CSP2348/CSP5243 Data Structures

Solutions to Tutorial 04: - Arrays Sorting Algorithms and Analysis

---

Tasks:

Complete the following.

Task 1: Consider the problem of reading a file of (unsorted) values into an array, where the array must be sorted. There are *n* values in the file.

(1) Write an algorithm to read all of the unsorted values into the array, and then sort the array using the selection sort algorithm given in Algorithm 3.27. What is the time efficiency of your algorithm?

(2) Write an algorithm to read each value in turn, and insert it into a sorted array (initially empty). What is the time efficiency of your algorithm? How does this compare with your answer to part (1)?

Answers:

(1) To read values from the unsorted file *f* into a sorted array *a*[0…] :

```
1. Set m to 0.
2. While not at end of file f, repeat:
      2.1. Read value val from f.
      2.2. Store val into a[m].
      2.3. Increment m.
3. Sort a[0…m–1] using Selection sort algorithm.
4. Terminate.
```

Analysis: Step 2 performs 0 comparisons. Step 3 uses selection sort to sort the array a[ ]. It performs about $n^2/2$ comparisons. Therefore this algorithm performs about $n^2/2$ comparisons.

(2) To read values from the unsorted file *f* into a sorted array *a*[0…] :

```
1. Set m to 0.
2. While not at end of file f, repeat:
      2.1. Read value val from f.
      2.2. Insert val in the sorted array a[0…m–1].
      2.3. Increment m.
3. Terminate.
```

Analysis: Step 2.2 would use the array insertion algorithm. This performs about $m/2$ comparisons. Since m ranges from 0 to $n–1$, the total number of comparisons is $0 + 1/2 + ... + (n–1)/2 = n(n–1)/4 \approx n^2/4$.

Both methods have time complexity $O(n^2)$.

Task 2: You are given two unsorted arrays of values. You are required to obtain a sorted array containing all these values. Suggest two different ways of achieving this. Compare their time efficiency. (Note: Assume that suitable merging and sorting algorithms are already available.)

Answers: Let $n1 = right1 - left1 + 1$, $n2 = right2 - left2 + 1$, and $n = n1 + n2$.

First way (version 1): Copy all values from arrays *a1*[*left1*…*right1*] and *a2*[*left2*…*right2*] into an array *a3*[0…n-1], and then sort the array *a3*[*0…n-1*] using an available array sorting algorithm.

```
1. Copy a1[left1…right1] to a3[0…n1-1];
2. Copy a2[left2…right2] to a3[n1…n-1];
3. Sort a3[0…n-1];
4. Terminate.
```

Analysis: Step 1 and 2 performs 0 comparisons. If step 3 uses (say) selection sort, it performs about $n^2/2$ comparisons. The total number of comparisons is therefore about $n^2/2 = (n1 + n2)^2/2 = n1^2/2 + n2^2/2 + n1n2$.

Second way (version 2): Sort arrays *a1*[*left1*…*right1*] and *a2*[*left2*…*right2*] respectively, and then merge the two sorted arrays into sorted array *a3*[*1*…n-1] using array merging algorithm:

```
1. Sort a1[left1…right1];
2. Sort a2[left2…right2];
3. Merge a1[left1…right1] and a2[left2…right2] into a3[1…n-1];
4. Terminate.
```

Analysis: If steps 1 and 2 use selection sorting algorithm, they perform about $n1^2/2$ and $n2^2/2$ comparisons, respectively. Step 3 performs about $n = n1 + n2$ comparisons. The total number of comparisons is therefore about $n1^2/2 + n2^2/2 + n1 + n2$.

For all but small values of $n1$ and $n2$, $n1 + n2 < n1n2$. Therefore the second way is faster than the first.

Task 3: Test the Java selection and insertion sorting programs given in WS0401

(Download the Java code from Blackboard)

a. Run this program to observe the sorting process using pre-coded data;
b. Explain the executed results according to the principles of the selection and insertion sorting algorithms;
c. Modified the code to test different ***data sets***.

Run the Java code by yourself.

Task 4: Test Java merge and quick sort programs given in WS0402

(Download the Java code from Blackboard).

a. Run this program to observe the sorting process using pre-coded data;
b. Explain the executed results according to the principles of the merge and quick sort algorithms;
c. Modified the code to test different ***data sets***.

Run the Java code by yourself.

Task 5: Test the Vector class using WS0403 (Download the Java code from Blackboard).

a. Note how to construct vector objects;

```
private static Vector v = new Vector();
private static Vector w = new Vector();
```

b. Observe the operation of some vector methods by analyzing the executed results.

Executed results:

```
v = []
v = [Perth, Sydney, Melbourne, Brisbane, Adelaide]
w = [Perth, Sydney, Melbourne, Brisbane, Adelaide]
w.equals(v) = true
v = [Perth, Sydney, Melbourne, Canberra, Adelaide]
w = [Perth, Sydney, Melbourne, Brisbane, Adelaide]
w.equals(v) = false
v = [Perth, Sydney, Melbourne, Hobart, Canberra, Adelaide]
w = [Perth, Sydney, Melbourne, Brisbane, Adelaide]
w.equals(v) = false
w = [Perth, Melbourne, Brisbane]
v = [Perth, Sydney, Melbourne, Hobart, Canberra, Perth,
Melbourne, Brisbane, Adelaide]
v.indexOf("Perth") = 0
v.indexOf("Perth",2) = 5
v.indexOf("Canberra") = 4
```