COMP10002 Foundations of Algorithms

Workshop Week5

Wenbin Cao

August 29, 2019

GitHub Repo: https://github.com/AlanChaw/COMP10002-FoA

Recap

Arrays

- Array Declaration and Using
- Reading into an array
- Arrays and Functions

Insertion Sort

Measuring Performance

- Linear Search
- Binary Search

Array Declaration and Using

```c
int main(int argc, char *argv[]) {
    int A[5];
    int i;
    // array subscripts start at zero
    for (i = 0; i < 5; i++) {
        A[i] = i;
    }
    for (i = 0; i < 5; i++) {
        printf("A[%d] = %d\n", i, A[i]);
    }
    return 0;
}
```

Output:

```
A[0] = 0
A[1] = 1
A[2] = 2
A[3] = 3
A[4] = 4
```

Reading into an array

- Array subscripts start at zero

- Bounds checking must be done by the programmer

Reading into an array

```c
int main(int argc, char *argv[]) {
    printf("Input some numbers: ");
    int N = 10, i, num;
    int A[N];
    for (i = 0; i < N; i++){  // initialize array
        A[i] = 0;
    }

    i = 0;
    while (scanf("%d", &num) == 1) { // get the inputs
        A[i] = num;
        i++;
    }

    for (i = 0; i < N; i++){  // print array elements
        printf("%d ", A[i]);
    }

    return 0;
}
```

Reading into an array

Output:

```
Input some numbers: 3 4 5 6 7
3 4 5 6 7 0 0 0 0 0
```

Insertion Sort

Insertion sort by visualgo.net

https://visualgo.net/bn/sorting

C Code:

```c
void sort_int_array(int A[], int n) {
    int i, j;
    /* assume that A[0] to A[n-1] have valid values */
    for (i=1; i<n; i++) {
        /* swap A[i] left into correct position */
        for (j=i-1; j>=0 && A[j+1]<A[j]; j--) {
            /* not there yet */
            int_swap(&A[j], &A[j+1]);
        }
    }
    /* and that's all there is to it! */
}
```

Discussion

7.3 Modify the program of Figure 7.3 on page 104 so that after the array has been sorted only the distinct values are retained in the array (with variable `n` suitably reduced):

```
mac: ./distinct
Enter as many as 1000 values, ^D to end
1 8 15 3 17 12 4 8 4
^D
9 values read into array
Before:    1   8  15   3  17  12   4   8   4
After :    1   3   4   8  12  15  17
```

Exercise 7.3 - Solution

```
int i, newn;

/* first, sort the array */
sort_int_array(A, n);
/* always accept first item into the output array */
newn = 1;
/* now do the reduction, starting at the second item, and
 bypassing items that are same as the last one already placed
 at the tail of the front part of the reduced A */
for (i=1; i<n; i++) {
    if (A[i] != A[newn-1]) {
        A[newn] = A[i];
        newn++;
    }
}
```

Discussion

7.6 An alternative sorting algorithm that you might be familiar with goes like this: scan the array to determine the location of the largest element, and then swap it into the last position. Then repeat the process, concentrating at each stage on the elements that have not yet been swapped into their final position. This strategy is called *selection sort*.

Write a function `void selection_sort(int A[], int n)` that orders the `n` elements in array `A`.

(*For a challenge*) Write your function using recursion rather than iteration.

Arrays and functions

- The array name is a *pointer constant* whose value is the address of the first variable in the array.

```
int main(int argc, char *argv[]) {
    int A[5];
    int *p;
    p = A;
    printf("address value of A is: %p\n", A);
    printf("address value of A[0] is: %p\n", &A[0]);
    printf("address value of p is: %p\n", p);

    return 0;
}
```

Output:

```
address value of A is: 0x7ffeefbff5a0
address value of A[0] is: 0x7ffeefbff5a0
address value of p is: 0x7ffeefbff5a0
```

Arrays and functions

- In the function header, the array can be declared as a pointer in the function header, or as an undimensioned array.

```
int read_int_array(int A[], int n);
void selection_sort(int A[], int n);
```

- If the array elements are changed in the function, the outside original array will also be changed.

- The length of the array should always be passed to the function together with the array.

Discussion

7.7 Write a function that takes as arguments an integer array `A` and an integer `n` that indicates how many elements of `A` may be accessed, and returns the value of the integer in `A` that appears most frequently. If there is more than one value in `A` of that maximum frequency, then the smallest such value should be returned. The array `A` may not be modified.

Hands on exercise

Exercise 7.8, 7.9, 7.10, 7.11, as many as you can.

Any questions for Chapters 1-6 ?

7.8 Write a function that takes as arguments an integer array `A`, an integer `n` that indicates how many elements of `A` may be accessed, and an integer `k`; and returns the value of the `k`'th smallest integer in `A`. That is, the value returned should be the one that would move into `A[k]` if array `A` were to be sorted. The array `A` may not be modified. Be sure that you handle duplicates correctly.

7.9 One way of quantifying how close an array is to being sorted is the number of ascending *runs*. For example, in the array $\{10, 13, 16, 18, 15, 22, 21\}$ there are three runs present, starting at 10, at 15, and at 21. Write a function that returns the number of runs present in the integer array `A` of size `n`.

7.10 Another way of quantifying sortedness is to count the number of *inversions* – pairs of elements that are out of order. For example, on the sequence in the previous question, there are three inversions: two caused by 15, and one caused by 21. Write a function that returns the number of inversions present in the integer array `A` of size `n`.