Graduate Algorithms

CS673-2016F-04

Sorting I

David Galles

Department of Computer Science
University of San Francisco

04-0: Divide & Conquer

- Divide a problem into 2 or more smaller subproblems
- Recursively solve each subproblem
- Combine the solutions of the subproblems

04-1: Divide & Conquer

- Mergesort:
  - Divide the list in half
  - Recursively sort each half of the list
  - Merge the sorted lists together
- Dividing the list is easy (no real work required)
- Combining solutions harder

04-2: Divide & Conquer

- Quicksort:
  - Pick a pivot element
  - Divide the list into elements $<$ pivot, elements $>$ pivot
  - Recursively sort each of these two segments
  - No work required after recursive step
- Dividing the list is harder
- Combining solutions is easy (no real work required)

04-3: Quicksort

```
Quicksort(A, low, high)
    if (low < high) then
        pivotindex ← Partition(A, low, high)
        Quicksort(A, low, pivotindex − 1)
        Quicksort(A, pivotindex + 1, high)
```

04-4: Quicksort

- How can we efficiently partition the list?

04-5: Quicksort

- How can we efficiently partition the list?
- Method 1:
  - Maintain two indices, i and j
  - Everything to left of $i \leq$ pivot
  - Everything to right if $j \geq$ pivot
  - Start i at beginning of the list, j at the end of the list, move them in maintaining the conditions above

04-6: Quicksort

- How can we efficiently partition the list?
- Method 2:
  - Maintain two indices, i and j
  - Everything to left of $i \leq$ pivot
  - Everything between i and $j \geq$ pivot
  - Start both i and j at beginning of the list, increase them while maintaining the conditions above

04-7: Partition

```
Partition(A, low, high)
    pivot = A[high]
    i ← low - 1
    for j ← low to high - 1 do
        if (A[j] ≤ pivot then
            i ← i + 1
            swap A[i] ↔ A[j]
    swap A[i+1] ↔ A[high]
```

04-8: Partition

Partition example:

5 7 1 3 6 2 8 4

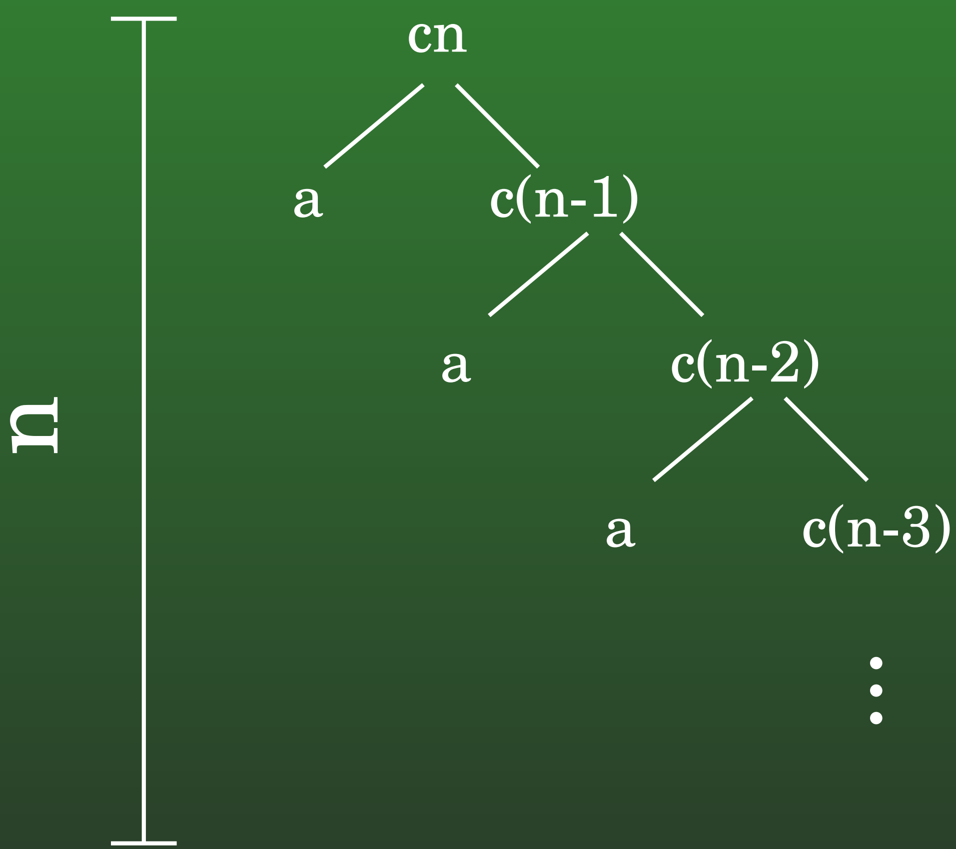
04-9: Quicksort

- Running time for Quicksort: Intuition
  - Worst case: list is split into size 0, size (n-1)

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \end{aligned}$$

Recursion Tree

04-10: Quicksort



cn

a+c(n-1)

a+c(n-2)

a+c(n-3

⋮

$$(n-1)a + \sum_{i=1}^{n} ci$$

04-11: **Quicksort**

Confirm $O(n^2)$ with substitution method:

$$T(n) \;=\; T(n-1) + c * n$$

04-12: Quicksort

Confirm $O(n^2)$ with substitution method:

$$\begin{aligned} T(n) &= T(n-1) + c*n \\ &\leq c_1*(n-1)^2 + c*n \\ &\leq c_1*(n^2-2n+1) + c*n \\ &\leq c_1*n^2 + (c-2*c_1+1/n)*n \\ &\leq c_1*n^2 \end{aligned}$$

(if $c_1 > (c+1/n)/2$)

04-13: Quicksort

Confirm $\Omega(n^2)$ with substitution method:

$$
\begin{aligned}
T(n) &= T(n-1) + c * n \\
&\geq c_1 * (n-1)^2 + c * n \\
&\geq c_1 * (n^2 - 2n + 1) + c * n \\
&\geq c_1 * (n^2 - 2n) + c * n \\
&\geq c_1 * n^2 + (c - 2 * c_1) * n \\
&\geq c_1 * n^2
\end{aligned}
$$

(if $c_1 > c/2$)

04-14: Quicksort

- Running time for Quicksort: Intuition
  - Best case: list is split in half

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + c * n \\
&\in \Theta(n \lg n)
\end{aligned}
$$

(Using the master theorem)

04-15: Quicksort

- Running time for Quicksort: Intuition
  - Average case:
    - What if we split the problem into size $(1/9)n$ and $(8/9)n$
    - What if we split the problem into size $(1/100)n$ and $(99/100)n$

(Show recursion trees)

04-16: Quicksort

- Worst Case:

$$T(n) = \max_{0 \leq q \leq n-1} T(q) + T(n - q - 1) + \Theta(n)$$

04-17: Quicksort

- Worst Case:

$$T(n) = \max_{0 \leq q \leq n-1} T(q) + T(n-q-1) + \Theta(n)$$

Guess $T(n) \in O(n^2)$

$$\begin{aligned} T(n) &\leq \max_{0 \leq q \leq n-1} c_1 q^2 + c_1 (n-q-1)^2 + c_2 * n \\ &\leq c_1 * \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2) c_1 + c_2 * n \end{aligned}$$

Maximizing $q^2 + (n-q-1)^2$ over range $0 \leq q \leq n-1$

04-18: Quicksort

Maximizing $q^2 + (n - q - 1)^2$ over range $0 \leq q \leq n - 1$

- 2nd derivative with respect to q is positive
- Maximim value needs to occur at the endpoints: $q = 0$ or $q = n - 1$

04-19: Quicksort

$$
\begin{aligned}
T(n) &\leq \max_{0 \leq q \leq n-1} c_1 q^2 + c_1(n-q-1)^2 + c_2 * n \\
&\leq c_1 * \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2) c_1 + c_2 * n \\
&\leq c_1(n-1)^2 + c_2 * n \\
&\leq c_1 n^2 - 2c_1 n + c_1 + c_2 * n \\
&\leq c_1 n^2
\end{aligned}
$$

(if $c_1 > c_2/2$)

04-20: Quicksort

- Average case:
  - What is the average case?
  - We can $assume$ that all permutations of the list are equally likely (is this a good assumption?)
  - What else can we do?

Partition

```
Partition(A, low, high)
    pivot = A[high]
    i ← low - 1
    for j ← low to high - 1 do
        if (A[j] ≤ pivot) then
            i ← i + 1
            swap A[i] ↔ A[j]
    swap A[i+1] ↔ A[hight]
```

04-22: Randomized Partition

```
Partition(A, low, high)
    swap A[high] ↔ A[random(low,high)]
    pivot = A[high]
    i ← low - 1
    for j ← low to high - 1 do
        if (A[j] ≤ pivot) then
            i ← i + 1
            swap A[i] ↔ A[j]
    swap A[i+1] ↔ A[hight]
```

Quicksort Analysis

- OK, we can assume that all permutations are equally likely (especially if we randomize partition)
- How long does quicksort take in the average case?

04-24: Quicksort Analysis

- Time for quicksort dominated by time spent in partition procedure
- Partition can be called a maximum of n times (why)?
- Time for each call to partition is $\Theta(1)$ + # of times through for loop
- Total number of times the test (A[j] $\leq$ pivot) is done is proportional to the time spent for the loop
- Therefore, the total # of times the test (A[j] $\leq$ pivot) is a bound on the time for the entire algorithm

04-25: Quicksort Analysis

Some definitions:

- Define z_i to be the ith smallest element in the list
- Define Z_{ij} to be the set of elements $z_i, z_{i+1}, \ldots z_j$

So, if our array A = {3, 4, 1, 9, 10, 7} then:

- $z_1 = 1$, $z_2 = 3$, $z_3 = 4$, etc
- Z_{35} = {4, 7, 9}
- Z_{46} = {7, 9, 10}

Quicksort Analysis

- Each pair of elements can be compared at most once (why)?
- Define an indicator variable $X_{ij} = I\{z_i$ is compared to $z_j\}$

$$
\begin{aligned}
X &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \\
E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] \\
E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]
\end{aligned}
$$

04-27: Quicksort Analysis

- Calculating $E[X_{ij}]$:
  - When will element z_i be compared to z_j?
- A = {1, 2, 3, 4, 5, 6, 7, 8, 9 , 10}
- If pivot = 6
  - 6 will be compared to every other element
  - 1-5 will never be compared to anything in 7-10

04-28: Quicksort Analysis

- Calculating $E[X_{ij}]$:
  - Given any two elements z_i, z_j, if we pick some element x as a pivot such that $z_i < x < z_j$, then z_i and z_j will never be compared to each other
  - z_i and z_j will be compared with each other when the first element chosen Z_{ij} is either z_i or z_j

04-29: Quicksort Analysis

Pr{z_i is compared to z_j}	=	Pr{z_i or z_j is first pivot selected from Z_{ij}
	=	Pr{z_i is first from Z_{ij}} + Pr{z_j is first from Z_{ij} }
	=	$1/(j-i+1) + 1/(j-i+1)$
	=	$2/(j-i+1)$

04-30: Quicksort Analysis

$$
\begin{aligned}
E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k} \\
&< \sum_{i=1}^{n-1} 2\ln(n-i)+1
\end{aligned}
$$

04-31: Quicksort Analysis

$$\begin{aligned}
E[X] &< \sum_{i=1}^{n-1} 2\ln(n-i)+1 \\
&< \sum_{i=1}^{n-1} 2\ln(n)+1 \\
&< 2*n\ln(n)+1 \\
&\in O(n\lg n)
\end{aligned}$$

 # Alternate Parition strategy

```
Partition(A, low, high)
    pivot = A[high]
    i = low
    j = high - 1
    while (i < j)
        while (A[i] < pivot)
            i++
        while (A[j] > pivot)
            j - -
        if (i < j)
            swap A[i] ↔ A[j]
            i++
            j - -
    swap A[i] ↔ A[high]
```

Alternate Parition strategy

```
Partition(A, low, high)
    pivot = A[high]
    i = low
    j = high - 1
    while (i < j)
        while (A[i] ≤ pivot)
            i++
        while (A[j] ≥ pivot)
            j - -
        if (i < j)
            swap A[i] ↔ A[j]
            i++
            j - -
    swap A[i] ↔ A[high]
```

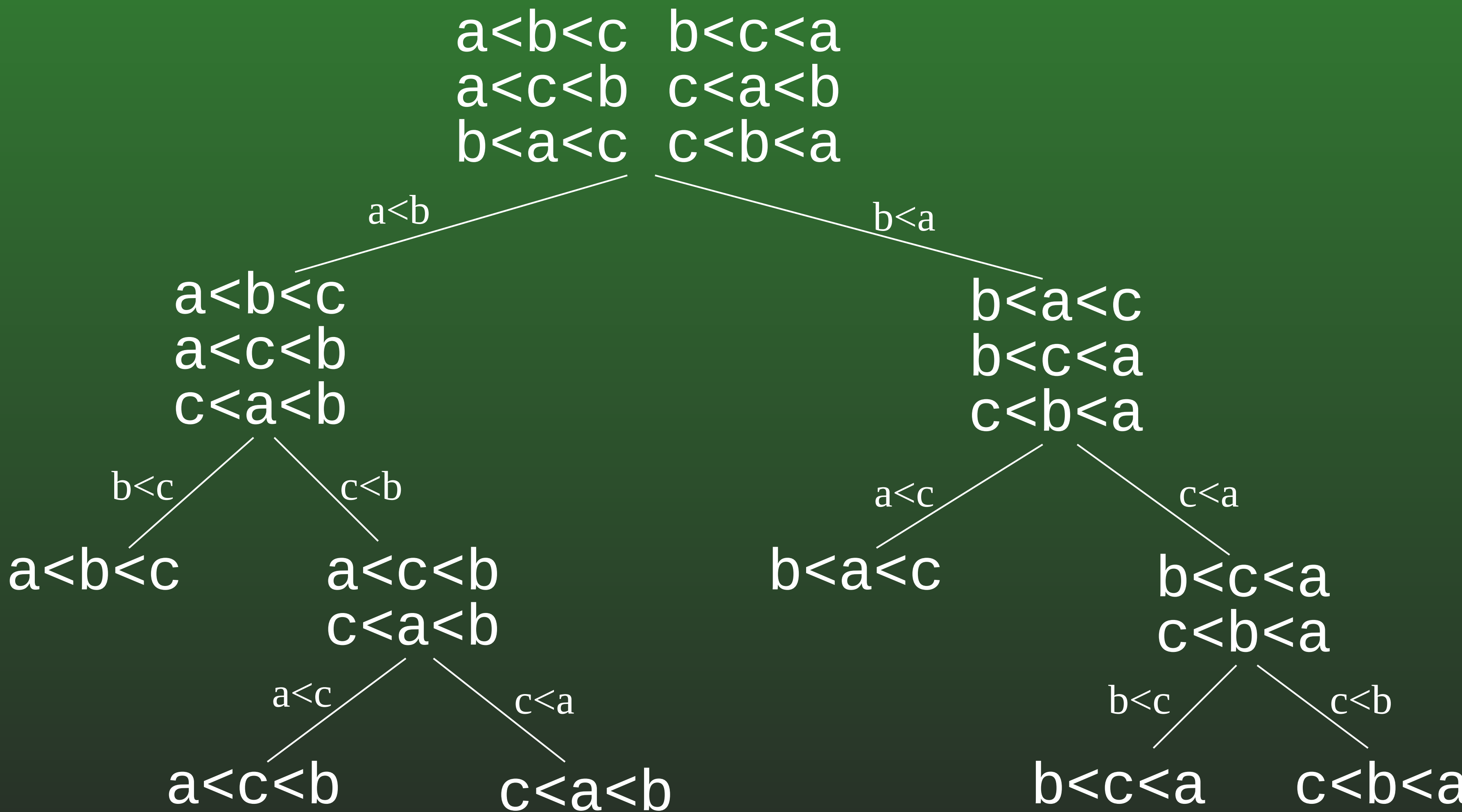
What happens if we change $<$ to $\leq$?

04-34: Comparison Sorting

- Comparison sorts work by comparing elements
  - Can only compare 2 elements at a time
  - Check for $<$, $>$, $=$.
- All the sorts we have seen so far (Insertion, Quick, Merge, Heap, etc.) are comparison sorts
- If we know nothing about the list to be sorted, we need to use a comparison sort

04-35: Decision Trees

Insertion Sort on list $\{a, b, c\}$



04-36: Decision Trees

- Every comparison sorting algorithm has a decision tree
- What is the best-case number of comparisons for a comparison sorting algorithm, given the decision tree for the algorithm?

Decision Trees

- Every comparison sorting algorithm has a decision tree
- What is the best-case number of comparisons for a comparison sorting algorithm, given the decision tree for the algorithm?
  - (The depth of the shallowest leaf) + 1
- What is the worst case number of comparisons for a comparison sorting algorithm, given the decision tree for the algorithm?

04-38: Decision Trees

- Every comparison sorting algorithm has a decision tree
- What is the best-case number of comparisons for a comparison sorting algorithm, given the decision tree for the algorithm?
  - (The depth of the shallowest leaf) + 1
- What is the worst case number of comparisons for a comparison sorting algorithm, given the decision tree for the algorithm?
  - The height of the tree – (depth of the deepest leaf) + 1

04-39: Decision Trees

- What is the largest number of nodes for a tree of depth d?

04-40: Decision Trees

- What is the largest number of nodes for a tree of depth d?
  - 2^d
- What is the minimum height, for a tree that has n leaves?

04-41: Decision Trees

- What is the largest number of nodes for a tree of depth d?
  - 2^d
- What is the minimum height, for a tree that has n leaves?
  - $\lg n$
- How many leaves are there in a decision tree for sorting n elements?

04-42: Decision Trees

- What is the largest number of nodes for a tree of depth d?
  - 2^d
- What is the minimum height, for a tree that has n leaves?
  - $\lg n$
- How many leaves are there in a decision tree for sorting n elements?
  - $n!$
- What is the minimum height, for a decision tree for sorting n elements?

04-43: Decision Trees

- What is the largest number of nodes for a tree of depth d?
  - 2^d
- What is the minimum height, for a tree that has n leaves?
  - $\lg n$
- How many leaves are there in a decision tree for sorting n elements?
  - $n!$
- What is the minimum height, for a decision tree for sorting n elements?
  - $\lg n!$

04-44: $\lg(n!) \in \Omega(n \lg n)$

$$
\begin{aligned}
\lg(n!) &= \lg(n * (n-1) * (n-2) * \ldots * 2 * 1) \\
&= (\lg n) + (\lg(n-1)) + (\lg(n-2)) + \ldots \\
&\quad +(\lg 2) + (\lg 1) \\
&\geq \underbrace{(\lg n) + (\lg(n-1)) + \ldots + (\lg(n/2))}_{n/2 \text{ terms}} \\
&\geq \underbrace{(\lg n/2) + (\lg(n/2)) + \ldots + \lg(n/2)}_{n/2 \text{ terms}} \\
&= (n/2)\lg(n/2) \\
&\in \Omega(n \lg n)
\end{aligned}
$$

Sorting Lower Bound

- All comparison sorting algorithms can be represented by a decision tree with $n!$ leaves
- Worst-case number of comparisons required by a sorting algorithm represented by a decision tree is the height of the tree
- A decision tree with $n!$ leaves must have a height of at least $n \lg n$
- All comparison sorting algorithms have worst-case running time $\Omega(n \lg n)$