CSCI 241

Lecture 5
Recursive Sorting:
Mergesort and Quicksort
Announcements

• First programming assignment (A1) out today.

• Mentor hours: 4-7pm, CF162/164
Goals:

• Know the generic steps of a divide-and-conquer algorithm.

• Thoroughly understand the mechanism of mergesort and quicksort.

• Be prepared to implement **merge** and **partition** helper methods.
## Incremental Algorithms

solve a problem a little bit at a time.

Natural programming mechanism: loops

<table>
<thead>
<tr>
<th>i</th>
<th>A sorted</th>
<th>?</th>
</tr>
</thead>
</table>

insertion sort
Divide-and-Conquer Algorithms

solve a problem by breaking it into smaller problems.

Natural programming mechanism: recursion

https://upload.wikimedia.org/wikipedia/commons/f/fe/Quicksort.gif
Divide-and-Conquer Algorithms

solve a problem by breaking it into smaller problems.

Natural programming mechanism: recursion

Three generic steps:
1. Divide (into sub-problems)
2. Conquer (the sub-problems)
3. Combine (into a solution to the original problem)
Divide-and-Conquer Algorithms

solve a problem by breaking it into smaller problems.

Natural programming mechanism: recursion

Three generic steps:
1. Divide (into sub-problems)
2. Conquer (the sub-problems)
3. Combine (into a solution to the original problem)
Why are we talking about divide-and-conquer, I thought we were learning how to sort things?
An example of Divide-and-Conquer

```python
/** sort A[start..end] using mergesort */
mergeSort(A, start, end):
    if (A.length < 2):
        return
    mid = (end-start)/2  # 1. Divide
    mergeSort(A,start,mid)
    mergeSort(A,mid, end)  # 2. Conquer
    merge(A, start, mid, end)  # 3. Combine
```
/** sort A[start..end] using mergesort */
mergeSort(A, start, end):
  if (A.length < 2):
    return
  mid = (end-start)/2
  Divide
  mergeSort(A, start, mid)  Conquer (left)
  mergeSort(A, mid, end)    Conquer (right)
  merge(A, start, mid, end) Combine
1. Spec

/** sort A[start..end] using mergesort */
mergeSort(A, start, end):
  if (A.length < 2):
    return
  mid = (end-start)/2

2. Base case

mergeSort(A, start, mid)  Conquer (left)
mergeSort(A, mid, end)    Conquer (right)

3. Progress

merge(A, start, mid, end)  Combine
mergeSort(A, start, end):
  if (A.length < 2):
    return
  mid = (end−start)/2
  sort A[start..mid]
  sort A[mid..end]
  merge(A, start, mid, end)
Merge Step

- Merge two halves, each of which is sorted.

https://facultyweb.cs.wwu.edu/~wehrwes/courses/csci241_18f/img/merge.gif
# Merge step: Loop Invariant

## Precondition

| B  | sorted |  | sorted |
|----|--------| |--------|
| A  |        |  |        |

## Invariant

<table>
<thead>
<tr>
<th>B</th>
<th>copied</th>
<th>not yet copied</th>
<th>copied</th>
<th>not yet copied</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>merged</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Postcondition

| B  | sorted |  | sorted |
|----|--------| |--------|
| A  |        |  | merged |
Merge step

### Pre

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>mid</td>
</tr>
</tbody>
</table>

| B sorted | sorted |

| A   | ?   |

### Inv

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>B copied</td>
<td>not yet copied</td>
</tr>
</tbody>
</table>

| copied | not yet copied |

| k   |

| A   | merged | ? |

### Post

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>mid</td>
</tr>
</tbody>
</table>

| sorted | sorted |

| A   | merged |

---

**merge(A, start, mid, end):**

- B = a deep copy of A
- i = start
- j = mid
- k = 0

**while** i < mid and **j** < end:

  - **if** B[i] < B[j]:
    - A[k] = B[i]
    - i++
    - k++
  - **else**:
    - A[k] = B[j]
    - j++
    - k++

**while** i < mid:

  - A[k] = B[i]
  - i++, k++

**while** j < end:

  - A[k] = B[j]
  - j++, k++

---

Smaller thing goes first

Ran out of things in one list or the other

Copy remaining things from nonempty half
/** sort A[start..end] using mergesort */
mergeSort(A, start, end):
    if (A.length < 2):
        return
    mid = (end-start)/2
    Divide

    mergeSort(A, start, mid)  Conquer (left)

    mergeSort(A, mid, end)  Conquer (right)

    merge(A, start, mid, end)  Combine

https://visualgo.net/bn/sorting
Merge step

merge(A, start, mid, end):
    B = a deep copy of A
    i = start
    j = mid
    k = 0

while i < mid and j < end:
    if B[i] < B[j]:  
        A[k] = B[i]
        i++
    else:
        A[k] = B[j]
        j++
        k++

while i < mid:
    A[k] = B[i]
    i++, k++

while j < end:
    A[k] = B[j]
    j++, k++

ABCD:
What’s the runtime of the merge step?
A. O(1)
B. O(n)
C. O(n^2)
D. O(n^3)
/** sort A[start..end] using mergesort */
mergeSort(A, start, end):
  if (A.length < 2):
    return
  mid = (end - start)/2
  mergeSort(A, start, mid)
  mergeSort(A, mid, end)
  merge(A, start, mid, end)

MergeSort: Runtime

O(1)
O(1)
O(huh?)
O(huh?)
O(n)
How many times can we divide \( n \) by 2 before we hit 1?

\[
n/2^x = 1
\]

\[
n = 2^x
\]

\[
x = \log_2 n
\]