More sorting

2/5/16
Announcements

• Quiz today: Writing a method for RecursiveBag (just the EmptyNode and DataNode methods)

• Makeup quizzes available after class, Monday, and Tuesday.

• Reading:
  – For Monday: Chapter 10
  – For Friday: Chapter 12 and Java Interlude 5
Recall: Sorting algorithms from last time

• Selection sort
  – Grow sorted part of array by finding smallest value in the remaining

• Insertion sort
  – Grow sorted part of the array by inserting new values into the proper place

• Bubble sort
  – Repeatedly find out-of-order pairs and swap them
Which of the sorting algorithms could cause the execution shown below?

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>1</th>
<th>4</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

A. Insertion sort  
B. Bubble sort  
C. Selection sort  
D. None of these  
E. More than one of these
Which of the sorting algorithms could cause the execution shown below?

A. Insertion sort
B. Bubble sort
C. Selection sort
D. None of these
E. More than one of these

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Which of the sorting algorithms could cause the execution shown below?

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

A. Insertion sort  
B. Bubble sort  
C. Selection sort  
D. None of these  
E. More than one of these
Which of the sorting algorithms could cause the execution shown below?

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

A. Insertion sort  
B. Bubble sort  
C. Selection sort  
D. None of these  
E. More than one of these (insertion & bubble)
Can we do better than $O(n^2)$ time?
Merge sort

- Split array into two equal-sized pieces, recursively sort each half, and merge them back

\[
\text{Mergesort}(A[1..n]) \{
\text{if}(n > 1) \{
\quad m = (n+1)/2;
\quad \text{copy } 1^{\text{st}} \text{ m values of } A \text{ into array } L \text{ and rest into array } R;
\quad \text{Mergesort}(L);
\quad \text{Mergesort}(R);
\quad \text{Merge}(L, R, A);
\}
\}
\]
Merge(A[1..n₁], B[1..n₂], R[1..(n₁+n₂)]) {
    //merge sorted arrays A and B into R
    int i = 1, j = 1;       //i is position in A, j is position in B
    for(int k = 1; k <= (n₁+n₂); k++) {  //k is position in R
        if(i ≤ n₁ and (j > n₂ or A[i] ≤ B[j])) {
            R[k] = A[i];
            i++;
        } else {
            R[k] = B[j];
            j++;
        }
    }
}
Merge(A[1..n₁], B[1..n₂], R[1..(n₁+n₂)]) {
  //merge sorted arrays A and B into R
  int i = 1, j = 1;  //i is position in A, j is position in B
  for(int k = 1; k <= (n₁+n₂); k++) { //k is position in R
    if(i ≤ n₁ and (j > n₂ or A[i] ≤ B[j])) {
      R[k] = A[i];
      i++;
    } else {
      R[k] = B[j];
      j++;
    }
  }
}

What most accurately characterizes the running time of Merge? (n= n₁ + n₂)
A. O(1)  
B. O(n) 
C. O(n²)  
D. None of the above
Merge(A[1..n₁], B[1..n₂], R[1..(n₁+n₂)]) { //merge sorted arrays A and B into R
    int i = 1, j = 1; //i is position in A, j is position in B
    for(int k = 1; k <= (n₁+n₂); k++) { //k is position in R
        if(i <= n₁ and (j > n₂ or A[i] <= B[j])) {
            R[k] = A[i];
            i++;
        } else {
            R[k] = B[j];
            j++;
        }
    }
}

What most accurately characterizes the running time of Merge? (n= n₁ + n₂)
A. O(1)
B. O(n)
C. O(n²)
D. None of the above
Running time of mergesort

Task of sorting n numbers
Running time of mergesort

- Task of sorting \( n \) numbers
  - Task of sorting \( \frac{n}{2} \) numbers
  - Task of sorting \( \frac{n}{2} \) numbers
Running time of mergesort

- Task of sorting $n$ numbers
  - Task of sorting $n/2$ numbers
    - Task of sorting $n/4$ numbers
    - Task of sorting $n/4$ numbers
  - Task of sorting $n/2$ numbers
    - Task of sorting $n/4$ numbers
    - Task of sorting $n/4$ numbers
    - Task of sorting $n/4$ numbers

... and so on
Running time of mergesort

Task of sorting \( n \) numbers

- Task of sorting \( \frac{n}{2} \) numbers
  - Task of sorting \( \frac{n}{4} \) numbers
  - Task of sorting \( \frac{n}{4} \) numbers
  - Task of sorting \( \frac{n}{4} \) numbers

- Task of sorting \( \frac{n}{2} \) numbers
  - Task of sorting \( \frac{n}{4} \) numbers
  - Task of sorting \( \frac{n}{4} \) numbers
  - Task of sorting \( \frac{n}{4} \) numbers

\( \ldots \) and so on
Running time of mergesort

- Task of sorting $n$ numbers
  - Task of sorting $n/2$ numbers
    - Task of sorting $n/4$ numbers
    - Task of sorting $n/4$ numbers
    - Task of sorting $n/4$ numbers
    - Task of sorting $n/4$ numbers
  - Task of sorting $n/2$ numbers
  - Task of sorting $n/2$ numbers

$c \times n$ non-recursive work for the merge
$c \times (n/2)$ non-recursive work for the merge

... and so on
Logarithms!
Logarithms

\( \log_b x = \text{“log base } b \text{ of } x” \)

= power of \( b \) that gives \( x \)

= number of times you can divide \( x \) by \( b \) before getting 1
Logarithms

\[ \log_b x = \text{“log base } b \text{ of } x\text{”} \]

= power of \( b \) that gives \( x \)

= number of times you can divide \( x \) by \( b \)

before getting 1

\[ 2^0 = 1 \text{ so } \log_2 1 \text{ is 0} \]

\[ 2^1 = 2 \text{ so } \log_2 2 \text{ is 1} \]

\[ 2^2 = 4 \text{ so } \log_2 4 \text{ is 2} \text{ (and } \log_2 3 \text{ is between 1 & 2)} \]
What is $\log_2 32$?

A. 3
B. 4
C. 5
D. 6
E. None of the above
What is $\log_2 32$?

A. 3  
B. 4  
C. 5  
D. 6  
E. None of the above
What is $\log_2 100$?

A. Between 6 and 7
B. Between 7 and 8
C. Between 8 and 9
D. Between 9 and 10
E. None of the above
What is $\log_2 100$?

A. Between 6 and 7
B. Between 7 and 8
C. Between 8 and 9
D. Between 9 and 10
E. None of the above