Divide and Conquer Pattern applied to Sorting Equals the MergeSort Algorithm

Project 3 Management and Data Structures

CSI 310 Lecture 17 (Spr 06)
Only true "logically"... this data is actually stored in the called activation.

ACTIVATION is destroyed, and its Record gets recycled.

Really: When an ACTIVATION executes the return statement, that

Definition of Activation Record: The data structure that holds (1) An

will control what THAT activation does.

Really: A new Function Activation is created, and the function's body

WRONG: Control "jumps" or "goes to" the function's body.

WHAT HAPPENS when the computer executes a FUNCTION CALL?
3. A stack is a sequence for which insertion and deletion are only done at ONE END (called the top.).

2. Activation records are stored in a STACK (like a pile of bills).

1. Last week’s Lab (4): OBSERVE the activation records for runs of the recursive function WRITE-VECTOR() from DSO chapter 9.
not here.

Note 1: mergesort is in progress.

Note 2: 2 dynamic *char[] used.

Note 3: argc, argv, etc. on Handout.

Exit
Start
We will then illustrate the (recursive) Mergesort half.
Sample list of items to sort:

Cat
Aardvark
Bat
Ape
Caterpillar
Dog
Ant
Zebra
(b) Many of fixed capacity for temporary storage of sequences.

(a) One of dynamic capacity for inputting an unpredictable number of

3. Dynamic arrays for

so, no copying of the characters in the strings is ever done!

pointer/rel. values.

array is to store POINTERS (Java references) in the array, and move

array. The fastest way to change the positions of STRINGS sequenced with an

MATTERS and multiple occurrences are OK with an array.

I. Implement the ADT sequence (element collection where ORDER

Main data structure ideas for sorting strings using an array.

University at Albany Computer Science Dept.
“Merging” is called "Merging"

3. Combine the two sorted groups into one large sorted list. This combining

independently.

2. Sort each of these smaller groups (by recursive calls). That means

size.

1. Divide the elements to be sorted into two groups of equal (or almost equal

paradigm or pattern is applied to the problem of sorting a sequence:

MergeSort is the algorithm that is invented when the divide-and-conquer

University at Albany Computer Science Dept.
First Recursive Activation
Recursive Solution is:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Input</th>
<th>ANSWER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
## Answer

See the Returned

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>20</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>12</td>
<td>25</td>
<td>31</td>
<td>5</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>9</td>
<td>12</td>
<td>25</td>
<td>31</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>12</td>
<td>31</td>
<td>25</td>
<td>5</td>
</tr>
</tbody>
</table>

University at Albany Computer Science Dept.
Again on 2nd half.

**NEXT, Recurse**

**Input**

<table>
<thead>
<tr>
<th>9</th>
<th>12</th>
<th>25</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

**Problem**

<table>
<thead>
<tr>
<th>?</th>
<th>?</th>
<th>?</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

**Answer**

<table>
<thead>
<tr>
<th>?</th>
<th>?</th>
<th>?</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Recursive Solution is:
## Answer by 2nd Recursion

### Problem 1

<table>
<thead>
<tr>
<th>20</th>
<th>6</th>
<th>8</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>20</td>
<td>3</td>
</tr>
</tbody>
</table>

### Problem 2

<table>
<thead>
<tr>
<th>9</th>
<th>12</th>
<th>25</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>9</td>
<td>12</td>
<td>31</td>
</tr>
</tbody>
</table>

---

### Problem 3

<table>
<thead>
<tr>
<th>6</th>
<th>3</th>
<th>20</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>20</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

---

### Problem 4

<table>
<thead>
<tr>
<th>9</th>
<th>12</th>
<th>31</th>
<th>25</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>31</td>
<td>25</td>
<td>5</td>
<td>9</td>
</tr>
</tbody>
</table>
NO More Recursion!!! Get answer by Merging:
NOTE: If there is insufficient dynamic memory, then exception bad alloc is thrown.

// Elements rearranged in sorted order.

post: data [0 .. (n1+n2-1)] contain the original

data [n1 .. (n1 + n2 - 1)] are also already sorted
and

data [n1-1 .. (n1-1)] are already sorted

(critical)

// Pre: data points to subarray with at least (n1 + n2) elements.

void merge (int data [], size_t n1, size_t n2)
void mergeSort(int data[], size_t n);

NOTE: If there is insufficient dynamic memory, then
exception bad_alloc is thrown.

// Pre: data==adder. of an array of length at least n.
// Post: The elements of data[0] ... (n-1) contain its
// original elements rearranged in sorted order. I.E.
// data[0] ... [n-1] => data[1] ... => ... => data[n-1].

// exception bad_alloc is thrown.
ORDER in which Activations are Created
3. Code separateSorter in with pre and post conditions. Implement functionality.

The print operation lets you and the TA TEST your input phase

```c
arc, char *array []

command line argument array (pointed to by and are of main)nt

command phase: dispatches the operations listed ALREADY in the

THE capacity DOUBLE!!
```

3. The SorterBench dynamic partially filled array MUST increase its own
capacity when necessary as explained in ch. 4 of DSO. Great Idea: MAKE

```c
SorterBench::append(char *)
```

2. Input phase Line reader and sorter. (Sorter calls

```c
of char pointers (or string pointers).
```

One object of class SorterBench will have one dynamical partially filled array

SorterBench.

Create and use header file SorterBench.h to declare the class

I. Input phase Line reader and skeleton main() . Details Given in the Handout.

Project 3 Management
May be write a test driver for merge.

8. Implementation, test and debug FIRST merge and SECOND merge sort.

- to arrays of int.

functions of Ch. 13, except they apply to arrays of char * or string * not

7. Write interfaces and pre/post conditions for the merge sort () and merge ()

6. Follow instructions to implement sort algorithm timing.

```c

#include <string>

void selectsortsort (char *data[], int n) {
    option #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>

    #include <string>
```c
{ return
    cout << A; // print the sorted string.
}

Now, A[0...nch-I] is SORTED.

Now, A[I] has the smallest char from A[1...nch-I]

/* [I] A[0...nch] A[0...nch] A

for (int j = I; j < nch; ++j)
    if (A[I] < A[MinIndex])
        MinIndex = I;

for (int j = 0; j < nch-1; ++j)
    if (GetNext(A, nch) == Char(A[I]))
        return 1;  // if the the array A
            const int ASize = 100;  // the array A.
    }

main()

using namespace std;

#include <string>
#include <iostream>
#include <algorithm>

// Selection sort demo: processes the array A.
```
We now illustrate what half your Project 3 work must do.

It uses the Selection Sort algorithm for sorting.
This program manipulates names as if they were numbers.