Warm up

Show $\log(n!) = \Theta(n \log n)$

Hint: show $n! \leq n^n$

Hint 2: show $n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$
\[ \log n! = O(n \log n) \]

\[ n! = n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 2 \cdot 1 \]

\[ n^n = n \cdot n \cdot n \cdot \ldots \cdot n \cdot n \]

\[ n! \leq n^n \]
\[ \Rightarrow \log(n!) \leq \log(n^n) \]
\[ \Rightarrow \log(n!) \leq n \log n \]
\[ \Rightarrow \log(n!) = O(n \log n) \]
\[ \log n! = \Omega(n \log n) \]
\[ n! = n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot \frac{n}{2} \cdot \left(\frac{n}{2} - 1\right) \cdot \ldots \cdot 2 \cdot 1 \]
\[ \left(\frac{n}{2}\right)^n = \frac{n}{2} \cdot \frac{n}{2} \cdot \frac{n}{2} \cdot \ldots \cdot \frac{n}{2} \cdot 1 \cdot \ldots \cdot 1 \cdot 1 \]
\[ n! \geq \left(\frac{n}{2}\right)^n \]
\[ \Rightarrow \log(n!) \geq \log \left(\left(\frac{n}{2}\right)^n\right) \]
\[ \Rightarrow \log(n!) \geq \frac{n}{2} \log \frac{n}{2} \]
\[ \Rightarrow \log(n!) = \Omega(n \log n) \]
Today’s Keywords

• Divide and Conquer
• Sorting
• Quicksort
• Decision Tree
• Worst case lower bound
CLRS Readings

- Chapter 7
- Chapter 8
Homeworks

• HW2 due 11pm tonight!
  – Divide and conquer
  – Closest Pair of Points
  – Remember to submit relevant .java or .py files (no .zip!)
• HW3 due 11pm Wednesday Feb. 20
  – Divide and conquer
  – Written (use LaTeX!)
Quicksort

- **Idea:** pick a **pivot** element, recursively sort two sublists around that element
- **Divide:** select an element $p$, $\text{Partition}(p)$
- **Conquer:** recursively sort left and right sublists
- **Combine:** Nothing!
Partition (Divide step)

• Given: a list, a pivot value $p$

Start: unordered list

Goal: All elements $< p$ on left, all $> p$ on right
Is it worth it?

• Using Quickselect to pick median guarantees $\Theta(n \log n)$ run time
  – Approach has very large constants
  – If you really want $\Theta(n \log n)$, better off using MergeSort

• Better approach: Random pivot
  – Very small constant (very fast algorithm)
  – Expected to run in $\Theta(n \log n)$ time
    • Why? Unbalanced partitions are very unlikely
Quicksort Run Time

- If the partition is always $\frac{n}{10}$ order statistic:

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + n$$
$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + n$$
Quicksort Run Time

• If the partition is always $\frac{n}{10}$ order statistic:

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + n$$

$$T(n) = \Theta(n \log n)$$
Quicksort Run Time

• If the partition is always $d^{th}$ order statistic:

```
1  5  2  3  6  4  7  8  10  9  11  12
```

• Then we shorten by $d$ each time

$$T(n) = T(n - d) + n$$

$$T(n) = O(n^2)$$

What’s the probability of this occurring?
Probability of $n^2$ run time

We must consistently select pivot from within the first $d$ terms

Probability first pivot is among $d$ smallest: $\frac{d}{n}$

Probability second pivot is among $d$ smallest: $\frac{d}{n-d}$

Probability all pivot are among $d$ smallest:

$$\frac{d}{n} \cdot \frac{d}{n-d} \cdot \frac{d}{n-2d} \cdot \ldots \cdot \frac{d}{2d} \cdot 1 = \frac{1}{\binom{n}{d}}!$$
Random Pivot

- Using Quickselect to pick median guarantees $\Theta(n \log n)$ run time
  - Approach has very large constants
  - If you really want $\Theta(n \log n)$, better off using MergeSort
- Better approach: Random pivot
  - Very small constant (very fast algorithm)
  - Expected to run in $\Theta(n \log n)$ time
    - Why? Unbalanced partitions are very unlikely
Formal Argument for $n \log n$ Average

• Remember, run time counts comparisons!
• Quicksort only compares against the **pivot**
  – Element $i$ only compared to element $j$ if one of them was the **pivot**
Partition (Divide step)

- Given: a list, a pivot value $p$

Start: unordered list

Goal: All elements $< p$ on left, all $> p$ on right
Formal Argument for $n \log n$ Average

• What is the probability of comparing two given elements?

1 2 3 4 5 6 7 8 9 10 11 12

• (Probability of comparing 3 and 4) = 1
  – Why?
    • Otherwise I wouldn’t know which came first
    • ANY sorting algorithm must compare adjacent elements
Formal Argument for $n \log n$ Average

• What is the probability of comparing two given elements?

1 2 3 4 5 6 7 8 9 10 11 12

• (Probability of comparing 1 and 12) = $\frac{2}{12}$
  – Why?
    • We only compare 1 with 12 if either was chosen as the first pivot
    • Otherwise they would be divided into opposite sublists
Formal Argument for $n \log n$ Average

• Probability of comparing $i$ and $j$ (where $j > i$):
  – inversely proportional to the number of elements between $i$ and $j$
    
    \[
    \frac{2}{j-i+1}
    \]

• Expected (average) number of comparisons:
  
  \[
  \sum_{i<j} \frac{2}{j-i+1}
  \]
Expected number of Comparisons

Consider when \( i = 1 \)

\[
\sum_{i<j} \frac{2}{j - i + 1}
\]

\begin{array}{cccccccccccc}
  1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\end{array}

Compared if 1 or 2 are chosen as pivot
(these will always be compared)

Sum so far: \( \frac{2}{2} \)
Expected number of Comparisons

Consider when \( i = 1 \)

\[
\sum_{i<j} \frac{2}{j - i + 1}
\]

Compared if 1 or 3 are chosen as pivot
(but not if 2 is ever chosen)

Sum so far: \( \frac{2}{2} + \frac{2}{3} \)
Expected number of Comparisons

Consider when $i = 1$

Compared if 1 or 4 are chosen as pivot
(but not if 2 or 3 are chosen)

Sum so far: $\frac{2}{2} + \frac{2}{3} + \frac{2}{4}$
Expected number of Comparisons

Consider when $i = 1$

\[\sum_{i<j} \frac{2}{j - i + 1}\]

Compared if 1 or 12 are chosen as pivot
(but not if 2 -> 11 are chosen)

Overall sum: \(\frac{2}{2} + \frac{2}{3} + \frac{2}{4} + \frac{2}{5} + \cdots + \frac{2}{n}\)
Expected number of Comparisons

$$\sum_{i<j} \frac{2}{j - i + 1}$$

When $i = 1$: $2 \left( \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} \right)$

$n$ terms overall

$$\sum_{i<j} \frac{2}{j - i + 1} \leq 2n \left( \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \right) = \Theta(\log n)$$

Quicksort overall: expected $\Theta(n \log n)$
Sorting, so far

• Sorting algorithms we have discussed:
  – Mergesort \(O(n \log n)\)
  – Quicksort \(O(n \log n)\)

• Other sorting algorithms (will discuss):
  – Bubblesort \(O(n^2)\)
  – Insertionsort \(O(n^2)\)
  – Heapsort \(O(n \log n)\)

Can we do better than \(O(n \log n)\)?
Worst Case Lower Bounds

• Prove that there is no algorithm which can sort faster than $O(n \log n)$
• Non-existence proof!
  – Very hard to do
Strategy: Decision Tree

- Sorting algorithms use comparisons to figure out the order of input elements
- Draw tree to illustrate all possible execution paths

Possible execution path:

Result of comparison:

Permutation of sorted list:

- One comparison

• Sorting algorithms use comparisons to figure out the order of input elements
• Draw tree to illustrate all possible execution paths
Strategy: Decision Tree

- Worst case run time is the longest execution path
- i.e., “height” of the decision tree

Possible execution path

Worst case run time is the longest execution path
i.e., “height” of the decision tree

\( \log(n!) \)

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

\( n! \) Possible permutations

Permutation of sorted list
Strategy: Decision Tree

- Conclusion: Worst Case Optimal run time of sorting is $\Theta(n \log n)$
  - There is no (comparison-based) sorting algorithm with run time $o(n \log n)$
Sorting, so far

• Sorting algorithms we have discussed:
  – Mergesort \( O(n \log n) \) Optimal!
  – Quicksort \( O(n \log n) \) Optimal!

• Other sorting algorithms
  – Bubblesort \( O(n^2) \)
  – Insertionsort \( O(n^2) \)
  – Heapsort \( O(n \log n) \) Optimal!
Speed Isn’t Everything

• Important properties of sorting algorithms:
  • Run Time
    – Asymptotic Complexity
    – Constants
  • In Place (or In-Situ)
    – Done with only constant additional space
  • Adaptive
    – Faster if list is nearly sorted
  • Stable
    – Equal elements remain in original order
  • Parallelizable
    – Runs faster with many computers