Computer Science 1510

Lecture 15

Lecture Outline

- Sorting
- Searching

Insertion Sort

- Insertion sort reads values and places them in the appropriate location in the array such that when all values have been read the resulting array is sorted.

- Consider the list

```
3, 1, 2, 5, 4, 2, 1, 9
```

- Insertion sort begins with the first element and adds it to the first position in the sorted list.

```
3
```

- The second element is then placed before or after the first element, depending on whether it is less than or greater than the first element. In this case, the second value read is to be placed in position 1, with the 3 moved to the second position.

```
1, 3
```

- The third element is then inserted into its proper place in the sorted list, shifting one or both of the first two elements if necessary. Here, just the 3 is shifted to insert the 2.

```
1, 2, 3
```

- The process continues with insertion of subsequent elements into their appropriate position in the list until all elements have been placed in the sorted list.

```
1, 2, 3, 5
1, 2, 3, 4, 5
1, 2, 2, 3, 4, 5
1, 1, 2, 2, 3, 4, 5
1, 1, 2, 2, 3, 4, 5, 9
```

Example: Insertion Sort

```
PROGRAM Insertion_sort
!---------------------------------------------------------------------
! This program sorts a list of integers in ascending order using
! insertion sort.
!  INPUT:
!    num - the number of integers to be sorted.
!    current - integer read from the user, to be placed in
!              the appropriate location in the sorted array.
!  OUTPUT:
!    to_sort - sorted list of integers.
!---------------------------------------------------------------------
   IMPLICIT NONE
   INTEGER :: to_sort(999)
   INTEGER::i,j,k,num,current,position

   WRITE(*,*) 'How many numbers are to be read?'
   READ(*,*) num
   WRITE(*,*) 'Enter value'
   READ(*,*) current
   WRITE(*,*) 'List thus far:'
   WRITE(*,'(X,I2)') current

   to_sort(1)=current ! Put the first element in the array
   DO k=2,num  ! For each element
      WRITE(*,*) 'Enter value'
      READ(*,*) current ! Get next element
      position=1  ! Assume that the element belongs at the beginning
      DO j=1,k-1  ! Check where element belongs among those already sorted
         IF (current > to_sort(j)) THEN
            position=j+1
         END IF
      END DO
      IF (position < k) THEN  ! Need to shift part or all of the list
         CALL Shift(to_sort,k,position)  ! Shift elements from position to k
         to_sort(position)=current ! Add current element in correct position
```

```
      ELSE
         to_sort(position)=current ! Add element to the end of the list
      END IF
      WRITE(*,*) 'List thus far:'
      DO i=1,k
         WRITE(*,'(X,I2)',ADVANCE='NO') to_sort(i)
      END DO
      WRITE(*,*)
   END DO
   WRITE(*,*)
   WRITE(*,*) 'Final sorted list:'
   DO i=1,num
      WRITE(*,'(X,I2)',ADVANCE='NO') to_sort(i)
   END DO
   WRITE(*,*)

CONTAINS

   SUBROUTINE Shift(array,end,start)
   !-------------------------------------------------------------
   ! Shift the elements of array from index start to index end
   ! ahead by one, leaving a gap at index start.
   !-------------------------------------------------------------
      INTEGER,INTENT(INOUT)::array(:)
      INTEGER,INTENT(IN)::end,start
      INTEGER::i

      DO i=end,start+1,-1
         array(i)=array(i-1)
      END DO
   END SUBROUTINE Shift

END PROGRAM Insertion_sort
```

Searching

- Another common programming problem is searching a list of data for a particular item.

- As with sorting, there are several different algorithms that can be used to search a list.

- We have seen a linear search which starts with the first element in the list and searches sequentially until either the desired item is found or the end of the list is reached.

- A more efficient search is a *binary search*.

- Binary search requires the initial list to be sorted. We first check the middle element in the list. If this is the desired value then the search is complete. Otherwise, if the middle value is greater than the desired value we repeat with the first half of the list, if the middle value is less than the desired value we repeat with the second half of the list.

- Consider the list

```
1, 1, 2, 2, 3, 4, 5, 9, 10
```

- Suppose that we are looking for the value 9. We first check the middle element (= 3).

- Since 3 is less than 9 we disregard the first half of the list and concentrate on the second half.

```
4, 5, 9, 10
```

- Since the second half of the list has 4 elements, there is no middle element. In this case we choose the element preceding the middle which is 5.

- Since 9 is greater than 5 we again choose the second half of the list.

```
9, 10
```

- Since there is again no middle element, we choose the element preceding the middle which is 9. This is the desired value.

Example: Binary Search

```
PROGRAM Binsearch
    IMPLICIT NONE
    INTEGER :: list(999), num, i, to_find, location
    INTEGER :: first, last, mid
    LOGICAL :: found=.FALSE.

    WRITE(*,*) 'How many items are to be entered?'
    READ(*,*) num
    WRITE(*,*) 'Enter the sorted list'
    READ(*,*) (list(i), i=1,num)
    WRITE(*,*) 'What item would you like to find?'
    READ(*,*) to_find

    ! Start with entire list, ie. elements from 1 to num
    first = 1
    last = num

    DO
        ! Terminate if list is empty or if item is found
        IF ((first > last) .OR. found) EXIT
        WRITE(*,*) 'Searching list:'
        DO i=first,last
            WRITE(*,'(X,I3)',ADVANCE='NO') list(i)
        END DO
        WRITE(*,*)

        ! Compute the midpoint.  Note that integer division will
        ! give the item preceding the middle if there is no
        ! middle element.
        mid = (first + last) / 2

        IF (list(mid) > to_find) THEN ! Search first half of list
            last = mid - 1
        ELSE IF (list(mid) < to_find) THEN ! Search second half
            first = mid + 1
```

```
        ELSE ! mid element is the item sought
            found = .TRUE.
            location = mid
        END IF
    END DO

    IF (found) THEN
        WRITE(*,'(A,I4,A,I4)') 'Value ', to_find,' was found at location ',location
    ELSE
        WRITE(*,*) 'Item not found'
    END IF
END PROGRAM Binsearch
```