Learning Goals

- State the implementation of the Quicksort algorithm
- Define Las Vegas and Monte Carlo algorithms
- Analyze the expected running time of a Las Vegas algorithm using linearity of expectation

Setup and the algorithm

- Input: A set S of n integers $a_1, \ldots, a_n$.
- Output: Sorted array of the n integers in increasing order.

Setup and the algorithm

- Input: A set S of n integers $a_1, \ldots, a_n$.
- Output: Sorted array of the n integers in increasing order.
- Recall: Deterministic algorithms: Merge Sort (divide and conquor, running time $O(n \log n)$.

Setup and the algorithm

- Input: A set S of n integers $a_1, \ldots, a_n$.
- Output: Sorted array of the n integers in increasing order.
- Recall: Deterministic algorithms: Merge Sort (divide and conquor, running time $O(n \log n)$.
- Recall lower bound: no deterministic algorithm can make $o(n \log n)$ comparisons in the worst case.

Setup and the algorithm

- Input: A set S of n integers $a_1, \ldots, a_n$.
- Output: Sorted array of the n integers in increasing order.
- Recall: Deterministic algorithms: Merge Sort (divide and conquor, running time $O(n \log n)$.
- Recall lower bound: no deterministic algorithm can make $o(n \log n)$ comparisons in the worst case.
- One of the best known sorting algorithm — Quicksort(S): If $|S| \leq 3$, return sorted S. Otherwise, pick an element a_i uniformly at random from S, form two sets: $S^+ := \{a_j : a_j > a_i\}$ and $S^- := \{a_j : a_j < a_i\}$. Return Quicksort($S^-$), a_j, Quicksort(S^+).

Categorization of Randomized Algorithms

- A *randomized algorithm* is simply an algorithm with access to random coins.

Categorization of Randomized Algorithms

- A *randomized algorithm* is simply an algorithm with access to random coins.
- Two categories of randomized algorithms:

Categorization of Randomized Algorithms

- A *randomized algorithm* is simply an algorithm with access to random coins.
- Two categories of randomized algorithms:
  - A *Las Vegas algorithm* always terminates with a correct solution; its running time is a random variable.

Categorization of Randomized Algorithms

- A *randomized algorithm* is simply an algorithm with access to random coins.
- Two categories of randomized algorithms:
  - A *Las Vegas algorithm* always terminates with a correct solution; its running time is a random variable.
  - A *Monte Carlo algorithm* returns a correct solution only probabilistically; its running time may or may not be a random variable.

Categorization of Randomized Algorithms

- A *randomized algorithm* is simply an algorithm with access to random coins.
- Two categories of randomized algorithms:
  - A *Las Vegas algorithm* always terminates with a correct solution; its running time is a random variable.
  - A *Monte Carlo algorithm* returns a correct solution only probabilistically; its running time may or may not be a random variable.
- Later in the semester we will also encounter algorithms that give *approximations*, and we reason about the quality of the approximations in a probabilistic manner.

Analysis of Quicksort

Theorem

The expected running time of Quicksort is $O(n \log n)$.

Analysis of Quicksort

Theorem

The expected running time of Quicksort is $O(n \log n)$.

- Observation: Forming S^+ and S^- altogether takes $O(n)$ time.

Analysis of Quicksort

Theorem

The expected running time of Quicksort is $O(n \log n)$.

- Observation: Forming S^+ and S^- altogether takes $O(n)$ time.
- Intuition: if a_j always roughly cuts S in the middle, then the running time is roughly $T(n) \approx 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$.

Analysis of Quicksort

Theorem

The expected running time of Quicksort is $O(n \log n)$.

- Observation: Forming S^+ and S^- altogether takes $O(n)$ time.
- Intuition: if a_j always roughly cuts S in the middle, then the running time is roughly $T(n) \approx 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$.

To simplify the presentation, we analyze a variant of Quicksort:

- ModifiedQuicksort(S):
  - If $|S| \leq 3$, return sorted S.
  - Pick an element a_i uniformly at random from S, form two sets: $S^+ := \{a_j : a_j > a_i\}$ and $S^- := \{a_j : a_j < a_i\}$. If $|S^-| < \frac{n}{4}$ or $|S^+| < \frac{n}{4}$, repeat (i.e., pick another a_j independently at random).
  - Output ModifiedQuicksort(S^-), a_j, ModifiedQuicksort(S^+).

Analysis

First thing: how many a_j's do we have to try, in expectation, to have $\frac{n}{4} \leq |S^-| \leq \frac{3n}{4}$?

Analysis

First thing: how many a_j's do we have to try, in expectation, to have $\frac{n}{4} \leq |S^-| \leq \frac{3n}{4}$?

- Let X be the number of attempts till we succeed.

Analysis

First thing: how many a_j's do we have to try, in expectation, to have $\frac{n}{4} \leq |S^-| \leq \frac{3n}{4}$?

- Let X be the number of attempts till we succeed.
- Each attempt succeds with probability $\frac{1}{2}$.

Analysis

First thing: how many a_j's do we have to try, in expectation, to have $\frac{n}{4} \leq |S^-| \leq \frac{3n}{4}$?

- Let X be the number of attempts till we succeed.
- Each attempt succeds with probability $\frac{1}{2}$.
- So $\mathrm{E}[X] = 2$.

Analysis

Second: How do we put together the recursion?

Analysis

Second: How do we put together the recursion?
What doesn't work: Since both $|S^-|$ and $|S^+|$ are $\leq \frac{3n}{4}$,

$$T(n) \leq 2T(\frac{3n}{4}) + 2n \leq 2\left(n + 2 \cdot \frac{3n}{4} + 4 \cdot \frac{3^2 n}{4^2} + \cdots\right).$$

Analysis

Second: How do we put together the recursion?
What doesn't work: Since both $|S^-|$ and $|S^+|$ are $\leq \frac{3n}{4}$,

$$T(n) \leq 2T(\frac{3n}{4}) + 2n \leq 2\left(n + 2 \cdot \frac{3n}{4} + 4 \cdot \frac{3^2 n}{4^2} + \cdots\right).$$

This analysis is wasteful and the bound too loose. We should be more careful.

Analysis

Second: How do we put together the recursion?
What doesn't work: Since both $|S^-|$ and $|S^+|$ are $\leq \frac{3n}{4}$,

$$T(n) \leq 2T(\frac{3n}{4}) + 2n \leq 2\left(n + 2 \cdot \frac{3n}{4} + 4 \cdot \frac{3^2 n}{4^2} + \cdots\right).$$

This analysis is wasteful and the bound too loose. We should be more careful.

- A subproblem is said to be type j if the size of the set it considers is in $(n(\frac{3}{4})^{j+1}, n(\frac{3}{4})^j]$.

Analysis

Second: How do we put together the recursion?
What doesn't work: Since both $|S^-|$ and $|S^+|$ are $\leq \frac{3n}{4}$,

$$T(n) \leq 2T(\frac{3n}{4}) + 2n \leq 2\left(n + 2 \cdot \frac{3n}{4} + 4 \cdot \frac{3^2 n}{4^2} + \cdots\right).$$

This analysis is wasteful and the bound too loose. We should be more careful.

- A subproblem is said to be type j if the size of the set it considers is in $(n(\frac{3}{4})^{j+1}, n(\frac{3}{4})^j]$.
- The original problem is of type 0.

Analysis

Second: How do we put together the recursion?
What doesn't work: Since both $|S^-|$ and $|S^+|$ are $\leq \frac{3n}{4}$,

$$T(n) \leq 2T(\frac{3n}{4}) + 2n \leq 2\left(n + 2 \cdot \frac{3n}{4} + 4 \cdot \frac{3^2 n}{4^2} + \cdots\right).$$

This analysis is wasteful and the bound too loose. We should be more careful.

- A subproblem is said to be type j if the size of the set it considers is in $(n(\frac{3}{4})^{j+1}, n(\frac{3}{4})^j]$.
- The original problem is of type 0.
- Key observation: after each recursion, the subproblems newly generated are disjoint, and their types are strictly higher.

Analysis

Second: How do we put together the recursion?
What doesn't work: Since both $|S^-|$ and $|S^+|$ are $\leq \frac{3n}{4}$,

$$T(n) \leq 2T(\frac{3n}{4}) + 2n \leq 2\left(n + 2 \cdot \frac{3n}{4} + 4 \cdot \frac{3^2 n}{4^2} + \cdots\right).$$

This analysis is wasteful and the bound too loose. We should be more careful.

- A subproblem is said to be type j if the size of the set it considers is in $(n(\frac{3}{4})^{j+1}, n(\frac{3}{4})^j]$.
- The original problem is of type 0.
- Key observation: after each recursion, the subproblems newly generated are disjoint, and their types are strictly higher.
- All subproblems of the same type must be disjoint. So the number of type j subproblems created throughout the algorithm is $\leq (\frac{4}{3})^{j+1}$.

Final steps

- Total time spent on type j subproblems: $O(n)$.

Final steps

- Total time spent on type j subproblems: $O(n)$.
  - More concretely, there are k subproblems of type j, where $k \leq (\frac{4}{3})^{j+1}$.

Final steps

- Total time spent on type j subproblems: $O(n)$.
  - More concretely, there are k subproblems of type j, where $k \leq (\frac{4}{3})^{j+1}$.
  - Let X_i denote the number of attempts in the i-th subproblem of type j, then $\mathrm{E}[X_i] = 2$.

Final steps

- Total time spent on type j subproblems: $O(n)$.
  - More concretely, there are k subproblems of type j, where $k \leq (\frac{4}{3})^{j+1}$.
  - Let X_i denote the number of attempts in the i-th subproblem of type j, then $\mathrm{E}[X_i] = 2$.
  - Total running time for type j subproblems is at most:

$$\sum_{i=1}^{k} \left(\frac{3}{4}\right)^{j} n \, \mathrm{E}\left[X_i\right] \leq \left(\frac{4}{3}\right)^{j+1} \cdot \left(\frac{3}{4}\right)^{j} n \cdot 2 = O(n),$$

where we used linearity of expectation.

Final steps

- Total time spent on type j subproblems: $O(n)$.
  - More concretely, there are k subproblems of type j, where $k \leq (\frac{4}{3})^{j+1}$.
  - Let X_i denote the number of attempts in the i-th subproblem of type j, then $\mathrm{E}[X_i] = 2$.
  - Total running time for type j subproblems is at most:

$$\sum_{i=1}^{k} \left(\frac{3}{4}\right)^{j} n \,\mathrm{E}\left[X_i\right] \leq \left(\frac{4}{3}\right)^{j+1} \cdot \left(\frac{3}{4}\right)^{j} n \cdot 2 = O(n),$$

    where we used linearity of expectation.
- Number of types: $\leq \log_{\frac{4}{3}} n$.

Final steps

- Total time spent on type j subproblems: $O(n)$.
  - More concretely, there are k subproblems of type j, where $k \leq (\frac{4}{3})^{j+1}$.
  - Let X_i denote the number of attempts in the i-th subproblem of type j, then $\mathrm{E}[X_i] = 2$.
  - Total running time for type j subproblems is at most:

$$\sum_{i=1}^{k} \left(\frac{3}{4}\right)^{j} n \,\mathrm{E}\left[X_i\right] \leq \left(\frac{4}{3}\right)^{j+1} \cdot \left(\frac{3}{4}\right)^{j} n \cdot 2 = O(n),$$

  where we used linearity of expectation.
- Number of types: $\leq \log_{\frac{4}{3}} n$.
- Total running time $O(n \log n)$.

Final steps

- Total time spent on type j subproblems: $O(n)$.
  - More concretely, there are k subproblems of type j, where $k \leq (\frac{4}{3})^{j+1}$.
  - Let X_i denote the number of attempts in the i-th subproblem of type j, then $\mathrm{E}[X_i] = 2$.
  - Total running time for type j subproblems is at most:

$$\sum_{i=1}^{k} \left(\frac{3}{4}\right)^{j} n \, \mathrm{E}\left[X_i\right] \leq \left(\frac{4}{3}\right)^{j+1} \cdot \left(\frac{3}{4}\right)^{j} n \cdot 2 = O(n),$$

    where we used linearity of expectation.
- Number of types: $\leq \log_{\frac{4}{3}} n$.
- Total running time $O(n \log n)$.
- See reading material for a direct analysis of Quicksort.

Final steps

- Total time spent on type j subproblems: $O(n)$.
  - More concretely, there are k subproblems of type j, where $k \leq (\frac{4}{3})^{j+1}$.
  - Let X_i denote the number of attempts in the i-th subproblem of type j, then $\mathrm{E}[X_i] = 2$.
  - Total running time for type j subproblems is at most:

$$\sum_{i=1}^{k} \left(\frac{3}{4}\right)^{j} n \, \mathrm{E}\left[X_i\right] \leq \left(\frac{4}{3}\right)^{j+1} \cdot \left(\frac{3}{4}\right)^{j} n \cdot 2 = O(n),$$

    where we used linearity of expectation.
- Number of types: $\leq \log_{\frac{4}{3}} n$.
- Total running time $O(n \log n)$.
- See reading material for a direct analysis of Quicksort.
- Later in the semester we'll show that Quicksort in fact runs in time $O(n \log n)$ also *with high probability*.

An Alternative Analysis

- In class we discussed an alternative analysis.

An Alternative Analysis

- In class we discussed an alternative analysis.
- Instead of categorizing the subproblems by the lengths of their inputs, we may categorize by their distances from the original problem.

An Alternative Analysis

- In class we discussed an alternative analysis.
- Instead of categorizing the subproblems by the lengths of their inputs, we may categorize by their distances from the original problem.
- Level 1 is the original problem; level 1 include the two subproblems recursed from it; level 2 include the 4 subproblems generated by level 1 problems, etc..

An Alternative Analysis

- In class we discussed an alternative analysis.
- Instead of categorizing the subproblems by the lengths of their inputs, we may categorize by their distances from the original problem.
- Level 1 is the original problem; level 1 include the two subproblems recursed from it; level 2 include the 4 subproblems generated by level 1 problems, etc..
- The key observation is that the total expected work we do on each level is $O(n)$, because all these problems are disjoint.

An Alternative Analysis

- In class we discussed an alternative analysis.
- Instead of categorizing the subproblems by the lengths of their inputs, we may categorize by their distances from the original problem.
- Level 1 is the original problem; level 1 include the two subproblems recursed from it; level 2 include the 4 subproblems generated by level 1 problems, etc..
- The key observation is that the total expected work we do on each level is $O(n)$, because all these problems are disjoint.
- The number of levels is $O(\log n)$. So the total work we do in expectation is $O(n \log n)$.