Sorting Concluded

David Kauchak
CS140
Spring 2023

Administrative

Assignment 2 out

LC meetings

Mentor hours this week:
- No mentor hours today
- Thursday
  - 9-11am (David)
  - 7-9pm (Emily)

What does it do?

A[r] is called the **pivot**
Partitions the elements A[p...r-1] into two sets, those ≤ pivot and those > pivot
Operates in place

Final result:

```
Partition(A, p, r)
1 i ← p - 1
2 for j ← p to r - 1
4       i ← i + 1
5 swap A[i] and A[r]
6 swap A[i + 1] and A[r]
7 return i + 1
```

```
Partition(A, p, r)
1 i ← p - 1
2 for j ← p to r - 1
4       i ← i + 1
5 swap A[i] and A[r]
6 swap A[i + 1] and A[r]
7 return i + 1
```
Partition(A, p, r)
1 i ← p - 1
2 for j ← p to r - 1
4 i ← i + 1
5 swap A[i] and A[j]
6 swap A[i + 1] and A[r]
7 return i + 1
What's happening?
i

j

... 5 1 2 7 8 4 3 6 ...

\[ p \]

\[ r \]

Partition(A, p, r)
1 \( i \leftarrow p - 1 \)
2 for \( j \leftarrow p \) to \( r - 1 \)
3 if \( A[j] \leq A[p] \)
4 \( i \leftarrow i + 1 \)
5 swap \( A[i] \) and \( A[j] \)
6 swap \( A[i + 1] \) and \( A[r] \)
7 return \( i + 1 \)
Partition running time?

**Θ(n)**

```
PARTITION(A, p, r)
1    i ← p - 1
2    for j ← p to r - 1
4            i ← i + 1
5        swap A[i] and A[r]
6    swap A[i + 1] and A[r]
7    return i + 1
```

**QuickSort**

```
QUICKSORT(A, p, r)
1    if p < r
2        q ← PARTITION(A, p, r)
3        QUICKSORT(A, p, q - 1)
4        QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1    i ← p - 1
2    for j ← p to r - 1
4            i ← i + 1
5        swap A[i] and A[r]
6    swap A[i + 1] and A[r]
7    return i + 1
```
QuickSort(A, p, r)
1 if p < r
2 q ← Partition(A, p, r)
3 QuickSort(A, p, q - 1)
4 QuickSort(A, q + 1, r)
QuickSort(A, p, r)
1. if p < r
2. q ← Partition(A, p, r)
3. QuickSort(A, p, q − 1)
4. QuickSort(A, q + 1, r)
What happens here?
Some observations

Divide and conquer: different than MergeSort – do the work before recursing

Is Quicksort correct?

How many times is/can an element be selected as a pivot?

What happens after an element is selected as a pivot?
Is Quicksort correct?
Assuming Partition is correct

Proof by induction
- Base case: Quicksort works on a list of 1 element
- Inductive case:
  - Assume Quicksort sorts arrays for arrays of smaller < n elements, show that it works to sort n elements
  - If partition works correctly then we have:
  - and, by our inductive assumption, we have:

Running time of Quicksort?
Worst case?

Each call to Partition splits the array into an empty array and n-1 array

Quicksort: Worse case running time

\[ T(n) = T(n-1) + \Theta(n) \]

Which is? \( \Theta(n^2) \)

When does this happen?
- sorted
- reverse sorted
- near sorted/reverse sorted

Quicksort best case?

Each call to Partition splits the array into two equal parts

\[ T(n) = 2T(n/2) + \Theta(n) \]

\( \Theta(n \log n) \)

When does this happen?
- random data?
Quicksort Average case?

How close to “even” splits do they need to be to maintain an $\Theta(n \log n)$ running time?

Say the Partition procedure always splits the array into some constant ratio $b$-to-$a$, e.g. 9-to-1

What is the recurrence?

$$T(n) \leq T\left(\frac{a}{a+b}n\right) + T\left(\frac{b}{a+b}n\right) + cn$$
What is the depth of the tree?

Leaves will have different heights
Want to pick the deepest leaf
Assume $a < b$

Cost of the tree

Cost of each level $\leq cn$
Cost of the tree

Cost of each level ≤ cn
Times the maximum depth
\[ O(n \log_{\frac{b}{a+b}} n) \]

Why not?
\[ \Theta(n \log_{\frac{b}{a+b}} n) \]

Quicksort average case: take 2

What would happen if half the time Partition produced a “bad” split and the other half “good”?

\[ T(n) = 2T(\frac{n-1}{2}) + \Theta(n) \]
Quicksort average case: take 2

We absorb the “bad” partition. In general, we can absorb any constant number of “bad” partitions.

\[ T(n) = T\left(\frac{n-1}{2}\right) + T\left(\frac{n-1}{2}\right) + \Theta(n) \]

How can we avoid the worst case?

Inject randomness into the data

\[
\text{RANDOMIZED-PARTITION}(A, p, r)
\]

1. \( i \leftarrow \text{RANDOM}(p, r) \)
2. \( \text{swap } A[r] \text{ and } A[i] \)
3. \( \text{returnPARTITION}(A, p, r) \)

What is the running time of randomized Quicksort?

Worst case?

\( \mathcal{O}(n^2) \)

Still could get very unlucky and pick “bad” partitions at every step.

Sorting bounds

Mergsort is \( \Theta(n \log n) \)

Quicksort is \( \mathcal{O}(n \log n) \) on average

Can we do better?
Comparison-based sorting
Sorted order is determined based only on a comparison between input elements

- $A[i] \leq A[j]$
- $A[i] \geq A[j]$

Do any of the sorting algorithms we've looked at use additional information?

- No
- All the algorithms we've seen are comparison-based sorting algorithms

In Java (and many languages) for a class of objects to be sorted we define a comparator

What does it do?

- Just compares any two elements
- Useful for comparison-based sorting algorithms

Can we do better than $O(n \log n)$ for comparison-based sorting approaches?
**Decision-tree model**

Full binary tree representing the comparisons between elements by a sorting algorithm

- Internal nodes contain indices to be compared
- Leaves contain a complete permutation of the input

Tracing a path from root to leave gives the correct reordering/permutation of the input for an input

```
[3, 12, 7]   ⇒   [3, 7, 12]
[3, 7, 12]   ⇒   [3, 7, 12]
```

Is 12 ≤ 7 or is 12 > 7?
A decision tree model

[12, 7, 3]
Is 12 ≤ 7 or is 12 > 7?

A decision tree model

[12, 7, 3]
Is 12 ≤ 3 or is 12 > 3?
A decision tree model

[12, 7, 3]

Is 7 ≤ 3 or is 7 > 3?

90

[12, 7, 3]

3, 2, 1

92
A decision tree model

[7, 12, 3]
How many leaves are in a decision tree?

Leaves **must** have all possible permutations of the input

What if decision tree model didn’t?

Some input would exist that didn’t have a correct reordering

Input of size \( n \), \( n! \) leaves

A lower bound

What is the worst-case number of comparisons for a tree?
A lower bound

The longest path in the tree, i.e. the height

A complete binary tree has $2^h$ leaves

$2^h \geq n!$

$h \geq \log n!$

$h = \Omega(n \log n)$ from group work!

Can we do better?