Randomized Algorithms

Jared Saia
University of New Mexico

Quicksort

- Based on divide and conquer strategy
- Worst case is $\Theta(n^2)$
- Expected running time is $\Theta(n \log n)$
- In-place apart from the recursion stack
- Often fast in practice; performance depends on pivot selection and implementation

Quicksort

- **Divide:** Choose a pivot $x = A[q]$ and rearrange the array so that every element before the pivot is at most x and every element after it is greater than x
- **Conquer:** Recursively sort the two subarrays on either side of the pivot
- **Combine:** No separate work is needed; partitioning and the recursive calls sort the array in place

The Algorithm

```
// PRE: p and r are valid inclusive indices of A
// POST: A[p..r] is in sorted order
QUICKSORT(A, p, r)
  if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q-1)
    QUICKSORT(A, q+1, r)
```

Partition

Assume p and r are valid inclusive indices, and let $x = A[r]$. The algorithm returns q such that $A[q] = x$, $A[p \ldots q-1] \leq x$, and $A[q+1 \ldots r] > x$.

```
Partition(A, p, r)
    x ← A[r]
    i ← p − 1
    for j ← p to r − 1 do
        if A[j] ≤ x then
            i ← i + 1
            exchange A[i] and A[j]
    exchange A[i + 1] and A[r]
    return i + 1
```

Analysis

- The function Partition takes $O(n)$ time. Why?

Example Quicksort

- Quicksort the array [2, 6, 9, 1, 5, 3, 8, 7, 4]

Randomized Quicksort

- A fixed pivot rule can repeatedly produce badly unbalanced splits
- Q: How can we avoid dependence on a favorable input order?
- A: Choose each pivot uniformly at random from the current subarray
- For every fixed input with distinct keys, the expected running time is $\Theta(n \log n)$ over the algorithm's random choices

R-Partition

```
// PRE: p and r are valid inclusive indices
// POST: chooses a uniformly random pivot from A[p..r],
//       partitions around it, and returns its final index
RANDOMIZED-PARTITION(A, p, r)
  i = RANDOM-INTEGER(p, r)  // endpoints included
  exchange A[r] and A[i]
  return PARTITION(A, p, r)
```

Randomized Quicksort

```
// PRE: p and r are valid inclusive indices of A
// POST: A[p..r] is in sorted order
RANDOMIZED-QUICKSORT(A, p, r)
  if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q-1)
    RANDOMIZED-QUICKSORT(A, q+1, r)
```

Analysis

- R-Quicksort is a *randomized* algorithm
- The run time is a *random variable*
- We'd like to analyze the *expected* run time of R-Quicksort
- To do this, we first need to learn some basic probability theory.

Probability Definitions

(from Appendix C.3)

- A *random variable* assigns a numerical value to each outcome of a random experiment. For example, the outcome of a die roll can be represented by a random variable X.
- The *expected value* of a random variable X is:

$$E(X) = \sum_{x \in X} x \Pr(X = x)$$

For example, if X is the outcome of a three-sided die,

$$\begin{aligned} E(X) &= 1(1/3) + 2(1/3) + 3(1/3) \\ &= 2 \end{aligned}$$

Probability Definitions

- Two events A and B are *mutually exclusive* if $A \cap B = \emptyset$ (Example: A is the event that the outcome of a die is 1 and B is the event that the outcome of a die is 2)
- Two random variables X and Y are *independent* if for all x and y, $\Pr(X = x, Y = y) = \Pr(X = x)\Pr(Y = y)$ (Example: let X be the outcome of the first roll of a die, and Y be the outcome of the second roll of the die. Then X and Y are independent.)

Indicator Random Variables

- An *indicator random variable* for event A is defined as:

$$I(A) = \begin{cases} 1 & \text{if event } A \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

- Example: Let A be the event that the roll of a die equals 2. Then $I(A)$ is 1 if the die roll is 2 and 0 otherwise.

Linearity of Expectation

- Let X and Y be two random variables
- Then $E(X+Y) = E(X) + E(Y)$
- (Holds even if X and Y are not independent.)

- More generally, let $X_1, X_2, \ldots, X_n$ be n random variables
- Then

$$E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i)$$

Example

- For $1 \leq i \leq n$, let X_i be the outcome of the i-th roll of a three-sided die
- Then

$$E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = 2n$$

Example

- Indicator variables and linearity of expectation are a powerful combination
- The *Birthday Paradox* illustrates this point
- To analyze the run time of Quicksort, we will also use indicator r.v.'s and linearity of expectation (analysis will be similar to "birthday paradox" problem)

Birthday Paradox

- Assume there are m people in a room, and n days in a year
- Assume that each of these m people is born on a day chosen independently and uniformly at random from the n days
- Q: What is the expected number of pairs of individuals that have the same birthday?
- We can use indicator random variables and linearity of expectation to compute this

Analysis

- For all $1 \leq i < j \leq m$, let $X_{i,j}$ be an indicator random variable defined such that:
  - $X_{i,j} = 1$ if person i and person j have the same birthday
  - $X_{i,j} = 0$ otherwise
- Note that for all i, j,

$$\begin{aligned} E(X_{i,j}) &= P(\text{person i and j have same birthday}) \\ &= 1/n \end{aligned}$$

Analysis

- Let X be a random variable giving the number of pairs of people with the same birthday
- We want $E(X)$
- Then $X = \sum_{1 \leq i < j \leq m} X_{i,j}$
- So $E(X) = E(\sum_{1 \leq i < j \leq m} X_{i,j})$

Analysis

$$
\begin{aligned}
E(X) &= E\left(\sum_{1\le i<j\le m} X_{i,j}\right) \\
&= \sum_{1\le i<j\le m} E(X_{i,j}) && \text{LOE} \\
&= \sum_{1\le i<j\le m} 1/n \\
&= \binom{m}{2}\frac{1}{n} \\
&= \frac{m(m-1)}{2n}
\end{aligned}
$$

The second step follows by Linearity of Expectation (LOE)

Reality Check

- The expected number of matching pairs is at least 1 exactly when $m(m-1) \geq 2n$
- Equivalently, $m \geq \left(1+\sqrt{1+8n}\right)/2 \approx \sqrt{2n}+1/2$
- For $n = 365$, the smallest integer meeting the threshold is $m = 28$, giving an expected 1.04 matching pairs

In-Class Exercise

- Assume there are m people in a room, and n days in a year
- Assume that each person's birthday is chosen independently and uniformly at random from the n days
- Let X be the number of groups of *three* people who all have the same birthday. What is $E(X)$?
- Let $X_{i,j,k}$ be an indicator r.v. which is 1 if people i,j, and k have the same birthday and 0 otherwise

In-Class Exercise

- Q1: Write the expected value of X as a function of the $X_{i,j,k}$ (use linearity of expectation)
- Q2: What is $E(X_{i,j,k})$?
- Q3: What is the total number of groups of three people out of m?
- Q4: What is $E(X)$?

Plan of Attack

"If you get hold of the head of a snake, the rest of it is mere rope" - Akan Proverb

- We will analyze the *total* number of comparisons made by quicksort
- We will let X be the total number of comparisons made by R-Quicksort
- We will write X as the sum of a bunch of indicator random variables
- We will use linearity of expectation to compute the expected value of X

Notation

- Let A be the array to be sorted
- Assume all keys are distinct; for equal keys, break ties in an arbitrary way
- Let z_i be the i-th smallest element in the array A
- Let $Z_{i,j} = \{z_i, z_{i+1}, \ldots, z_j\}$

Indicator Random Variables

- Let $X_{i,j}$ be 1 if z_i is compared with z_j and 0 otherwise
- Note that $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j}$
- Further note that

$$E(X) = E\left(\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j}\right) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(X_{i,j})$$

Questions

- Q1: So what is $E(X_{i,j})$?
- A1: It is $P(z_i$ is compared to $z_j)$
- Q2: What is $P(z_i$ is compared to $z_j)$?
- A2: They are compared exactly when the first pivot chosen from $Z_{i,j}$ is either z_i or z_j
- Why?
  - Until a pivot is chosen from $Z_{i,j}$, all its elements remain in the same recursive subarray
  - If an internal element of $Z_{i,j}$ is chosen first, it splits z_i and z_j into separate subarrays, so they are never compared

More Questions

- The set $Z_{i,j}$ contains $j - i + 1$ elements
- Each is equally likely to be the first pivot chosen from this set
- Therefore

$$P(z_i \text{ is compared with } z_j) = \frac{1}{j-i+1} + \frac{1}{j-i+1}$$
$$= \frac{2}{j-i+1}$$

Conclusion

$$E(X_{i,j}) = P(z_i \text{ is compared to } z_j)$$
$$= \frac{2}{j - i + 1}$$

Putting it together

$$
\begin{aligned}
E(X) &= E\left(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{i,j}\right) \\
&= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E(X_{i,j}) && \text{LOE} \\
&= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1}\sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1}\sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\log n) \\
&= O(n \log n)
\end{aligned}
$$

Questions

- Q: Why is $\sum_{k=1}^{n} \frac{2}{k} = O(\log n)$?
- A:

$$\sum_{k=1}^{n} \frac{2}{k} = 2 \sum_{k=1}^{n} \frac{1}{k}$$
$$\leq 2(\ln n + 1) \quad \text{By an integral bound (p. 1067)}$$

Markov's Inequality

- We've shown that randomized Quicksort makes $O(n \log n)$ expected comparisons and therefore has expected running time $O(n \log n)$.
- But what does this tell us about the *probability* that the algorithm takes significantly more than that?
- To go from expectations to probability bounds, we can use a special inequality: Markov's inequality (next slide).
- This tells us (among other things) that the probability of taking 100 times the expected runtime is no more than 1/100.

Markov's Inequality

Let X be a nonnegative random variable. For any $\lambda > 0$,

$$\Pr(X \geq \lambda) \leq \frac{E(X)}{\lambda}.$$

Indeed, with $I = I\{X \geq \lambda\}$,

$$E(X) \geq E(XI) \geq \lambda \Pr(X \geq \lambda).$$

Rearranging proves the inequality.

How Fast Can We Sort?

- Q: What is a lower bound on the runtime of any sorting algorithm?
- We know that $\Omega(n)$ is a trivial lower bound
- But all the algorithms we've seen so far are $O(n \log n)$ (or $O(n^2)$), so is $\Omega(n \log n)$ a lower bound?

Comparison Sorts

- Definition: A sorting algorithm is a *comparison sort* if the sorted order it determines is based only on comparisons between input elements
- Heapsort, mergesort, quicksort, bubblesort, and insertion sort are all comparison sorts
- We will show that every comparison sort makes $\Omega(n \log n)$ comparisons in the worst case

Comparisons

- Assume we have an input sequence $A = (a_1, a_2, \ldots, a_n)$
- In a comparison sort, we only perform tests of the form $a_i < a_j$, $a_i \leq a_j$, $a_i = a_j$, $a_i \geq a_j$, or $a_i > a_j$ to determine the relative order of all elements in A
- We'll assume that all elements are distinct, and so note that the only comparison we need to make is $a_i \leq a_j$.
- This comparison gives us a yes or no answer

Decision Tree Model

- For a fixed deterministic comparison sort and input size n, a decision tree represents every possible sequence of comparison outcomes
- Each internal node specifies a comparison; the comparison may depend on earlier outcomes
- Each leaf specifies the output permutation determined by those outcomes

Decision Tree Model

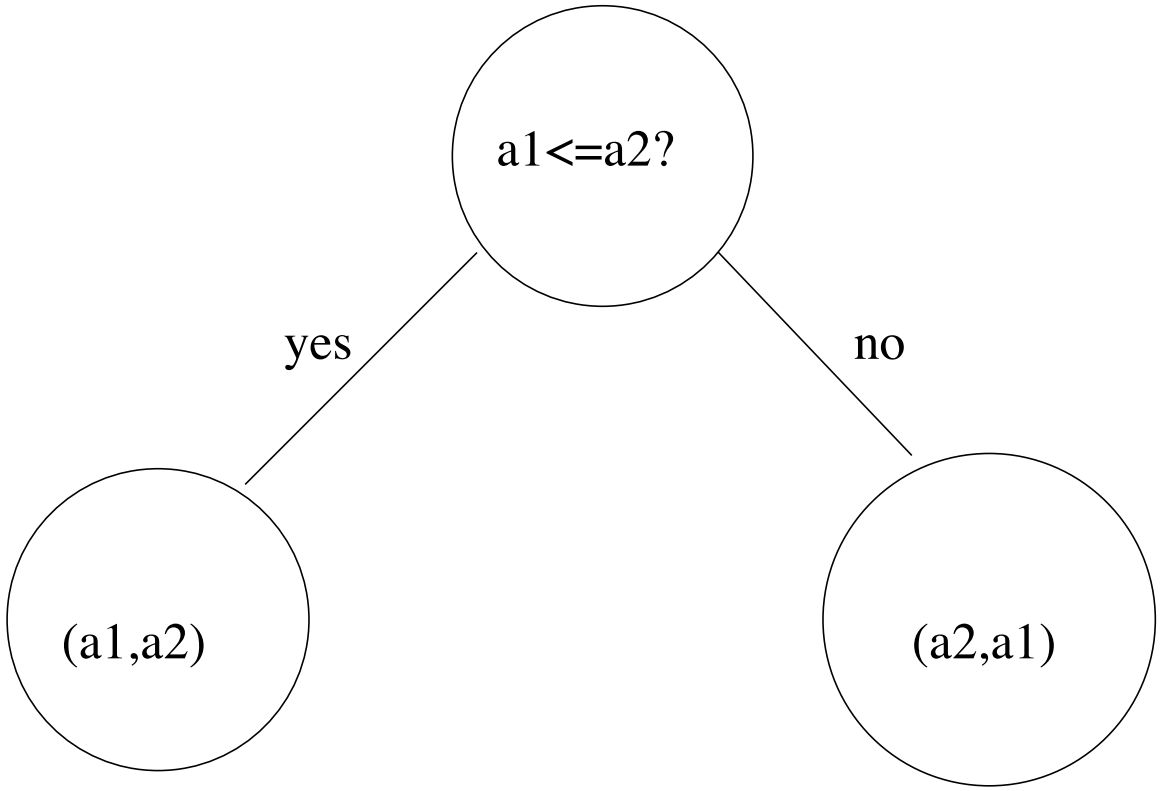
- The execution of the sorting algorithm corresponds to a path from the root node to a leaf node in the tree.
- We take the left child of the node if the comparison is $\leq$ and we take the right child if the comparison is $>$
- The internal nodes along this path give the comparisons made by the algorithm, and the leaf gives the output permutation

Leaf Nodes

- With distinct keys, a correct sorting algorithm must distinguish all $n!$ possible input orderings
- Thus there must be at least $n!$ leaf nodes
- The length of the longest path from the root node to a leaf in this tree is the worst-case number of comparisons
- The runtime is therefore at least the height of the tree

Example

- Consider the problem of sorting an array of size two: $A = (a_1, a_2)$
- Following is a decision tree for this problem.



In-Class Exercise

- Give a decision tree for sorting an array of size three: $A = (a_1, a_2, a_3)$
- What is the height? What is the number of leaf nodes?

Height of Decision Tree

- Q: What is the height of a binary tree with at least $n!$ leaf nodes?
- A: If h is the height, we know that $2^h \geq n!$
- Taking base-2 logarithms gives $h \geq \log_2(n!)$

Height of Decision Tree

- Q: What is $\log(n!)$?
- A: It is

$$\begin{aligned}
\log(n * (n-1) * \cdots * 1) &= \log n + \log(n-1) + \cdots + \log 1 \\
&\geq (n/2)\log(n/2) \\
&\geq (n/2)(\log n - \log 2) \\
&= \Omega(n \log n)
\end{aligned}$$

- Thus any decision tree for sorting n elements will have a height of $\Omega(n \log n)$

Take Away

- We've proven that every comparison-based sorting algorithm takes $\Omega(n \log n)$ time in the worst case
- This does *not* imply the same bound for every sorting algorithm
- With additional assumptions, non-comparison sorts can be asymptotically faster

Bucket Sort

- Bucket sort assumes that the input elements are drawn independently and uniformly from $[0, 1)$
- Basic idea is to divide the interval $[0, 1)$ into n equal size regions, or buckets
- We expect that a small number of elements in A will fall into each bucket
- To get the output, we can sort the numbers in each bucket and just output the sorted buckets in order

Bucket Sort

Assume $n = |A| \geq 1$ and every $x \in A$ satisfies $0 \leq x < 1$.

Bucket-Sort(A)

$n \leftarrow |A|$
$B[0 \ldots n-1] \leftarrow$ an array of empty lists
for each $x \in A$ **do**
  append x to $B[\lfloor nx \rfloor]$
for $i \leftarrow 0$ **to** $n-1$ **do**
  Insertion-Sort($B[i]$)
return $B[0] \parallel B[1] \parallel \cdots \parallel B[n-1]$

Bucket Sort

- Claim: If the input elements are independent and uniform on $[0, 1)$, then bucket sort takes expected time $O(n)$
- Let $T(n)$ be the run time of bucket sort on a list of size n
- Let B_i be the random variable giving the number of elements in bucket $B[i]$
- Then $T(n) = O\left(n + \sum_{i=0}^{n-1} B_i^2\right)$

Analysis

- For some constant $C > 0$, $T(n) \leq Cn + C \sum_{i=0}^{n-1} B_i^2$
- Taking expectations gives

$$E(T(n)) \leq Cn + C\, E\left(\sum_{i=0}^{n-1} B_i^2\right)$$

$$= Cn + C \sum_{i=0}^{n-1} E(B_i^2) \qquad \text{LOE}$$

- Linearity of expectation does not require the B_i to be independent

Analysis

- We claim that $E(B_i^2) = 2 - 1/n$
- To prove this, we define indicator random variables: $X_{ij} = 1$ if $A[j]$ falls in bucket i and 0 otherwise (defined for all i, $0 \leq i \leq n-1$ and j, $1 \leq j \leq n$)
- Thus, $B_i = \sum_{j=1}^{n} X_{ij}$
- We can now compute $E(B_i^2)$ by expanding the square and regrouping terms

Analysis

$$
\begin{aligned}
E(B_i^2) &= E\left(\left(\sum_{j=1}^{n} X_{ij}\right)^2\right) \\
&= E\left(\sum_{j=1}^{n} X_{ij}^2 + 2\sum_{1\le j<k\le n} X_{ij}X_{ik}\right) \\
&= \sum_{j=1}^{n} E(X_{ij}^2) + 2\sum_{1\le j<k\le n} E(X_{ij}X_{ik}) \qquad \text{LOE}
\end{aligned}
$$

Analysis

- We can evaluate the two summations separately. X_{ij} is 1 with probability $1/n$ and 0 otherwise
- Thus $E(X_{ij}^2) = 1 * (1/n) + 0 * (1 - 1/n) = 1/n$
- For $j < k$, the random variables X_{ij} and X_{ik} are independent
- If X and Y are independent, then $E(XY) = E(X)E(Y)$
- Thus we have that

$$\begin{aligned} E(X_{ij}X_{ik}) &= E(X_{ij})E(X_{ik}) \\ &= (1/n)(1/n) \\ &= (1/n^2) \end{aligned}$$

Analysis

- Substituting these two expected values back into our main equation, we get:

$$
\begin{aligned}
E(B_i^2) &= \sum_{j=1}^{n} E(X_{ij}^2) + 2 \sum_{1 \le j < k \le n} E(X_{ij} X_{ik}) \\
&= \sum_{j=1}^{n} \frac{1}{n} + 2 \sum_{1 \le j < k \le n} \frac{1}{n^2} \\
&= n \left(\frac{1}{n} \right) + 2 \binom{n}{2} \frac{1}{n^2} \\
&= 1 + \frac{n-1}{n} = 2 - \frac{1}{n}
\end{aligned}
$$

Analysis

- Recall that $E(T(n)) \leq Cn + C\sum_{i=0}^{n-1} E(B_i^2)$
- Since $E(B_i^2) = 2 - 1/n$,

$$\begin{aligned} E(T(n)) &\leq Cn + C\sum_{i=0}^{n-1}(2 - 1/n) \\ &= Cn + C(2n - 1) \\ &= O(n) \end{aligned}$$

- Reading the input takes $\Omega(n)$ time, so bucket sort has expected running time $\Theta(n)$ under the stated assumptions