CS341: Algorithms
Lecture 01: Merge Sort

Yao-Liang Yu

School of Computer Science
University of Waterloo

January 5, 2017
Plan for Today

1. Recap
2. More Examples
3. Merge Sort
Plan

1 Recap

2 More Examples

3 Merge Sort
Course Information

- **Yao, T 10:00 - 11:00, DC 3206** or by email appointment
  
yaoliang.yu@uwaterloo.ca

- **Piazza** (will be invited)
  
piazza.com/uwaterloo.ca/winter2017/cs341

- **Learn**
  
  https://learn.uwaterloo.ca

Assignments submitted here using dropbox.
## Coursework

<table>
<thead>
<tr>
<th></th>
<th>out date</th>
<th>due date</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Jan 6</td>
<td>Jan 20</td>
<td>5%</td>
</tr>
<tr>
<td>A2</td>
<td>Jan 20</td>
<td>Feb 3</td>
<td>5%</td>
</tr>
<tr>
<td>A3</td>
<td>Feb 3</td>
<td>Feb 17</td>
<td>5%</td>
</tr>
<tr>
<td>A4</td>
<td>Feb 17</td>
<td>Mar 10</td>
<td>5%</td>
</tr>
<tr>
<td>A5</td>
<td>Mar 10</td>
<td>Mar 31</td>
<td>5%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Midterm</th>
<th>STC 1012 &amp; AHS 1689</th>
<th>Mar 2, 7:00 – 8:50pm</th>
<th>25%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final</td>
<td>??</td>
<td>??</td>
<td>50%</td>
</tr>
</tbody>
</table>

- All 5 sections have the same coursework and grading
- **No late policy**
- Emergencies considered if formal proof provided
“An algorithm is a finite, definite, effective procedure, with some input and some output.”

— Donald Knuth

A procedure that efficiently computes an “answer” to a problem.

- Algorithmic problems
- Algorithm design techniques
- Type of algorithms
- Algorithm analysis: efficiency and formal proof
**Sorting**

**Problem**


**Sorting**: same array of integers in increasing order $\forall i$, $A[i] \leq A[i + 1]$

---

**Algorithm 1: SelectionSort** (array $A$ of size $n$)

1. for $i = 1$ to $n$ do
2.  \hspace{1em} $minIndex = i$
3.  \hspace{1em} for $j = i + 1$ to $n$ do
4.  \hspace{2em} if $A[j] < A[minIndex]$ then
5.  \hspace{3em} $minIndex = j$
6.  \hspace{1em} $A[i] \leftrightarrow A[minIndex]$ \hspace{1em} // in-place swap
7. return $A$
Correctness and Efficiency

On the $i$-th iteration, SelectionSort selects the $i$-th smallest element $A[\text{minIndex}]$ and swaps it with $A[i]$.

- Outer loop: $n$ iterations
- Inner loop: $n - i$ iterations for the $i$-th outer iteration
- In total: $\sum_{i=1}^{n} (n - i) = n^2 - \frac{n(n+1)}{2}$ comparisons, $n$ swaps
- Complexity $O(n^2)$
How to sort in decreasing order?

- reduce to something familiar

  negate, sort increasingly, negate

- or modify SelectionSort directly

---

**Algorithm 2: SelectionSort** (array $A$ of size $n$)

1. for $i = 1$ to $n$ do
2.   $\maxIndex = i$
3.     for $j = i + 1$ to $n$ do
4.       if $A[j] > A[\maxIndex]$ then
5.           $\maxIndex = j$
6.     end if
7.   $A[i] \leftrightarrow A[\maxIndex]$ // in-place swap
8. return $A$
Nice properties of SelectionSort

- in-place: no extra memory is needed
  
  All right, we do need $O(1)$ extra memory...


  Suppose students are sorted alphabetically, then after SelectionSort by GPA, students with the same GPA are still alphabetically sorted.
Plan

1. Recap
2. More Examples
3. Merge Sort
A different sorting algorithm

Algorithm 3: InsertionSort (array $A$ of size $n$)

1. for $i = 2$ to $n$ do
2.     key = $A[i]$
3.     $j = i - 1$
4.     while $j > 0$ && $A[j] > key$ do
6.         $j = j - 1$
7.     $A[j + 1] = key$ // insert the key
8. return $A$
Correctness of InsertionSort

courtesy to CLRS

At the beginning of the $i$-th iteration, $A[1], \ldots, A[i - 1]$ has been ordered increasingly.
Efficiency of InsertionSort

- $O(n^2)$ comparisons, read textbook for details
- in-place?
- stable?
- enjoys a nice property that other sorting algs do not share
Still another sorting algorithm

**Algorithm 4: BubbleSort** (array $A$ of size $n$)

1. \textbf{for} $i = 1$ to $n - 1$ \textbf{do}

2. \hspace{1em} isdone $= true$

3. \hspace{2em} \textbf{for} $j = n$ \textbf{down to} $i + 1$ \textbf{do}

4. \hspace{3em} \textbf{if} $A[j] < A[j - 1]$ \textbf{then}

5. \hspace{4em} $A[j] \leftrightarrow A[j - 1]$ \hspace{1em} \text{// in-place swap}

6. \hspace{2em} isdone $= false$

7. \hspace{1em} \textbf{if} isdone \textbf{then}

8. \hspace{2em} \textbf{break}

9. \hspace{1em} \textbf{return} $A$

Still $O(n^2)$ comparisons. in-place? stable?

Y-L. Yu (UW)
Still another sorting algorithm

Algorithm 5: BubbleSort (array \(A\) of size \(n\))

1. for \(i = 1\) to \(n - 1\) do
2.   isdone = true
3.   for \(j = n\) downto \(i + 1\) do
4.     if \(A[j] < A[j - 1]\) then
5.       \(A[j] \leftrightarrow A[j - 1]\) \hspace{1cm} // in-place swap
6.       isdone = false
7.   if isdone then
8.     break
9. return \(A\)

Still \(O(n^2)\) comparisons. in-place? stable?
Is $O(n^2)$ the best we can do for Sorting?

- No, if we find a better algorithm
- Yes, if we prove a lower bound $\Omega(n^2)$

Which one do you think it is?
Problem


Maximum: find maximum element in $A$

Algorithm 6: Maximum(array $A$ of size $n$)

1. $\max = A[1]$
2. for $i = 2$ to $n$ do
3.     if $A[i] > \max$ then
4.         $\max = A[i]$
5. return $\max$

- We actually used (a version of) it in which of our sorting algorithms?
- $n - 1$ comparisons. Optimal!
Finding cont’

Problem


MaxMin: find maximum and minimum element in $A$

Algorithm 7: MaxMin(array $A$ of size $n$)

1. $max = A[1]$
2. $min = A[1]$
3. for $i = 2$ to $n$ do
   4. if $A[i] > max$ then
      5. $max = A[i]$
   else if $A[i] < min$ then
      6. $min = A[i]$
4. return $max, min$

- $2(n - 1)$ comparisons, but can be improved!
Reduction revisited

**Algorithm 8:** MaxSub(array A of size n)

1. $A = \text{Sort}(A)$
2. return $A[n]$

**Algorithm 9:** MaxMinSub(array A of size n)

1. $A = \text{Sort}(A)$
2. return $A[n], A[1]$
Algorithm 8: Maximum Sub(array $A$ of size $n$)
1. $A = \text{Sort}(A)$
2. return $A[n]$

Algorithm 9: MaxMin Sub(array $A$ of size $n$)
1. $A = \text{Sort}(A)$
2. return $A[n], A[1]$

- At least $O(n \log n)$ comparisons; suboptimal
- Reduction is powerful but can be overkill

Never “reduce” a problem to something harder!
Plan

1. Recap

2. More Examples

3. Merge Sort
A lower bound on comparison sorting

- $n!$ possible permutations.
- A binary tree with depth $h$ can have at most $2^h$ leaves.
- $2^h \geq n! \approx n^n \iff h \geq n \log n$.

Figure: courtesy to CLRS
Merge sort

Split, solve, merge, and recurse

Algorithm 8: mSort (array $A$, $p$, $r$)

1. if $p < r$ then
2. \[ q = \lceil (p + r)/2 \rceil \] // split at $q$
3. mSort ($A$, $p$, $q$) // solve the left piece
4. mSort ($A$, $q$, $r$) // solve the right piece
5. Merge ($A$, $p$, $q$, $r$) // merge
The merge function

**Algorithm 9: Merge** (array $A$, $p$, $q$, $r$)

1. for $i = 1$ to $q - p + 1$ do
   2. $L[i] = A[p + i - 1]$  // copy the left piece
2. for $j = 1$ to $r - q$ do
5. $i = 1$, $j = 1$
6. for $k = p$ to $r$ do
   7. if $j > r - q$ then  // right piece is empty
      8. $A[k] = L[i]$, $i++$
   9. else if $i <= q - p + 1$ && $L[i] \leq R[j]$ then
      10. $A[k] = L[i]$, $i++$  // pick the smaller one
   12. else
Analysis of mSort

- Merge costs \( O(r - p) \)
- with \( p = 1, r = n \)

\[
T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n)
\]

- Claim: \( T(n) = \Theta(n \log n) \)

This matches our lower bound, hence we have found an optimal algorithm!
Yet another sorting algorithm

**Algorithm 10: CountingSort** (array \( A \) of size \( n \))

```
1 for i = 1 to n do
2    count[A[i]]++
3 csum = 1
4 for j = 1 to k do
5    c = count[j]
6    count[j] = csum
7    csum += c
8 for i = 1 to n do
9    S[count[A[i]]] = A[i]
10   count[A[i]]++
11 return S
```

\( O(n + k) \) complexity. No comparison; in-place? Stable?