Topic 1: Introduction and Median Finding

David Woodruff

Grading and Course Policies

• All available here: https://www.cs.cmu.edu/~15451/policies.html

<table>
<thead>
<tr>
<th>Component</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Written Homeworks</td>
<td>20% (5% each)</td>
</tr>
<tr>
<td>3 Oral Homeworks</td>
<td>15% (5% each)</td>
</tr>
<tr>
<td>Online Quizzes+Class Participation+Bonus</td>
<td>12% (see below)</td>
</tr>
<tr>
<td>Midterm exams (in class)</td>
<td>30% (15% each)</td>
</tr>
<tr>
<td>Final exam</td>
<td>23%</td>
</tr>
</tbody>
</table>

• 12 weekly online quizzes due Friday 11:59pm
• Solve written homeworks individually. Come to office hours or ask questions on piazza! Latex solutions and submit on gradescope
• Oral homeworks can be solved in groups of 3
• Each quiz is worth 1 point, also up to 3 points for participation, bonus problems

Homework

• Each HW has 3-4 problems

• Typically, one problem is a programming problem – submit via Autolab (languages accepted are Java, C, C++, Ocaml, SML)

• For oral HWs you can collaborate, but write the programming problem yourself. Each team has 45 minutes to present the 3 problems. Feel free to bring in notes!

• Cite any reference material or webpage if you use it

• Randomized grading – we will choose 2 of the 3 problems to grade, while always grading the programming problem

• Late homeworks and “grace/mercy” days – please see the website for details!
Goals of the Course

- Design and analyze algorithms!
- **Algorithms**: dynamic programming, divide-and-conquer, hashing and data structures, randomization, network flows, linear programming
- **Analysis**: recurrences, probabilistic analysis, amortized analysis, potential functions
- **Dual to Algorithms**: complexity theory and lower bounds
- **New Models**: online algorithms, machine learning, data streams

 Guarantees on Algorithms

- Want **provable guarantees** on the running time of algorithms
- **Why?**
  - **Composability**: if we know an algorithm runs in time at most $T$ on any input, don’t have to worry what kinds of inputs we run it on
  - **Scaling**: how does the time grow as the input size grows?
  - **Designing better algorithms**: what are the most time-consuming steps?

Example: Median Finding

- In the median-finding problem, we have an array $a_1, a_2, ..., a_n$
  - and want the index $i$ for which there are exactly $\lfloor n/2 \rfloor$ numbers larger than $a_i$

  - **How can we find the median?**
    - Check each item to see if it is the median: $\Theta(n^2)$ time
    - Sort items with MergeSort (deterministic) or QuickSort (randomized): $\Theta(n \log n)$ time
    - Can we find it faster? What about finding the k-th smallest number?

QuickSelect Algorithm to Find the k-th Smallest Number

- Assume $a_1, a_2, ..., a_n$ are all distinct for simplicity
- Choose a random element $a_i$ in the list – call this the “pivot”
- Compare each $a_j$ to $a_i$
  - Let LESS = {$a_j$ such that $a_j < a_i$}
  - Let GREATER = {$a_j$ such that $a_j > a_i$}
- If $k \leq |\text{LESS}|$, find the k-th smallest element in LESS
- If $k = |\text{LESS}| + 1$, output the pivot $a_i$
- Else find the $(k-|\text{LESS}|-1)$-th smallest item in GREATER
- Similar to Randomized QuickSort, but **only recurse on one side!**
Bounding the Running Time

- **Theorem:** the expected number of comparisons for QuickSelect is at most 4n
- Let $T(n) = \max_k T(n, k)$, where $T(n, k)$ is the expected number of comparisons to find the $k$-th smallest item in an array of length $n$, maximized over all arrays
- $T(n)$ is a non-decreasing function of $n$
- Let’s show $T(n) < 4n$ by induction
  - **Base case:** $T(1) = 0 < 4$
  - **Inductive hypothesis:** $T(n-1) < 4(n-1)$

What About Deterministic Algorithms?

- Can we get an algorithm which does not use randomness and always performs $O(n)$ comparisons?
  - **Idea:** suppose we could deterministically find a pivot which partitions the input into two pieces LESS and GREATER each of size $\lceil \frac{n}{2} \rceil$
  - **How to do that?**
    - Find the median and then partition around that
      - Um... finding the median is the original problem we want to solve....

Bounding the Running Time

- Suppose we have an array of length $n$
- Pivot randomly partitions the array into two pieces, LESS and GREATER, with $|\text{LESS}| + |\text{GREATER}| = n-1$
  - $|\text{LESS}|$ is uniform in the set $\{0, 1, 2, 3, ..., n-1\}$
  - Since $T(i)$ is non-decreasing with $i$, to upper bound $T(n)$ we can assume we recurse on larger half
  - $T(n) \leq n - 1 + \frac{2}{n} \sum_{i=0}^{n-1} 4i$
    - by inductive hypothesis
    - $< n - 1 + 4\left(\frac{3n}{4}\right)$
    - since the average $\frac{2}{n} \sum_{i=0}^{n-1} i$ is at most $3n/4$
    - $< 4n$
    - completing the induction

Deterministically Finding a Pivot

- **Idea:** deterministically find a pivot with $O(n)$ comparisons to partition the input into two pieces LESS and GREATER each of size at least $3n/10-1$
- **DeterministicSelect:**
  1. Group the array into $n/5$ groups of size 5 and find the median of each group
  2. Recursively, find the median of medians. Call this $p$
  3. Use $p$ as a pivot to split into subarrays LESS and GREATER
  4. Recurse on the appropriate piece
- **Theorem:** DeterministicSelect makes $O(n)$ comparisons to find the $k$-th smallest item in an array of size $n$
Running Time of DeterministicSelect

- **DeterministicSelect:**
  1. Group the array into n/5 groups of size 5 and find the median of each group
  2. Recursively, find the median of medians. Call this p
  3. Use p as a pivot to split into subarrays LESS and GREATER
  4. Recurse on the appropriate piece

- Step 1 takes O(n) time since it takes O(1) time to find the median of 5 elements
- Step 2 takes T(n/5) time
- Step 3 takes O(n) time

Claim: |LESS| ≥ 3n/10-1 and |GREATER| ≥ 3n/10-1

Running Time of DeterministicSelect

- **T(n) ≤ cn + T(n/5) + T(7n/10)**

- Time is cn + \(\frac{n}{5}\) + \(\frac{7n}{10}\) ≤ 10cn
- Recurrence works because n/5 + 7n/10 < n

- For constants c and a_1, a_2, …, a_r with a_1 + a_2 + … + a_r < 1, the recurrence T(n) ≤ T(a_1n) + T(a_2n) + … + T(a_rn) + cn solves to T(n) = O(n)
  - If instead a_1 + a_2 + … + a_r = 1, the recurrence solves to T(n) = O(n log n)
  - If we use median of 3 in DeterministicSelect instead of median of 5, what happens?