1. Repeat until you have completed 10-20 experiments.

   Flip a coin until it comes up heads and then record the number of coin flips it took to get heads.

2. Compute the average number of flips it took to get heads.

Randomization

Algorithmic design patterns.

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Randomization.

in practice, access to a pseudo-random number generator

Randomization. Allow fair coin flip in unit time.

Why randomize? Can lead to simplest, fastest, or only known algorithm for a particular problem.

Ex. Symmetry breaking protocols, graph algorithms, quicksort, hashing, load balancing, Monte Carlo integration, cryptography.

Selection

Given : multi-set of n data values $S= \{a_1, a_2, a_3, \dots , a_n\}$

Position: p, $1 \leq p \leq n$

Find the value that would be in position p if the numbers were arranged in sorted order (from smallest to largest).

p=1: Find the minimum value.
p=n: Find the maximum value.
When n is odd and p= (n+1)/2: Find the median value.
When n is even:
the median m is the average of the values in positions n/2 and n/2+1.

Example

{12, 52, 94, 36, 42, 11, 98}

Sorted order:
11, 12, 36, 42, 52, 94, 98

p=1: 11 (min)
p=4: 42 (median)
p=6: 94

It takes $O(n \log_2 n)$ time to sort under the comparison model (that is, with no bit twiddling allowed as per radix sort).

Can we find the pth element faster than this?

Randomized Algorithm

```
Select( multi-set S,  int p)

Choose an element pivot from S (at random).

Partition S into three multi-sets:
    $S^-$  : elements less than pivot.
    $S^=$  : elements equal to the pivot.
    $S^+$  : elements greater than the pivot.
If  $|S^-| \geq p$ then return ( Select($S^-$ , p ))  // Element p less than pivot.

If  $|S^-| + |S^=| \geq p$ return (pivot) // Element p equal to pivot.

num_before= $|S^-| + |S^=|$

return( Select($S^+$ , p- num_before)) // Element p in $S^+$
```

Implementation of Partitioning

Multi-sets could be represented as linked lists.

Partitioning can be done in-place in an array as
per quickSort.

In both cases, the partitioning time is $O(n)$ where n is the size of the multi-set S.

```
// partition A[left] to A[right] = pivot
private static int partition(int [] A,
                                   int left, int right) {
int i = left; int j = right-1;

while (true)
{
    while (A[i]  < A[right]) {i++;}
    while (j > left && A[right] < A[j]) {j--};

    if (i >= j) break;
    swap(A, i, j); i++; j--;
}
swap(A, i, right); // Put pivot element into place
return i;
}
```

Code partitions when values are all distinct. How can you modify it to work with repeated values?

What is the expected time for this algorithm?

n= |S|

A "good" pivot partitions so that S^- and S^+ both have size at least n/4.

Given distinct values, how many values are good pivots?

What is the expected time that it takes to choose a good pivot if pivots are chosen at random?

How many steps can we expect the algorithm to take?

Initial phase is phase 0.
The algorithm is in phase j if the current size of S is s and

$n(\frac{3}{4})^{j+1} < s \le n(\frac{3}{4})^{j}$

Since the expected number of trials before we get a good pivot is two, we expect to stay in phase j for at most 2 trials.

Partitioning time: c n
Upper bound on expected time:
$2cn + 2c(\frac{3}{4})n + 2c(\frac{3}{4})^2 n + 2c(\frac{3}{4})^3 n + \ldots$

Taking limits as n goes to infinity:
$= 2cn\,[1 + (\frac{3}{4}) + (\frac{3}{4})^2 + (\frac{3}{4})^3 n + \ldots]$

Geometric sums

$[1 + (\frac{3}{4}) + (\frac{3}{4})^2 + (\frac{3}{4})^3 + \dots]$

Generic form:

$1 + r + r^2 + r^3 + \dots$

The value of this infinite sum is equal to:

$$\lim_{n\to\infty} (1 + r + r^2 + \dots + rn)$$

Multiply:

$(1 + r + r^2 + \dots + rn) \quad (1 - r) = (1 - r^{n+1})$

So the value we want is:

$\lim_{n\to\infty} (1 + r + r^2 + \dots + rn) = \lim_{n\to\infty} (1 - r^{n+1})/(1-r) = 1/(1-r)$