Problem solving with pointers and arrays

Lecture Topics

- insertion sort
- binary search

Lecture materials

- Textbook Ch. 16

Topics

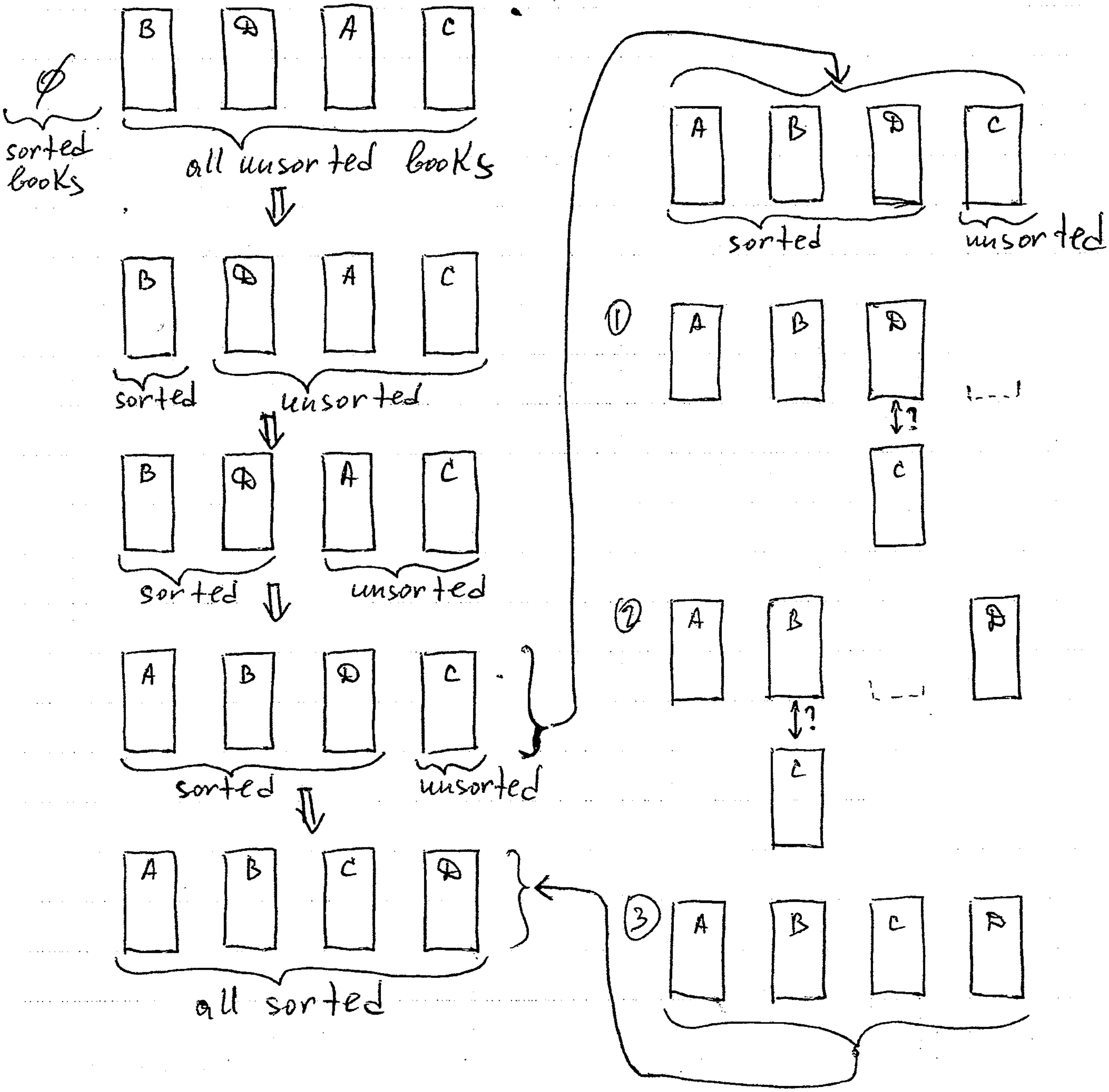
→ Problem solving with arrays
→ Testing and debugging.

Read @ home
Ch 16, Ch. 15

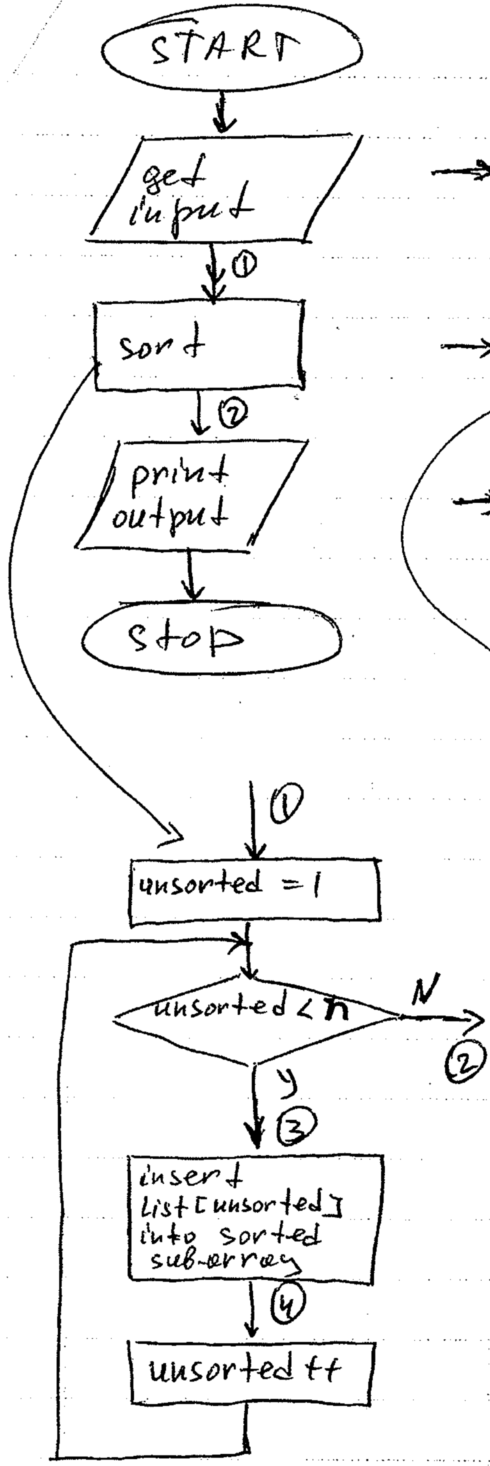
→ Insertion Sort

→ problem statement: order elements of an array a[] such that $a[0] \le a[1] \le a[2] \le \ldots \le a[n-1]$ (ascending order)

→ a bookshelf example



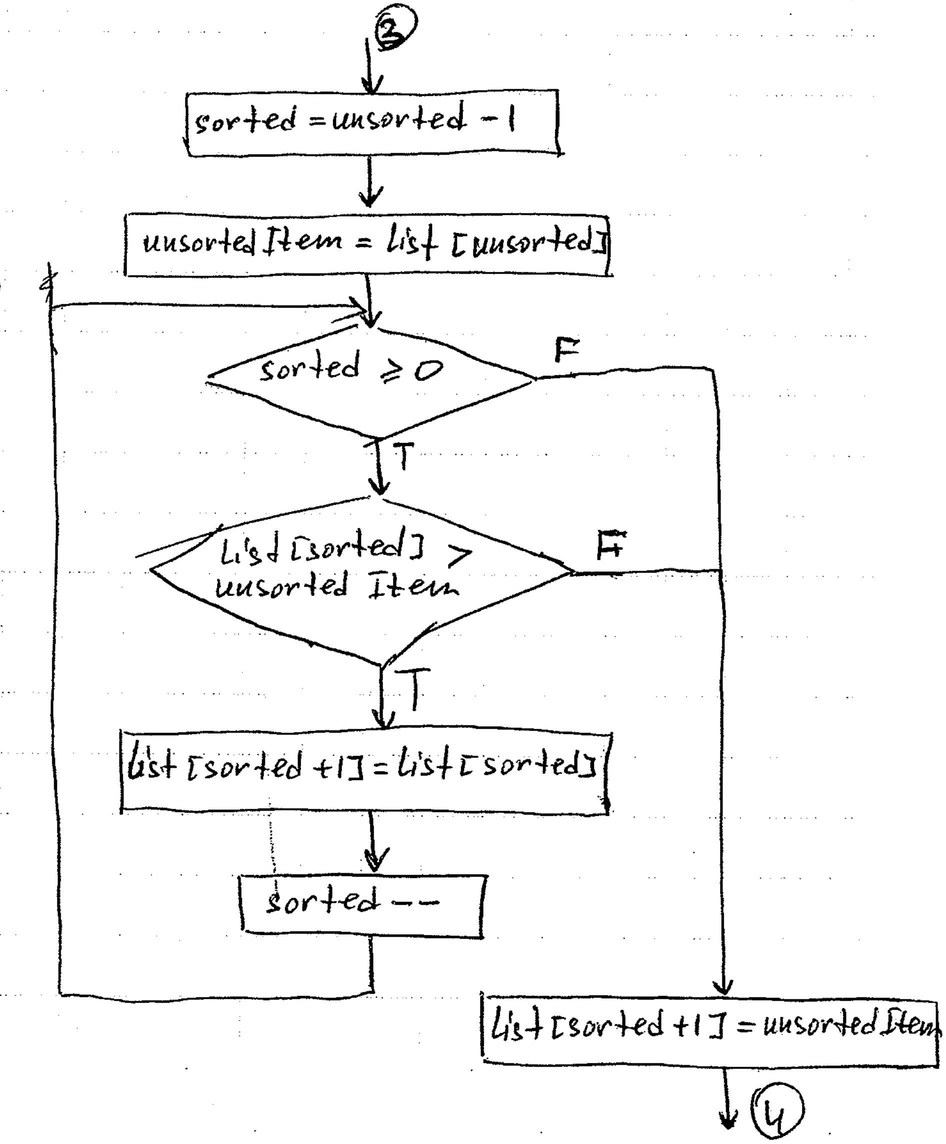
4,20 P2



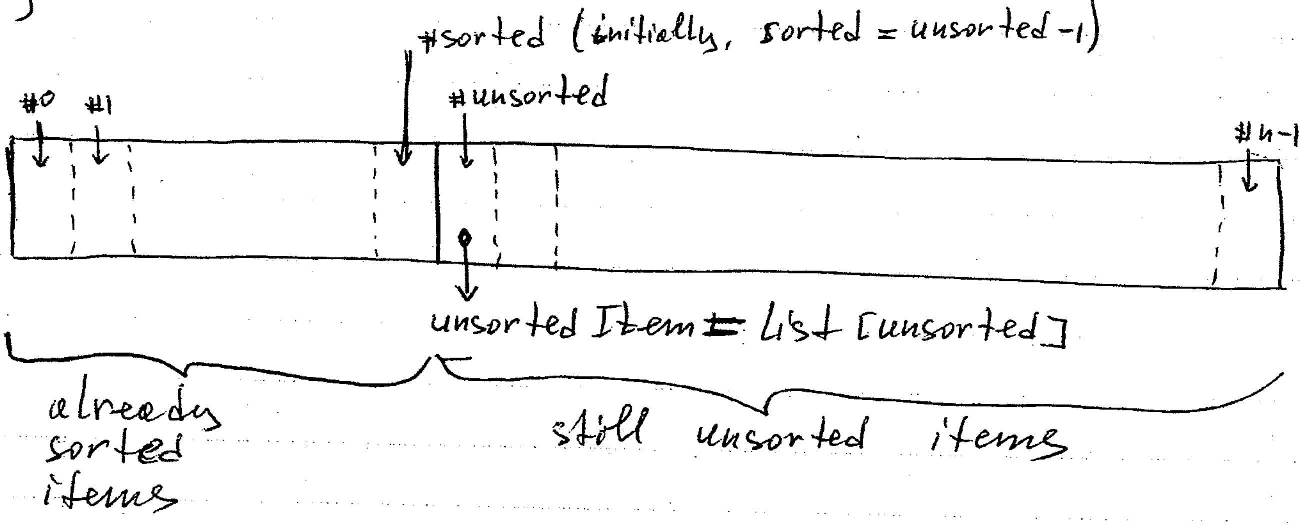
```
#define N 10
→ int numbers [N] = { 3, 1, -1, ... };
                      10 values
→ Insertion Sort(numbers, N);
→ /* print an array of numbers */
```

function prototype:

```
void Insertoon Sort ( int list[], int n);
```


③
sorted = unsorted - 1
unsorted Item = list [unsorted]
sorted ≥ 0
F
T
List [sorted] > unsorted Item
F
T
list [sorted + 1] = list [sorted]
sorted --
List [sorted + 1] = unsorted Item
④

```
void InsertionSort (int list[], int n)
{
    int unsorted, sorted;
    int unsortedItem;
    /* loop through unsorted items */
    for (unsorted = 1; unsorted < n; unsorted++)
    {
        unsortedItem = list[unsorted];   /* pull unsorted item */
        /* Loop through sorted items until we find */
        /* a spot for unsortedItem */
        for (sorted = unsorted-1;
                (sorted >= 0) && (list[sorted] > unsortedItem);
                sorted--)
        {
            List[sorted+1] = List[sorted];
        }
        List[sorted+1] = unsortedItem;   /* insert */
    }
}
```


#sorted (initially, sorted = unsorted -1)
#unsorted
#0
#1
#n-1
unsortedItem = list[unsorted]
already sorted items
still unsorted items

Example: binary search

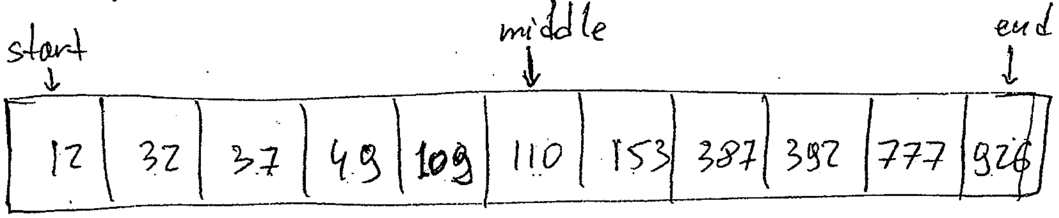
→ given an ordered array of elements, $a[0] \le a[1] \le a[2] \le \dots a[n-1]$, find a particular element, b. Return its index in the array a, or −1 if b does not exist in a.

→ binary search is a very rapid way of accomplishing this task.

→ step 1: examine the mid point of the array, a:

- → if b equals to the value at midpoint, return index of the mid point value
- → if b is less than the value at the midpoint, perform the search again, but only on the elements from the first half of the array
- → if b is larger than the value at the midpoint perform the search on the second half of the array.
- → If a subarray has no elements, return −1.

example array:

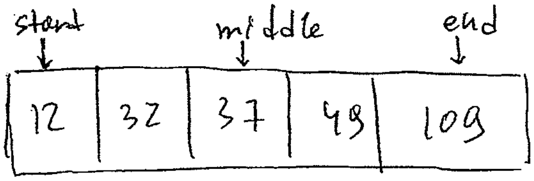


12	32	37	49	109	110	153	387	392	777	926

a

we want to find the index of value 109

step 1: $a[5] > 109$, thus, perform search among elements a[0] to a[4]:



12	32	37	49	109

step 2: $a[2] < 109$, thus, perform seach among elements a[3] to a[4].

and so on.

```
#include <stdio.h>
#include <stdlib.h>

#define N 10

void getData(int array[], int n);
void printData(int array[], int n);
void InsertionSort(int array[], int n);

int main()
{
    int list[N];
    int item, found;

    /* generate random list */
    getData(list, N);
    printData(list, N);

    /* call insertion sort */
    InsertionSort(list, N);
    printData(list, N);

    /* get item from the user */
    printf("Which number do you want to find in the list? ");
    scanf("%d", &item);

    /* call biary search */
    found = BinarySearch(item, list, N);

    printf("Item %d was %sfound in the list\n", item, (found==-1) ? "NOT " : "");

    return 0;
}


void getData(int array[], int n)
{
    int i;

    for (i = 0; i < n; i++)
        array[i] = rand() / 1000000;
}

void printData(int array[], int n)
{
    int i;

    for (i = 0; i < n; i++)
        printf("%d ", array[i]);
    printf("\n");
}
```

```
void InsertionSort(int array[], int n)
{
    int us, s;
    int usItem;

    /* loop through us items */
    for (us = 1; us < n; us++)
    {
        usItem = array[us];

        /* loop through items until we find a spot for usItem */
        for (s = us-1; (s >= 0) && (array[s] > usItem); s--)
            array[s+1] = array[s]; /* shift s items */

        array[s+1] = usItem; /* insert us item */
    }
}
```

```
int BinarySearch(int item, int array[], int n)
{
    int start=0, end=n-1, middle;
    int found = -1;

    while (end >= start)
    {
        middle = (end + start) / 2;

        if (item == array[middle])
        {
            found = middle;
            break;
        }
        else if (item < array[middle])
            end = middle - 1;
        else
            start = middle + 1;
    };

    return found;
}
```