CS311H

Prof: Peter Stone

Department of Computer Science
The University of Texas at Austin

Multi-person Elections

- Suppose that the votes of n people for several (more than 2) candidates for a particular office are the elements of a sequence. To win, a candidate must receive a majority (more than half) of the votes. Devise a divide-and-conquer algorithm that determines whether a candidate received a majority and if so determine who this candidate is. (must use constant, i.e. O(1), memory) What is its Big-O runtime?

Good Morning, Colleagues

Good Morning, Colleagues

Are there any questions?

Logistics

- No discussion next Wed.

Logistics

- No discussion next Wed.
- Long module on proving correctness of mergesort for Thursday.

Logistics

- No discussion next Wed.
- Long module on proving correctness of mergesort for Thursday.
- Rest of modules published.

Logistics

- No discussion next Wed.
- Long module on proving correctness of mergesort for Thursday.
- Rest of modules published.
- More Big-O practice on last slides of this slide deck

Questions / Important Points

- What's the point of the fast multiplication module?

Questions / Important Points

- What's the point of the fast multiplication module?
  - Synthesizes recurrences and Master theorem,

Questions / Important Points

- What's the point of the fast multiplication module?
  - Synthesizes recurrences and Master theorem,and starts to introduce analysis of algorithms.

Questions / Important Points

- What's the point of the fast multiplication module?
  - Synthesizes recurrences and Master theorem,and starts to introduce analysis of algorithms.
- What do we need to know?

Questions / Important Points

- What's the point of the fast multiplication module?
  - Synthesizes recurrences and Master theorem,and starts to introduce analysis of algorithms.
- What do we need to know?
  - Wouldn't expect you to come up with the algorithm on your own. . .

Questions / Important Points

- What's the point of the fast multiplication module?
  - Synthesizes recurrences and Master theorem,and starts to introduce analysis of algorithms.
- What do we need to know?
  - Wouldn't expect you to come up with the algorithm on your own. . .
  - . . . or even reproduce it without notes

Questions / Important Points

- What's the point of the fast multiplication module?
  - Synthesizes recurrences and Master theorem,and starts to introduce analysis of algorithms.
- What do we need to know?
  - Wouldn't expect you to come up with the algorithm on your own. . .
  - . . . or even reproduce it without notes
  - Given the algorithm, you should be able to come up with the recurrence and analyze the complexity.

Questions / Important Points

- What's the point of the fast multiplication module?
  - Synthesizes recurrences and Master theorem,and starts to introduce analysis of algorithms.
- What do we need to know?
  - Wouldn't expect you to come up with the algorithm on your own. . .
  - . . . or even reproduce it without notes
  - Given the algorithm, you should be able to come up with the recurrence and analyze the complexity.

Binary Search

Binary search can determine if a specific value x is in a sorted list of size n. It works by comparing x to the element in the middle of the list, and then searching half of the remaining list recursively.

Binary Search

Binary search can determine if a specific value x is in a sorted list of size n. It works by comparing x to the element in the middle of the list, and then searching half of the remaining list recursively.

- What is the recurrence relation describing the runtime?

Binary Search

Binary search can determine if a specific value x is in a sorted list of size n. It works by comparing x to the element in the middle of the list, and then searching half of the remaining list recursively.

- What is the recurrence relation describing the runtime?
- What is the Big-O runtime?

Binary Search

Binary search can determine if a specific value x is in a sorted list of size n. It works by comparing x to the element in the middle of the list, and then searching half of the remaining list recursively.

- What is the recurrence relation describing the runtime?
- What is the Big-O runtime?
- Would it help to do a ternary or quaternary search?

Multi-person Elections

- Suppose that the votes of n people for several (more than 2) candidates for a particular office are the elements of a sequence. To win, a candidate must receive a majority (more than half) of the votes. Devise a (constant space) divide-and-conquer algorithm that determines whether a candidate received a majority and if so determine who this candidate is. (must use constant, i.e. O(1), memory)

Multi-person Elections

- Suppose that the votes of n people for several (more than 2) candidates for a particular office are the elements of a sequence. To win, a candidate must receive a majority (more than half) of the votes. Devise a (constant space) divide-and-conquer algorithm that determines whether a candidate received a majority and if so determine who this candidate is. (must use constant, i.e. O(1), memory) What is its Big-O runtime?

Multi-person Elections

- Suppose that the votes of n people for several (more than 2) candidates for a particular office are the elements of a sequence. To win, a candidate must receive a majority (more than half) of the votes. Devise a (constant space) divide-and-conquer algorithm that determines whether a candidate received a majority and if so determine who this candidate is. (must use constant, i.e. O(1), memory) What is its Big-O runtime?
  - Hint: If you split the sequence in half (or off by one), note that a candidate could not have an overall majority without receiving a majority of votes in at least one of the 2 halves.

Compute a^n

- What is the run time of the trivial iterative method?

Compute a^n

- What is the run time of the trivial iterative method? $O(n)$

Compute a^n

- What is the run time of the trivial iterative method? $O(n)$
- Propose a divide and conquer algorithm which can provide better time complexity.

Compute a^n

- What is the run time of the trivial iterative method? $O(n)$
- Propose a divide and conquer algorithm which can provide better time complexity.
  - Find the recurrence relation and complexity.

Compute a^n

- What is the run time of the trivial iterative method? $O(n)$
- Propose a divide and conquer algorithm which can provide better time complexity.
  - Find the recurrence relation and complexity.

Answer: Let the runtime be $T(n)$.

Compute a^n

- What is the run time of the trivial iterative method? $O(n)$
- Propose a divide and conquer algorithm which can provide better time complexity.
  - Find the recurrence relation and complexity.

Answer: Let the runtime be $T(n)$. Since we have $a^n = a^{\frac{n}{2}} \times a^{\frac{n}{2}}$

Compute a^n

- What is the run time of the trivial iterative method? $O(n)$
- Propose a divide and conquer algorithm which can provide better time complexity.
  - Find the recurrence relation and complexity.

Answer: Let the runtime be $T(n)$. Since we have $a^n = a^{\frac{n}{2}} \times a^{\frac{n}{2}}$ and $a^{\frac{n}{2}}$ can be computed in $T(\frac{n}{2})$,

Compute a^n

- What is the run time of the trivial iterative method? $O(n)$
- Propose a divide and conquer algorithm which can provide better time complexity.
  - Find the recurrence relation and complexity.

Answer: Let the runtime be $T(n)$. Since we have $a^n = a^{\frac{n}{2}} \times a^{\frac{n}{2}}$ and $a^{\frac{n}{2}}$ can be computed in $T(\frac{n}{2})$, we get the recurrence relation: $T(n) = T(\frac{n}{2}) + 1$.

Compute a^n

- What is the run time of the trivial iterative method? $O(n)$
- Propose a divide and conquer algorithm which can provide better time complexity.
  - Find the recurrence relation and complexity.

Answer: Let the runtime be $T(n)$. Since we have $a^n = a^{\frac{n}{2}} \times a^{\frac{n}{2}}$ and $a^{\frac{n}{2}}$ can be computed in $T(\frac{n}{2})$, we get the recurrence relation: $T(n) = T(\frac{n}{2}) + 1$.
So $a = 1, b = 2, d = 0$.

Compute a^n

- What is the run time of the trivial iterative method? $O(n)$
- Propose a divide and conquer algorithm which can provide better time complexity.
  - Find the recurrence relation and complexity.

Answer: Let the runtime be $T(n)$. Since we have $a^n = a^{\frac{n}{2}} \times a^{\frac{n}{2}}$ and $a^{\frac{n}{2}}$ can be computed in $T(\frac{n}{2})$, we get the recurrence relation: $T(n) = T(\frac{n}{2}) + 1$.
So $a = 1, b = 2, d = 0$. Since we have $a = b^d$, by Master Theorem, we have

Compute a^n

- What is the run time of the trivial iterative method? $O(n)$

- Propose a divide and conquer algorithm which can provide better time complexity.
  - Find the recurrence relation and complexity.

Answer: Let the runtime be $T(n)$. Since we have $a^n = a^{\frac{n}{2}} \times a^{\frac{n}{2}}$ and $a^{\frac{n}{2}}$ can be computed in $T(\frac{n}{2})$, we get the recurrence relation: $T(n) = T(\frac{n}{2}) + 1$.
So $a = 1, b = 2, d = 0$. Since we have $a = b^d$, by Master Theorem, we have
$T(n) = O(n^0 \log n) = O(\log n)$

Compute a^n

- What is the run time of the trivial iterative method? $O(n)$

- Propose a divide and conquer algorithm which can provide better time complexity.
  - Find the recurrence relation and complexity.

Answer: Let the runtime be $T(n)$. Since we have $a^n = a^{\frac{n}{2}} \times a^{\frac{n}{2}}$ and $a^{\frac{n}{2}}$ can be computed in $T(\frac{n}{2})$, we get the recurrence relation: $T(n) = T(\frac{n}{2}) + 1$.
So $a = 1, b = 2, d = 0$. Since we have $a = b^d$, by Master Theorem, we have
$T(n) = O(n^0 \log n) = O(\log n)$ which is better than $O(n)$.

Nuts and Bolts

- You are given a collection of n bolts of different widths and n corresponding nuts. You are allowed to try a nut and bolt together, from which you can determine whether the nut is larger than the bolt, smaller than the bolt, or matches the bolt exactly. However, there is no way to compare two nuts together or two bolts together. Create and analyze the expected runtime of an efficent algorithm to match each bolt to its nut.

Nuts and Bolts Solution

Nuts and Bolts Solution

Solution: Randomly select a nut and traverse all bolts to find its match. Meanwhile, partition all bolts into two sets. One contains all bolts smaller than this nut and the other contains all bolts larger than this nut. Then after finding the matched bolt, use this bolt to do same partition for all nuts. These two partitions can be done in $2n$ comparisons. Then we need to deal with two sets, each of size $\frac{n}{2}$ **(on average)**. Thus we get the recurrence relation below:

$$T(n) = 2T(\frac{n}{2}) + 2n$$

By Master Theorem, we have $T(n) = O(n \log n)$.

More Big-O practice

- Let a be any positive number. Show that $a^n = O(n!)$.