Outline
- Quicksort Wrapup
- Randomized Quicksort
- Intro to Probability
- Birthday Paradox

Quicksort

//PRE: A[p..r] is the array to be sorted, p=1 and r <= size of A, A[r] is the pivot element
//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

//PRE: A[p..r] is the array to be partitioned, p>=1 and r <= size of A, A[r] is the pivot element
//POST: Let A’ be the array A after the function is run. Then A’[p..res-1] are <= A[r], A’[res] = A[r], and all elements in A’[res+1..r] are > A[r]
Partition (A, p, r){
  x = A[r];
  i = p-1;
  for (j=p; j<=r-1; j++){
    if (A[j]<=x){
      i++;
      exchange A[i] and A[j];
    }
  }
  exchange A[i+1] and A[r];
  return i+1;
}

Correctness

Basic idea: The array is partitioned into four regions, x is the pivot

- Region 1: Region that is less than or equal to x (between p and i)
- Region 2: Region that is greater than x (between i+1 and j-1)
- Region 3: Unprocessed region (between j and r-1)
- Region 4: Region that contains x only (r)

Region 1 and 2 are growing and Region 3 is shrinking

Loop Invariant

At the beginning of each iteration of the for loop, for any index k:

1. If p ≤ k ≤ i then A[k] ≤ x
2. If i + 1 ≤ k ≤ j - 1 then A[k] > x
3. If k = r then A[k] = x
In Class Exercise

- Show Initialization for this loop invariant
- Show Termination for this loop invariant
- Show Maintenance for this loop invariant:
  - Show Maintenance when \( A[j] > x \)
  - Show Maintenance when \( A[j] \leq x \)

Worst Case

- In the worst case, the partition always splits the original list into a singleton element and the remaining list
- Then we have the recurrence \( T(n) = T(n-1) + T(1) + \Theta(n) \), which is the same as \( T(n) = T(n-1) + \Theta(n) \)
- The solution to this recurrence is \( T(n) = \Theta(n^2) \). Why?

Analysis

- The function Partition takes \( O(n) \) time, where \( n = p - r \).
  Why?
- Q: What is the runtime of Quicksort?
- A: It depends on the size of the two lists in the recursive calls

Average Case Intuition

- Even if the recurrence tree is somewhat unbalanced, Quick-sort does well
- Imagine we always have a 9-to-1 split
- Then we get the recurrence \( T(n) \leq T(9n/10) + T(n/10) + cn \)
- Solving this recurrence (with recursion tree and induction) gives \( T(n) = \Theta(n \log n) \)

Best Case

- In the best case, the partition always splits the original list into two lists of half the size
- Then we have the recurrence \( T(n) = 2T(n/2) + \Theta(n) \)
- This is the same recurrence as for mergesort and its solution is \( T(n) = O(n \log n) \)

Wrap Up

- Take away: Both the worst case, best case, and average case analysis of algorithms can be important.
- You will have a hw problem on the “average case intuition” for deterministic quicksort
- (Note: A solution to the in-class exercise is on page 147 of the text)
We’d like to ensure that we get reasonably good splits reasonably quickly. We’d like an algorithm which is expected to perform well no matter what sort of input it gets.

Q: How do we ensure that we “usually” get good splits? How can we ensure this even for worst case inputs?

A: We use randomization.

R-Partition

//PRE: A[p..r] is the array to be partitioned, p>1 and r <= size of A
//POST: Let A’ be the array A after the function is run. Then
// A’[p..r] contains the same elements as A[p..r]. Further,
// all elements in A’[p..res-1] are <= A[i], A’[res] = A[i],
// and all elements in A’[res+1..r] are > A[i], where i is
// a random number between $p$ and $r$.
R-Partition (A,p,r){
   i = Random(p,r);
   exchange A[r] and A[i];
   return Partition(A,p,r);
}

Randomized Quicksort

//PRE: A is the array to be sorted, p>1, and r <= size of A
//POST: A[p..r] is in sorted order
R-Quicksort (A,p,r){
   if (p<r){
      q = R-Partition (A,p,r);
      R-Quicksort (A,p,q-1);
      R-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 is a variable that takes on one of several values, each with some probability. (Example: if $X$ is the outcome of the role of a die, $X$ is a random variable)
- The expected value of a random variable, $X$ is defined as:
  \[
  E(X) = \sum_x x \cdot P(X = x)
  \]
  (Example if $X$ is the outcome of the role of a three sided die,
  \[
  E(X) = 1 \cdot (1/3) + 2 \cdot (1/3) + 3 \cdot (1/3) = 2
  \]
- Two events $A$ and $B$ are mutually exclusive if $A \cap B$ is the empty set. (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 $y$, $P(X = x \text{ and } Y = y) = P(X = x)P(Y = y)$ (Example: let $X$ be the outcome of the first role of a die, and $Y$ be the outcome of the second role of the die. Then $X$ and $Y$ are independent.)
Probability Definitions

- An Indicator Random Variable associated with event \( A \) is defined as:
  - \( I(A) = 1 \) if \( A \) occurs
  - \( I(A) = 0 \) if \( A \) does not occur
- Example: Let \( A \) be the event that the role of a die comes up 2. Then \( I(A) \) is 1 if the die comes up 2 and 0 otherwise.

Example

- Indicator Random Variables and Linearity of Expectation used together are a very powerful tool
- 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)

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 role of three-sided die
  - Then
    \[
    E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = 2n
    \]

“Birthday Paradox”

- Assume there are \( k \) people in a room, and \( n \) days in a year
  - Assume that each of these \( k \) people is born on a day chosen 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 \( \leq i < j \leq k \), 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 \( i, j \),
    \[
    E(X_{i,j}) = P(\text{person } i \text{ and } j \text{ have same birthday}) = \frac{1}{n}
    \]
Let $X$ be a random variable giving the number of pairs of people with the same birthday.

We want $E(X)$.

The $X = \sum_{i=1}^{k} \sum_{j=i+1}^{k} X_{i,j}$

So $E(X) = E(\sum_{i=1}^{k} \sum_{j=i+1}^{k} X_{i,j})$

The second step follows by Linearity of Expectation.

Thus, if $k(k-1) \geq 2n$, expected number of pairs of people with same birthday is at least 1.

Thus if have at least $\sqrt{2n} + 1$ people in the room, can expect to have at least two with same birthday.

For $n = 365$, if $k = 28$, expected number of pairs with same birthday is 1.04.