University of Glasgow | School of Computing Science

CSC1103 Tutorial 6 : One Dimensional Array

1. Quick Sort Algorithm

1. Problem definition:

Write a C program to take in 10,000 uniform distributed random numbers and generate the quick sort of these 10,000 in ascending order using recursive function and array.

2. Problem Analysis

The sorting algorithm is a recursive function that mainly do two functional roles.

1. Partition the data set into two halves with a pivot as the comparator.
    i. Choose a pivot (normally choose the first element of the data set to be compared). Set the position index to 0.
    ii. One by one compare the pivot with the succeeding data till the last data in the set. Those data that is smaller than pivot, the position index will be incremented by 1, and these data will swap with the data at the location indicated by the position index. Data that is larger than pivot will stay put and position index remain the same number.
    iii. Swap the pivot with data at the location indicated by the position index so that two partition has been formed with left partition having data smaller than pivot and right partition having data larger than pivot
2. Sub partition the left partition into two halves and recursively applied the quicksort algorithm gain to the left partition till left one element to be partitioned and sorted. Similarly applied to the right partition.

Input variable

i. The 10,000 uniformly distributed random data x (int x[ ])

Process variable

i. first position of the data in the set to be sorted, $first$ (int $first$)
ii. last position of the data in the set to be sorted, $last$ (int $last$)
iii. the position index of the pivot element (int $pivot$)

Output variable

i. The 10,000 sorted uniformly distributed random data x (int x[ ])

3. Algorithm

The program is divided into following functions

a) **main ()** :
i. to obtain 10,000 uniformly distributed random number and store in the data array $x[]$. Use the inbuilt library **rand()** to generate number
ii. call the function **quick_sort( )** to sort the 10,000 random data
iii. print out the sorting result of the 10,000 random data

b) **quick_sort** ():
i. recursively call the function **partition ()** to sort the data into left and right partition and get new pivot position index (pivot) for next sort
ii. recursively call the function **quick_sort ()** to sort left partition with respect to the pivot position
iii. recursively call the function **quick_sort ()** to sort right partition with respect to the pivot position

c) **partition ():**
i. compare the inputted pivot value with the succeeding data value. If the data value is smaller than pivot value, increase the position index (pivot) by 1 and swap this data with the data at location pointed by position index (pivot)
ii. Once pivot has compared till the last data in the set, swap the piviot value with the data at location pointed by position index (pivot)
iii. return position index (pivot) value back to **quick_sort ()**

Algorithm for **main ()**

1. Set num_items =10,000
2. For $i = 0\ to\ num\_items - 1$ do the following
   2.1 $x\ [i] = \boldsymbol{rand}\ ()$
   2.2 Print the value of $x\ [i]$
3. Call **quick_sort (**$x\ , 0, num\_items - 1$**)**
4. For $i = 0\ to\ num\_items - 1$ do the following
   4.1 Print the sorted value of $x\ [i]$

Algorithm for **quick_sort(**$x\ , first, last$**)**

1. If $(first < last)$
   1.1. $pivot$ = **partition(**$x\ , first, last$**)**
   1.2. call **quick_sort (**$x\ , first, pivot - 1$**)**
   1.3. call **quick_sort (**$x\ , pivot + 1, last$**)**

Algorithm for **partition** (x , $first, last$)

1. Set $pivot = first$
2. Set $pivot\_value = x[first]$
3. For $i = first$ *to* $last$ do the following
   - 3.1 If $(x[i] < pivot\_vlaue)$
     - 3.1.1 Set $pivot = pivot + 1$
     - 3.1.2 If $(i\ != pivot)$
       - 3.1.2.1 Set $temp = x[pivot]$
       - 3.1.2.2 Set $x[pivot] = x[i]$
       - 3.1.2.3 Set $x[i] = temp$
4. Set $temp = x[pivot]$
5. Set $x[pivot] = x[first]$
6. Set $x[first] = temp$
7. return $pivot$

Pseudocode

```
BEGIN
    num_items ← 10,000
    FOR i = 0 to num_items − 1 do
        x[i] ← rand()
        PRINT x[i]
    END FOR
    quick_sort(x, 0, num_items − 1)
    FOR i = 0 to num_items − 1 do
        PRINT x[i]
    END FOR
END

FUNCTION quick_sort(x, first, last)
    IF (first < last)
        pivot ← partition(x, first, last)
        quick_sort(x, first, pivot − 1)
        quick_sort(x, pivot + 1, last)
    ENDIF
ENDFUNCTION
```

```
FUNCTION partition(x, first, last)
        pivot ← first
        pivot_value ← x[first]
        FOR i = first to last do
                IF (x[i] < pivot_value)
                        pivot ← pivot + 1
                        IF (i! = pivot)
                                temp ← x[pivot]
                                x[pivot] ← x[i]
                                x[i] ← temp
                        ENDIF
                ENDIF
        END FOR
        temp ← x[pivot]
        x[pivot] ← x[first]
        x[first] ← temp
        return pivot
ENDFUNCTION
```

2. VECTOR CROSS PRODUCT

1. Problem definition:

Write a C program to take in the user input of two 3D vectors **A** and **B** and store in the respective 3 vector components in array parameters $a[]\ and\ b[]$. Compute the cross product of the two vectors **A** and **B** and store in $c[]$ and print the result.

2. Problem Analysis

The two 3D vector **A** and **B** are given as

$$\mathbf{A} = \mathrm{A}_x\boldsymbol{i} + \mathrm{A}_y\boldsymbol{j} + \mathrm{A}_z\boldsymbol{k}$$
$$\mathbf{B} = \mathrm{B}_x\boldsymbol{i} + \mathrm{B}_y\boldsymbol{j} + \mathrm{B}_z\boldsymbol{k}$$

The cross product of two vector $\mathbf{A} \times \mathbf{B} = \mathbf{C}$

$$\begin{aligned}\mathbf{C} &= (\mathrm{A}_y\mathrm{B}_z - \mathrm{A}_z\mathrm{B}_y)\boldsymbol{i} + (\mathrm{A}_z\mathrm{B}_x - \mathrm{A}_x\mathrm{B}_z)\boldsymbol{j} + (\mathrm{A}_x\mathrm{B}_y - \mathrm{A}_y\mathrm{B}_x)\boldsymbol{k} \\ &= \mathrm{C}_x\boldsymbol{i} + \mathrm{C}_y\boldsymbol{j} + \mathrm{C}_z\boldsymbol{k}\end{aligned}$$

where

$$\mathrm{C}_x = \mathrm{A}_y\mathrm{B}_z - \mathrm{A}_z\mathrm{B}_y$$
$$\mathrm{C}_y = \mathrm{A}_z\mathrm{B}_x - \mathrm{A}_x\mathrm{B}_z$$
$$\mathrm{C}_z = \mathrm{A}_x\mathrm{B}_y - \mathrm{A}_y\mathrm{B}_x$$

Using the array $a[], b[] and\ c[]$, we can define

$$\mathrm{A}_x = a[0], \mathrm{A}_y = a[1], \mathrm{A}_z = a[2]$$
$$\mathrm{B}_x = b[0], \mathrm{B}_y = b[1], \mathrm{B}_z = b[2]$$
$$\mathrm{C}_x = c[0], \mathrm{C}_y = c[1], \mathrm{C}_z = c[2]$$

Input variable

i. The vector A component $a[]$ (float $a[3]$)
ii. The vector B component $b[]$ (float $b[3]$)

Output variable

i. The cross product vector C component $c[]$ (float $c[3]$)

3. Algorithm

The program is divided into following 2 functions

a) **main ()** :
i. prompt for user input of the 3 vector components of vector **A** and **B**
ii. call the function **crossproduct ()** to compute the cross product of vector **A** and **B** which is vector **C**.
iii. print out the vector components of vector **C**

b) **crossproduct** ():
i. compute the 3 components of vector C which is cross product of vector **A** and **B**

Algorithm for **main ()**

1. For $i = 0\ to\ 2$ do the following
   1.1 Read $a\ [i]$
2. For $i = 0\ to\ 2$ do the following
   2.1 Read $b\ [i]$
3. Call **crossproduct (**$a\ , b, c$**)**
4. For $i = 0\ to\ 2$ do the following
   4.1 print $c\ [i]$

Algorithm for **crossproduct (**$a\ , b, c$**)**

1. Set $c[0] = a[1] * b[2] - a[2] * b[1]$
2. Set $c[1] = a[2] * b[0] - a[0] * b[2]$
3. Set $c[2] = a[0] * b[1] - a[1] * b[0]$

Pseudocode

```
BEGIN
    FOR i = 0 to 2 do
        READ a[i]
    ENDFOR
    FOR i = 0 to 2 do
        READ b[i]
    END FOR
    crossproduct(a, b, c)
    FOR i = 0 to 2 do
        PRINT c[i]
    END FOR
END

FUNCTION crossproduct(a, b, c)
    c[0] ← a[1] * b[2] − a[2] * b[1]
    c[1] ← a[2] * b[0] − a[0] * b[2]
    c[2] ← a[0] * b[1] − a[1] * b[0]
ENDFUNCTION
```