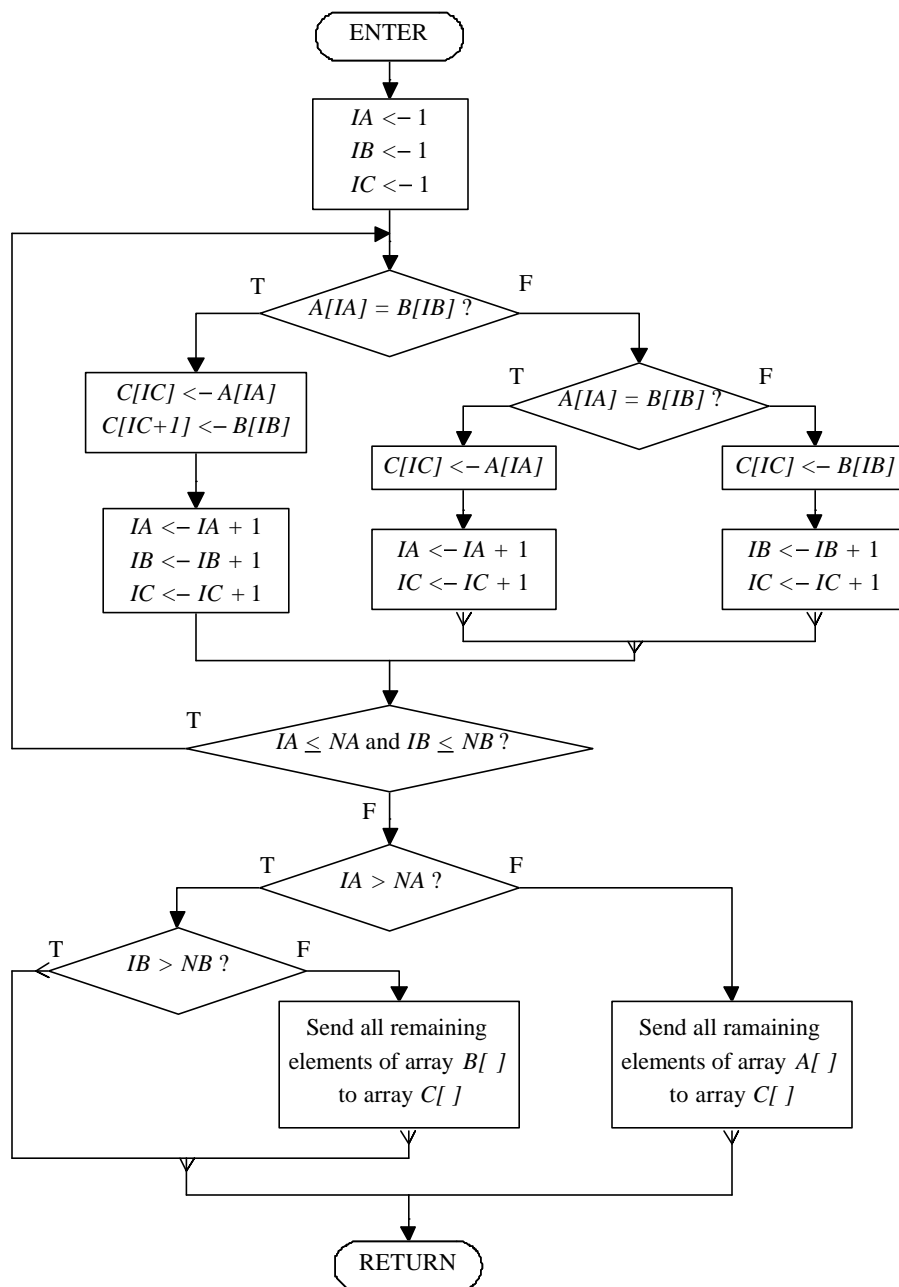


Fig. 18.1

```

program Merging;
const
  NA = 3;
  NB = 5;
  NC = 8;
var
  I : integer;
  A : array [1..NA] of integer;
  B : array [1..NB] of integer;
  C : array [1..NC] of integer;

procedure Merge;
var IA, IB, IC, J : integer;
begin
  IA := 1;
  IB := 1;
  IC := 1;
  repeat
    if A[IA] = B[IB] then
      begin
        C[IC] := A[IA];
        C[IC+1] := B[IB];
        IA := IA + 1;
        IB := IB + 1;
        IC := IC + 2
      end
    else if A[IA] < B[IB] then
      begin
        C[IC] := A[IA];
        IA := IA + 1;
        IC := IC + 1
      end
    else
      begin
        C[IC] := B[IB];
        IB := IB + 1;
        IC := IC + 1
      end
    end
  until (IA > NA) or (IB > NB);
  if IA > NA then
    (*Send all remaining elements of B[] to C[]*)
    for J := IB to NC do
      C[IC+J-IB] := B[J]
    else
    (*Send all remaining elements of A[] to C[]*)
    for J := IA to NC do
      C[IC+J-IA] := A[J]
  end; (*End of procedure Merge*)

begin (*Main program*)
  writeln( 'Enter the elements of array A' );
  for I := 1 to NA do
    begin
      write( 'Data ', I, ' : ' );
      readln( A[I] )
    end;
  writeln;
  writeln( 'Enter the elements of array B' );
  for I := 1 to NB do
    begin
      write( 'Data ', I, ' : ' );
      readln( B[I] )
    end;
  writeln;
  Merge;
  writeln( 'The numbers after merging are :' );
  for I := 1 to NC do
    write( C[I]:4 );
  writeln
end.

```

end of unit 11