Lecture 19 Part 2 (Sorting) CSI333

Slide 1

Sorting:

<table>
<thead>
<tr>
<th>21</th>
<th>16</th>
<th>42</th>
<th>38</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>16</td>
<td>16</td>
<td>21</td>
<td>38</td>
<td>42</td>
</tr>
</tbody>
</table>

NOT SUFFICIENT for kwic333 and many other applications!

A

<table>
<thead>
<tr>
<th>Record</th>
<th>Record</th>
<th>Record</th>
<th>Record</th>
<th>Record</th>
<th>Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Key</td>
<td>Key</td>
<td>Key</td>
<td>Key</td>
<td>Key</td>
</tr>
<tr>
<td>Other Data</td>
<td>Other Data</td>
<td>Other Data</td>
<td>Other Data</td>
<td>Other Data</td>
<td>Other Data</td>
</tr>
</tbody>
</table>

Sorting Problem: Rearrange Records so:

Slide 2

Separate Problems:

1. Given two records, extract their keys and determine which is earlier, or if they are ordered equally.

2. Given a key comparison function, use its return values to rearrange the records.

(Stable Sort: A sorting method where two records with equally ordered keys are output in their original input order. Not required for the current project.)
Inefficient \(O(N^2)\) time sorting algorithm:

```c
for( i = 0; i <= N-2; i++ )
    for( j = i + 1; j <= N-1; j++ )
        if( A[i].Key > A[j].Key )
            SWAP( A[i], A[j] );
// Invariant: A[i].Key is minimal
// among the Keys in A[i...j].
```

Good for nothing but very small or rarely solved problems (relatively: STILL USEFUL!)

Sorting is much easier if the input consists of two record sequences, A and B, each already sorted.

They can be “merged” to form a sorted output sequence.

Each record is copied only once, and the number of key comparisons is at most 
#records – 1.

merging loop:
```c
if( A is empty )
    if( B is empty ) {break out of loop;}
    else {remove and output next B record}
else    //Here, A is non-empty.
    if( B is empty )
        {remove and output next A record}
    else
        {compare next A record, next B record;
         remove and output a minimal one }
```
Merge Algorithm Implementation Challenges:

1. Programmer must design how to implement sequences, remove records and test when A or B become empty.
2. We don’t want to copy records except when outputting them.
3. Merging arrays requires a separate output array since we cannot predict when A or B will be emptied.

The natural “divide and conquer” idea leads to one of the best sorting algorithms: Merge Sort.

1. If the input sequence has length 1 (or is “small”), sort it a simple way and return. Otherwise,
2. Divide the input into A and B of sizes inputLen/2, inputLen-inputLen/2.
3. Sort A; Sort B (recursively).
4. Merge sorted A with sorted B.

Two data structure choices for sequences:

**Linked List** Disadvantages:
Time wasted to scan to find center.
Space for links.

**Array** Disadvantage:
Another array is needed to hold merge’s output.

Let’s use arrays.
Slide 9

Iterative View of Merge Sort:

1. (Each single record is already sorted.)
2. Sort successive groups of 2 records.
3. Sort successive groups of 4 records.
4. Sort successive groups of 8 records.
5. etc... until there is just one sorted group.

This does the same calculation as the recursive version but in a different order.

Slide 10

Let $S$ be the size of the previously sorted groups.

Next iteration: For $I=0$, $2S$, $4S$, etc.,

$$\text{Merge } A[I \ldots I+S-1] \text{ with } A[I+S \ldots I+2S-1] \text{ to get sorted groups of size } 2S$$

How To Study This

What does it mean when $S = 1$?


Slide 11

When the iteration with $S=1$ is finished, each successive pair of $A$ entries are sorted: $A[0..1]$ is sorted, $A[2..3]$, etc.

What happens during the iteration with $S = 2$?

Merge $A[0..1]$, $A[2..3]$ ($I=0$)

Now entries $0..3$ are sorted.


Now entries $4..7$ are sorted.

e tc.
Slide 12

Challenges:

1. Where can merge's output be written?

2. What if I+S-1 or I+2*S-1 is out of A's subscript bound? This happens when a group is not full size.

Slide 13

Answers:

1. Use a "scratch" array Z with length the same as A's to put the output of each run of Merge. Then, copy the records from Z back to A. (Use "parallel" subscripts.)

   (If you want to cleverly avoid the copying, design the program to switch the roles of A and Z with each new S value.)

2. (See next Slide).

Slide 14

merging loop:
if( A is empty:CHECK SUBSCRIPT)
  {if( B is empty:CHECK SUBSCRIPT) {break;}
   else {remove and output next B record}}
else //Here, A is non-empty.
  {if( B is empty:CHECK SUBSCRIPT)
   {remove and output next A record}
   else
   {compare next A record, next B record;
    remove and output a minimal one }
  } NOTE:to "remove," increment an index.
Slide 15

Merge Sort Running Time Analysis

$L = \text{Number of Passes}$

$(S = 1 \text{ before first pass, } S = 2^1 \text{ after 1st pass})$

$S = 2^L \text{ after pass } L. \text{ Sort is finished when } 2^{L-1} < N \leq 2^L$

Conclude: $L = \text{ceil}(\log_2(N))$

Number of copyings of records is

$\text{NCPY} = N \times L = N \times \log_2(N)$

$\log_2(1024) = 10; \log_2(1\text{Meg}) = 20$

Slide 16

Selection Sort Running Time Analysis

Number of compare and swaps = NCS

$= (N - 1) + (N - 2) + \cdots + 2 + 1$

$= \frac{N(N - 1)}{2}$

Slide 17

<table>
<thead>
<tr>
<th>$N$</th>
<th>Merge Sort’s NCPY</th>
<th>Sel. Sort’s NCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>$N \times \log_2(N)$</td>
<td>$N(N - 1)/2$</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td>120</td>
</tr>
<tr>
<td>1024</td>
<td>10,240</td>
<td>523,776</td>
</tr>
<tr>
<td>8192</td>
<td>106,496</td>
<td>33,550,336</td>
</tr>
<tr>
<td>1Meg</td>
<td>10,485,760</td>
<td>549,755,200,000</td>
</tr>
</tbody>
</table>