INFO 6205 – Algorithms and Data
Assignment 5 Solutions

Student Name: _____
Professor: Nik Bear Brown

Q1 (5 Points)

A mobile game randomly and uniformly awards a special coin for completing each level. There are *n* different types of coins. Assuming all levels are equally likely to award each coin, how many levels must you complete before you expect to have ≥ 1 coin of each type?

Solution:

This is identical to the "coupon collector" problem in the slides and book.

The expected number of steps is Θ(n log n).

If j is the number of coins we have and there are n different types of coins; then our expectation of getting a coin with don't have is (n-j)/n ⇒ expected waiting time = n/(n-j)
Therefore

$$E[X]=\sum_{j=0}^{n-1} E[X_j] =\sum_{j=0}^{n-1} \frac{n}{n-j}=n\sum_{i=1}^{n} \frac{1}{i}=nH(n)$$

Note that H(n) is the harmonic series https://en.wikipedia.org/wiki/Harmonic_series_(mathematics)

Where

ln(n+1) < H(n) < 1 + ln n

Q2 (10 Points) Short answer:

PushPush is a 2-D pushing-blocks game with the following rules:

a. Initially, the planar square grid is filled with some unit-square blocks (each occupying a cell of the grid) and the robot placed in one cell of the grid.
b. The robot can move to any adjacent free square (without a block).
c. The robot can push any adjacent block, and that block slides in that direction to the maximal extent possible, i.e., until the block is against another block.
d. Pushing means moving away (farther) from the robot (not pulling closer to robot)
e. The goal is to get the robot to a particular position.

A solution to PushPush is specified as a list of the following moves and x,y coordinates:

```
MoveRobot(x,y) # moves the robot from its current position to the position
x,y
PushBlock(x,y) # pushes a block from its current position to the position x,y
CheckGoal(robot) # takes the position of the robot and checks whether it is
at the goal
```

Is the Push Push problem in NP? If so prove it.

Solution:

2 points for yes.
2 points for a polynomial-time certificate.
1 point for an analysis of the running-time

2 points for a polynomial-time certificate.

polynomial-time certificate

Loop through the list with m moves:
If any move violates a rule return no
If we finish the moves list without violating any rules and the robot and goal are at the same position return yes

1 point for an analysis of the running-time.

Note: The analysis of the running-time doesn't need to be this detailed, but should say something about loop through the list with m moves, and the cost of comparing x and y positions on a grid.

The moves are provided as a list with m moves
`MoveRobot(x,y) #` The robot can move to any adjacent free square (without a block).
We can check whether the move `x,y to x',y' is adjacent in constant time either x or y must be one away from x',y' for this to be adjacent`

We can check whether something exists at x',y' in constant time be looking up the positions at that point in the grid
PushBlock(x,y)
We can check whether the robot is adjacent (and able to push in constant time)
A push will increment or decrement either x or y. We can check whether something exists at each point in the path in constant time.
CheckGoal(robot) We can check whether the robot and goal are at the same x and y in constant time

In the worst case our running time is m* (the average length of a push) and in the best m* O(1) which are both within polynomial-time.

Q3 (5 Points) What makes a configuration "stable" in the Hopfield Neural Network algorithm?

Solution:

A configuration is stable if all nodes are satisfied.

The above the fine for full credit but adding the below doesn't hurt.

Def. With respect to a configuration S, edge e = (u, v) is good if $w_e s_u s_v < 0$. That is, if $w_e < 0$ then $s_u = s_v$; if $w_e > 0$, $s_u \neq s_v$.

Def. With respect to a configuration S, a node u is satisfied if the weight of incident good edges ≥ weight of incident bad edges.

$$\sum_{v: e=(u,v) \in E} w_e s_u s_v \leq 0$$

Q4 (5 Points) Does the payoff matrix below have any Nash equilibria? Why or why not?

	Cooperate	Defect
Cooperate	2,1	1,2
Defect	1,2	2,1

Solution:

To find the Nash equilibria, we examine each action profile in turn.
(Cooperate,Cooperate)
Player 2 (Green) can increase its payoff from 1 to 2 by choosing the action Defect rather than the action Cooperate. Thus this action profile is not a Nash equilibrium.

(Cooperate,Defect)
Player 1 (Red) can increase its payoff from 1 to 2 by choosing the action Defect rather than the action Cooperate. Thus this action profile is not a Nash equilibrium.

(Defect,Cooperate)
Player 1 can increase its payoff from 1 to 2 by choosing the action Cooperate rather than the action Defect. Thus this action profile is not a Nash equilibrium.

(Defect,Defect)
Player 2 can increase its payoff from 1 to 2 by choosing the action Cooperate rather than the action Defect. Thus this action profile is not a Nash equilibrium.

We conclude that the game has no Nash equilibrium!

Q5 (5 Points) Coin is heads with probability p and tails with probability 1-p. How many independent flips X until first heads? (Express the expectation as a function of p.)

Solution:

$$E[X]=\sum_{j=0}^{\infty} j\cdot\Pr[X=j]=\sum_{j=0}^{\infty} j(1-p)^{j-1}p=\frac{p}{1-p}\sum_{j=0}^{\infty} j(1-p)^{j}=\frac{p}{1-p}\cdot\frac{1-p}{p^2}=\frac{1}{p}$$

For example, a fair coin with p=0.5 we expect to get a heads in 1/0.5 or 2 flips.

Q6 (5 Points) Give an argument or proof for the questions below:

(A 2 points) Can always convert a Las Vegas algorithm into a Monte Carlo algorithm?

(B 3 points) Can always convert a Monte Carlo algorithm into a Las Vegas algorithm?

Solution:

(A 2 points) Can always convert a Las Vegas algorithm into a Monte Carlo algorithm?

Las Vegas algorithm. Guaranteed to find correct answer, likely to run in poly-time. (1 point)

Yes, stop algorithm after a certain point. (1 point)

(B 3 points) Can always convert a Monte Carlo algorithm into a Las Vegas algorithm?

A Monte Carlo algorithm is guaranteed to run in poly-time, likely to find correct answer. (1 point)

No, consider an example such as randomized 3-SAT. There may not be an answer, so running it forever may not find an answer.

Q7 (5 Points)
Euclid's algorithm, is an efficient method for computing the greatest common divisor (GCD) of two numbers, the largest number that divides both of them without leaving a remainder.

The algorithm in pseudocode is below:

```
int euclids_algorithm(int m, int n){
  if(n == 0)
     return m;
   else
     return euclids_algorithm(n, m % n);}
```

(A 3 points) Write a recurrence relation for Euclid's algorithm.

T(n) = T(n/2) + c (we know f(n) is c because checking `n == 0 or m % n` can be done in constant time.)

Note: For FULL CREDIT any recurrence relation of the form T(n) = T(n/b) + c where b is some estimate b > 1 is OK.

So T(n) = T(n/2) + c OR T(n) = T(n/b) + c OR T(n) ≤ T(n/2) + O(1) is given full credit.

Note as to why n/2:

For any two integers m and n such that n >= m, it is always true that
n mod m <n/2.

If m > n / 2, then 1 =< n / m < 2, and so the floor [n / m] = 1, which means that n mod m =n - m < n - n/2=n/2.

(B 2 points) Give an expression for the runtime T(n) if your Euclid's algorithm recurrence can be solved with the Master Theorem.

Solution:

$\Theta(\log n)$ or $\Theta(\log_b n)$

Euclid's Algorithm $T(n) \leq T(n/2) + O(1)$

$T(n) \leq T(n/b) + O(1)$

$T(n) = aT(n/b) + f(n)$ where a >= 1 and b > 1

A=1, b=1 f(n)=c

There are following three cases:
If $f(n) = \Theta(n^c)$ where $c < \log_b a$ then $T(n) = \Theta(n^{\log_b a})$
if $f(n) = \Theta(n^c)$ where $c = \log_b a$ then $T(n) = \Theta(n^c \log^{p+1} n)$
If $f(n) = \Theta(n^c)$ where $c > \log_b a$ then $T(n) = \Theta(f(n))$

$\log_2(1) = 0$ $c = n^0 * c$
so this is case B (they are equal)
$T(n) = \Theta(n^c \log^{p+1} n)) = \Theta(n^0 \log^{0+1} n) = = \Theta(\log n)$

For the recurrence
$T(n) = T(n/b) + c$
$\text{Log}_b(1) = 0$ $c = n^0 * c$
so this is case B (they are equal)
$T(n) = \Theta(n^c \log_b{}^{p+1} n)) = \Theta(n^0 \log_b{}^{0+1} n) = \Theta(\log_b n)$

Q8 (10 Points)

Consider a random algorithm to find a maximum Independent Set in a graph. That is, a graph *G* = (*V, E*) with a node having the value 0 or 1 and an edge joining pairs of nodes. Two nodes are in conflict if they belong to the same set (i.e. a 0 node connecting to a 0 node or a 1 node connecting to a 1 node). We know finding the maximum-size Independent Set *S,* is NP-Complete so we want to find as large an Independent Set *S* as we can using a randomized algorithm. We will suppose for purposes of this question that each node in has exactly *d* neighbors in the graph *G.* (That is, each node is in conflict with exactly *d* other nodes.)

Consider the following randomized algorithm to find a large Independent Set in this special graph.

Each node P_i *independently picks a random value* x_i*; it sets* x_i *to* 1 *with probability p=0.5 and sets* x_i *to* 0 *with probability* 1 – *p=0.5. It then decides to enter the set S if and only if it chooses the value* 1, *and each of the d nodes with which it is connected chooses the value* 0.

Give a formula for the expected size of *S* when *p* is set to 0.5.

Solution:

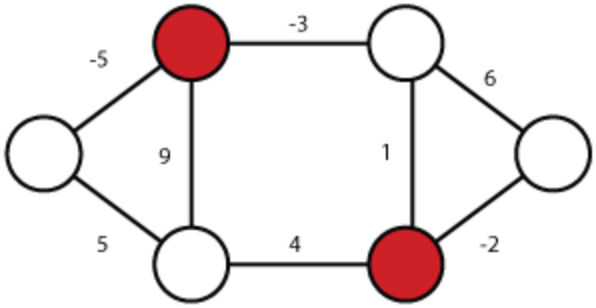
Assume that using the described protocol, we get a set *S* that is not conflict free. Then there must be 2 processes Pi and Pj in the set *S* if that both picked the value 1 and are going to want to share the same resource. But this contradicts the way our protocol was implemented, since we selected processes that, picked the value 1 and whose set of conflicting processes all picked the value 0. Thus if Pi and Pj both picked the value 1, neither of them would be selected and so the resulting set *S* is conflict free. For each process Pi, the probability that it is selected depends on the fact that Pi picks the value 1 and all its *d* conflicting processes pick the value 0. Thus P[Pi selected] $=\frac{1}{2}*\left(\frac{1}{2}\right)^{d}$ or $=\left(\frac{1}{2}\right)^{d+1}$ And since there are n nodes that pick values independently, the expected size of the set *S* is $n*\left(\frac{1}{2}\right)^{d+1}$

Note that we use “linearity of expectation” to calculate the expected size of the set *S*

10 points for P[Pi selected] $=\frac{1}{2}*\left(\frac{1}{2}\right)^{d}$ or $\left(\frac{1}{2}\right)^{d+1}$ (Partial credit for “close” arguments)

5 points for the expected size of the set *S* is $n*\left(\frac{1}{2}\right)^{d+1}$

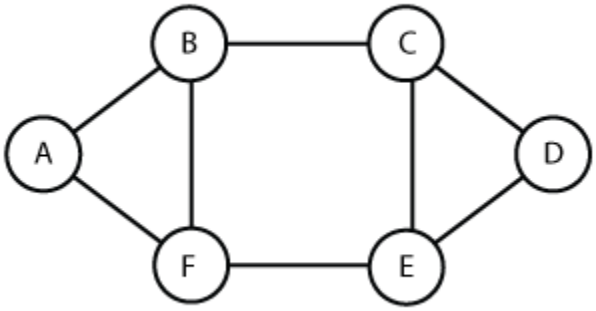
Q9 (5 Points)



Use the *Hopfield Neural Network State-flipping algorithm* to find a stable configuration on the weighted undirected network graph above. Show your work. Draw the *final* stable configuration.

Solution:

I'm using the labeling scheme A-F clockwise from far left as below and considering red/dark -1 and white/light 1.



Def. With respect to a configuration S, edge e = (u, v) is good if $w_e s_u s_v < 0$. That is, if $w_e < 0$ then $s_u = s_v$; if $w_e > 0$, $s_u \neq s_v$.

Def. With respect to a configuration S, a node u is satisfied if the weight of incident good edges $\geq$ weight of incident bad edges.

Def. A configuration is stable if all nodes are satisfied.

```
Hopfield-Flip(G, w) {
   S ← arbitrary configuration
   while (current configuration is not stable) {
      u ← unsatisfied node
      s_u = -s_u
   }
```

```
return S }
```

A: 5 + 5 = 10 > 0, flip
A: -5 - 5 = -10 <= 0 don't flip

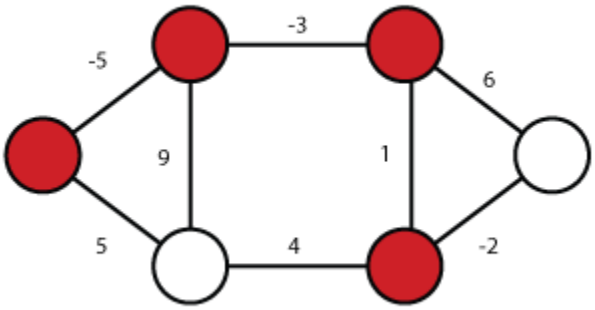
B: -5 – 9 +3 = -11 <= 0 don't flip
C: 3 + 6 -1 = 8 > 0, flip
C: -3 + -6 + 1 = -8 <= 0 don't flip

B, D, and E are incident to C (B is the only previously checked)
B: -5 – 9 -3 = -14 <= 0 don't flip

D: -6 + 2 = -4 <= 0 don't flip
E: 2 + 1 - 4 = -1 <= 0 don't flip
F: -5 - 9 - 4 = -18 <= 0 don't flip

All nodes satisfied

Final configuration



Q10 (5 Points)

(A 1 points) Sort the list of integers below using Quicksort. Show your work.
(22, 13, 29, 1, 2, 31, 33, 11)

(B 2 points) Write a worst case and average case recurrence relation for Quicksort.

(C 2 points) Give an expression for the runtime T(n) if your worst case and average case Quicksort recurrence relations can be solved with the Master Theorem.

Solution:

Choose pivot as n-1 item. Use that to partition the array in-place.

```
Quicksort(A as array, low as int, high as int)
   if (low < high)
      pivot_location = Partition(A,low,high)
   Quicksort(A,low, pivot_location - 1)
   Quicksort(A, pivot_location + 1, high)

Partition(A as array, low as int, high as int)
    pivot = A[low]
    leftwall = low

    for i = low + 1 to high
       if (A[i] < pivot) then
          leftwall = leftwall + 1
          swap(A[i], A[leftwall])

    swap(A[low],A[leftwall])

   return (leftwall)
```

Assume the Pivot is last element of array
Assume we are swapping in-place around the pivot

Note correctly implementing a couple of steps of Quicksort correctly is fine for full credit

Start: (22, 13, 29, 1, 2, 31, 33, 11)

Next: QS(22, 13, 29, 1, 2, 31, 33, 11)
Next: QS(1,13,2)+QS(11)+QS(22,31,33,29)
Next: QS(1,2)+QS(13)+QS(11)+QS(22)+QS(29,33,31)
Next: QS(1)+QS(2)+QS(13)+QS(11)+QS(22)+QS(29,31)+QS(33)
Next: QS(1)+QS(2)+QS(13)+QS(11)+QS(22)+QS(29)+QS(31)+QS(33)

Sorted: 1, 2, 13, 11, 22, 29, 31, 33
Worst case performance (big-O): $O(n^2)$

Average case performance (big-O): O(n log n)

Worst case Recurrence relation: T(n) = T(0) + T(n-1) + O(n) = T(n-1) + O(n)

Average recurrence relation: T(n) = 2 T(n/2) + O(n)

By the Master Theorem, there are following three cases:
If $f(n) = \Theta(n^c)$ where $c < \log_b a$ then $T(n) = \Theta(n^{\log_b a})$
if $f(n) = \Theta(n^c)$ where $c = \log_b a$ then $T(n) = \Theta(n^c \log^{p+1} n)$

If $f(n) = \Theta(n^c)$ where $c > \log_b a$ then $T(n) = \Theta(f(n))$

$\log_2(2) = 1$ $c = n^1$
so this is case B (they are equal)
then $T(n) = \Theta(n^{\log_b a} \log^{p+1} n) = \Theta(n^1 \log^{0+1} n) = \Theta(n \log n)$

Q11 (10 Points)

Ebay is considering a very simple online auction system that works as follows. For an auction, if there are *n bidders*; agent *i* has a bid b_i, which is a positive natural number. We will assume that all bids b_i are distinct from one another. The bidders appear in an order chosen uniformly at random, each proposes its bid b_i in turn, and at all times the system maintains a variable b^* equal to the highest bid seen so far. (Initially b^* is set to 0.) In essence, this system collects all n bids then presents them uniformly at random.

What is the expected number of times that b^* is updated when this process is executed, as a function of the parameters in the problem?

Example. Suppose $b_1 = 22$, $b_2 = 33$, and $b_3 = 11$, and the bidders arrive in the order 1, 3, 2. Then b^* is updated for 1 (b_0 is initially 0 so first bid is always updated) and 2, but not for 3.

Solution:

Lots of partial credit for reasonable approaches is given, but …. $E[X_i] = 1/i$.

Let *X* be a random variable equal to the number of times that b* is updated. We write $X = X_1 + \cdots + X_n$, $where\ X_1 = 1$ if the i[th] bid in order cause *b** to be updated, and $X_i = 0$ otherwise.

So $X_i = 1$ if and only if, focusing just on the sequence of the first *i* bids, the largest one comes at the end. But the largest value among the first *i* bids is equally likely to be anywhere, and hence $E[X_i] = 1/i$.

FULL CREDIT for getting $E[X_i] = 1/i$.

Alternately, the number of permutations in which the number at position *i* is larger than any of the numbers before it can be computed as follows. We can choose the first numbers in $\binom{n}{i}$ ways, put the largest in position i, order the remainder in (i—1)! ways, and order the subsequent (n — i) numbers in (n — i)! ways. Multiplying this together, we have $\binom{n}{i}(i-1)!\,(n-i)! = \frac{n!}{i}$. $Dividing\ by\ n!, we\ get\ EX_i = \frac{1}{i}$.

For FULL CREDIT for getting $EX_i = \frac{1}{i}$.

Now, by linearity of expectation, we have $EX = \sum_{i=1}^{Ti} EX_i = \sum_{i=1}^{Ti} \frac{1}{i} = H_n = \Theta(logn)$.

Q12 (5 Points)

In a co-op, you develop an algorithm for a content delivery network like YouTube or Miley.com. Suppose that in a typical minute, you get a k (e.g. a bazillion) content requests, and each needs to be served from one of your *n* servers. Your algorithm is randomly assign each job to a random server.

A. (1 point) What is the expected number of jobs per server?
B. (2 points) What is the probability that a server gets twice the average load? That is, 2 times the expected number of jobs? (A bound is acceptable)
C. (2 points) What is the probability that a server gets no load? That is, no jobs? (A bound is acceptable)

Solution:

A. (1 point) What is the expected number of jobs per server?

k jobs over your *n* servers so k/n

B. (2 points) What is the probability that a server gets twice the average load? That is, 2 times the expected number of jobs?

Chernoff Bounds (above mean)

$$\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}} \right)^{\mu}$$

μ= k/n
(1+ δ) = 2 … (twice the average or expected load) so δ = 1

Pr[X>= twice the expected load] =< $[e^{-1}/2^2]^{k/n}$
= $[e^{-}/4]^{k/n}$

Note that $e^{-}/4$ is less than 1 so this will be exponentially decreasing. Let x = k/n we can plot this WolframAlpha - (e/4)^x http://wolfr.am/6ouN7vfH via @wolfram_alpha

Note that even though one will get a varying number of processes in the real world, one usually knows the number of servers on one's cluster. So for a fixed n, say n = 100 we can graph this as well. (i.e. (e/4)^(k/100))

WolframAlpha - (e/4)^(k/100) http://wolfr.am/6ovcXfmr via @wolfram_alpha

C. (2 points) What is the probability that a server gets no load? That is, no jobs?

Use with μ= k/n and δ = 1

Chernoff Bounds (below mean)

$$\Pr[X < (1-\delta)\mu] \leq e^{-\delta^2 \mu/2}$$

or

$$\Pr[X < (1-\delta)\mu] \leq e^{-\delta^2 \mu/2}$$

-(1^2) equals -1 k/n divided by 2 equals k/2n

So

$$\Pr[X < (1-\delta)\mu] \leq e^{-k/2n}$$

Note calculating probabilities explicitly rather than using bounds is fine, so what is the probability that a server gets no load? Pr[server gets a job] = 1/n. Pr[server doesn't get a job] = 1-1/n. Hence the probability comes out to be $(1-1/n)^k$

Note that even though one will get a varying number of processes in the real world, one usually knows the number of servers on one's cluster. So for a fixed n, say n = 100 we can graph this as well.

Q13 (5 Points) Does the state-flipping algorithm always terminate? If so why?

Solution:

Proof. Consider measure of progress $\Phi(S) = \Sigma_{e \text{ good}} |w_e|$.

- Clearly $0 \leq \Phi(S) \leq W$.
- We show $\Phi(S)$ increases by at least 1 after each flip.
  When u flips state:
  - all good edges incident to u become bad
  - all bad edges incident to u become good
  - all other edges remain the same

This is directly from Kleinberg & Tardos slides:

Claim. State-flipping algorithm terminates with a stable configuration after at most $W = \Sigma_e |w_e|$ iterations.

Pf. Consider measure of progress $\Phi(S) = \Sigma_{e \text{ good}} |w_e|$.

- Clearly $0 \le \Phi(S) \le W$.
- We show $\Phi(S)$ increases by at least 1 after each flip.
  When u flips state:
  - all good edges incident to u become bad
  - all bad edges incident to u become good
  - all other edges remain the same

$$\Phi(S') = \Phi(S) - \sum_{e:\, e=(u,v) \in E,\ e \text{ is bad}} |w_e| + \sum_{e:\, e=(u,v) \in E,\ e \text{ is good}} |w_e| \ge \Phi(S) + 1$$

u is unsatisfied

Q14 (5 Points) What is the probability of getting exactly no heads after flipping four coins?

Solution:

1/16

If one is head and 0 tails:
0 0 0 0
0 0 0 1
0 0 1 0
…
There are 2^4 or 16 combos
There is one event of getting exactly no heads after flipping four coins.
{0 0 0 0 }

So 1/16 is the probability of getting exactly no heads after flipping four coins.

We can also the formula for a discrete random variable based on a binomial distribution:

$$\binom{n}{k} p^k (1-p)^{n-k}$$

Q15 (10 Points)

Suppose you are using a randomized version of quicksort on a very large set of real numbers, and you would like to pick a pivot by sampling. Suppose you sample a subset uniformly at random (with replacement) and use that to estimate the value of your pivot. Use a Chernoff-bound to calculate your confidence in your approximate pivot estimate. You can choose your confidence level (typical is 90%, 95% or 99% confidence) the distribution of the numbers and your sample size.

Solution:

Use Sampling Theorem

$n \geq ((2 + \varepsilon)/\varepsilon^2) * \ln 2/\delta$

So pick a confidence of 0.90 and a confidence interval of (+/-5%)
$1-\delta = 0.90$ so $\delta = 0.10$

ε is our spread
Confidence interval of (+/-5%) so $\varepsilon = 0.05$

$n \geq ((2 + \varepsilon)/\varepsilon^2) * \ln 2/\delta$

Becomes

$n \geq ((2 + 0.05)/(0.05)^2) * \ln 2/0.10$ or

$$\frac{2 + 0.05}{0.05^2} \log\left(\frac{2}{0.1}\right)$$

or a sample 2457 numbers

see WolframAlpha - (2+.05)/(.05)^2*ln(2/.10) http://wolfr.am/5w_YnmpX via @wolfram_alpha

Or use a Chernoff bound

Note it's OK just to pick a confidence level, and your sample size and use a Chernoff bound above and below the mean. To understand these Chernoff bounds you need to understand there are two sources of uncertainty in this process. There's the probability that you obtain a "good" estimate. And there's the extent to which your estimate is "good". If you look at professional poll results that you'll find that they use phrases like, "This poll is accurate to within ±5%, 19 times out of 20." The ±5% is the range of the estimate and the 19 times out of 20 (95%) is the confidence in the estimate. Think of the coin flip bias check examples. If we wanted to check a 90% biased coin our range is very large ±40% so we need very few flips to get an estimate of high confidence. If we wanted to check whether a coin was slightly

biased, say ±2%, we would need a lot of flips to estimate this with high confidence. So you will receive full credit for just deciding a confidence level and an estimate range to determine the sample size by plugging values into an appropriate Chernoff bound equation.

See Chernoff bound equations below:

Chernoff bound (above mean)

Chernoff Bounds (above mean)

Theorem. Suppose $X_1, \ldots, X_n$ are independent 0-1 random variables. Let $X = X_1 + \ldots + X_n$. Then for any $\mu \geq E[X]$ and for any $\delta > 0$, we have

$$\Pr[X > (1+\delta)\mu] < \left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu}$$

sum of independent 0-1 random variables is tightly centered on the mean

Pf. We apply a number of simple transformations.

- For any $t > 0$,

$$\Pr[X > (1+\delta)\mu] = \Pr\left[e^{tX} > e^{t(1+\delta)\mu}\right] \leq e^{-t(1+\delta)\mu} \cdot E[e^{tX}]$$

$f(x) = e^{tx}$ is monotone in x

Markov's inequality: $\Pr[X > a] \leq E[X] / a$

- Now $E[e^{tX}] = E[e^{t\sum_i X_i}] = \prod_i E[e^{tX_i}]$

definition of X

independence

Chernoff bound (below mean)

Chernoff Bounds (below mean)

Theorem. Suppose $X_1, \ldots, X_n$ are independent 0-1 random variables. Let $X = X_1 + \ldots + X_n$. Then for any $\mu \leq E[X]$ and for any $0 < \delta < 1$, we have

$$\Pr[X < (1-\delta)\mu] < e^{-\delta^2 \mu / 2}$$

Pf idea. Similar.

Remark. Not quite symmetric since only makes sense to consider $\delta < 1$.

The answer below is Kleinberg and Tardos preferred approach to this problem. This is also accepted but not expected.

Suppose we want 95% confidence (an *ε-approximate median* of .05). Imagine dividing the set S into 20 quantiles Q_1 .. Q_{20} where Q_i, consists of all elements that have at least. .05(i — 1)*n* elements less than them, and at. least .05(20 — i)*n* elements greater than them. Choosing the sample S' is like throwing a set of numbers at random into bins labeled with Q_1 .. Q_{20}.

Suppose we choose |S'| = 40,000 and sample with replacement. Consider the event E that S' intersection with Qi is between 1900 and 2100 for each i. If E occurs, then the first nine quantiles contain at most 19, 900 elements of S', and the last nine quantiles do as well. Hence the median of S' will belong to Q_{10} union with Q_{11} and thus will be a (.05)-approximate median of S.

The probability that a given Q_i, contains more than 2100 elements can be computed using the Chernoff bound (above mean) with μ = 2000 and δ = .05; it is less than

Pr[X>2100]

$\Pr[X>(1+\delta)\mu] < [e^{\cdot\delta}/(1+\delta)^{(1+\delta)}]^{\mu}$

$\Pr[X>(1+\delta)\mu] < [e^{.05}/(1.05^{)(\ 1.05)}\ ]^{2000}$
(i.e. [*e*^.05/(1.05)^(1.05)]^2000

Pr[X>(1+ δ) μ] is Pr[X>(1.05)2000] is Pr[X>2100]

The probability that a given Q_i, contains fewer than 1900 elements can be computed using the Chernoff bound (below mean), with μ = 2000 and δ = .05; it is less than

Pr[X<1900]

$$\Pr[X < (1-\delta)\mu] \ \leq \ e^{-\delta^2\mu/2}$$

$\Pr[X<(1-\delta)\mu] < e^{-\delta^{\wedge}2\,\mu/2}$

$e^{-(.5)(\ .05)(\ .05)200}$ (i.e. [*e*^-(.5)(.05)(.05)2000)

We then apply the Union Bound over the 20 choices of I, to get the probability that E does not occur.

Q16 (10 Points)

Consider a server cluster with *p* processes and n servers (p > n). Your load balancing algorithm selects a server at random to place any new process, all equally likely.

A. (1 points) What is the expected number of process in each server?
B. (3 points) How big can the difference in server load be? Give an explicit probability.
C. (3 points) What is the likelihood that a server is k times an average server?
D. (3 points) What is the likelihood that a server is 1/k below an average server?

Solution:

A. (3 points) What is the expected number of process in each server?

p/n

B. (4 points) How big can the difference in server load be? Give an explicit probability.

P(a server gets a process) = 1/n

P(a server gets all processes) = $(1/n)^p$

C. (4 points) What is the likelihood that a server is k times an average server?

Average server load is p/n = μ
1+ δ = k therefore k-1 = δ

$\Pr[X>(1+\delta)\mu] < [e^{\cdot\delta}/(1+\delta)^{(1+\delta)}]^{\mu}$

$\Pr[X>k\mu] < [e^{\cdot\ k-1}/(1+k-1)^{(1+k-1)}]^{p/n}$

$\Pr[X>k\mu] < [e^{\cdot\ k-1}/(k)^{(k)}]^{p/n}$

D. (4 points) What is the likelihood that a server is 1/k below an average server?

Average server load is p/n = μ
1 δ = 1/k therefore 1-1/k = δ

$$\Pr[X < (1-\delta)\mu] \leq e^{-\delta^2\mu/2}$$

$\Pr[X<(1-\delta)\mu] < e^{-\delta\wedge 2\,\mu/2}$

$\Pr[X<\mu/k] < e^{-(1-1/k)\wedge 2\ p/2n}$