Final In-Class Exam (Solution Ideas)

(2:35 PM – 3:50 PM : 75 Minutes)

- This exam will account for either 15% or 30% of your overall grade depending on your relative performance in the midterm and the final. The higher of the two scores (midterm and final) will be worth 30% of your grade, and the lower one 15%.
- There are three (3) questions, worth 75 points in total. Please answer all of them in the spaces provided.
- There are 16 pages including three (3) blank pages and two (2) pages of appendices. Please use the blank pages if you need additional space for your answers.
- The exam is *open slides* and *open notes*.

GOOD LUCK!

Question	Pages	Score	Maximum
1. Parallel Prefix Sum	2–5		25
2. ϵ-Approximate Frequency	7–9		30
3. Matrix Rotation	11–13		20
Total			75

NAME: ____

Question 1. **[25 Points] Parallel Prefix Sum.** Given a sequence of n elements $\langle x_1, x_2, \ldots x_n \rangle$ drawn from a set S with a binary associative operator $\oplus$ (e.g., addition, multiplication, maximum, matrix product, union, etc.), the *prefix sum* problem asks one to compute a sequence of n partial sums $\langle s_1, s_2, \ldots s_n \rangle$ such that $s_i = x_1 \oplus x_2 \oplus \ldots x_i$ for $1 \leq i \leq n$. In lecture 26 we studied a parallel prefix sum algorithm with $\Theta(n)$ work and $\Theta\left(\log^2 n\right)$ span[1].

In this problem we will analyze another parallel prefix sum algorithm given in Figure 1.

Alt-Prefix-Sum($\langle x_1, x_2, \ldots, x_n \rangle$, $\oplus$)

(Input is a sequence of n elements $\langle x_1, x_2, \ldots, x_n \rangle$ and a binary associative operator $\oplus$. Output is a sequence $\langle s_1, s_2, \ldots, s_n \rangle$ with $s_i = x_1 \oplus x_2 \oplus \ldots \oplus x_i$, for $1 \leq i \leq n$. We assume $n = 2^k$ for some integer $k \geq 0$.)

if $n = 1$ ***then***

  $s_1 \leftarrow x_1$ {*the prefix sum of a single element is the element itself*}

else

  spawn $\left\langle s_1, s_2, \ldots, s_{\frac{n}{2}} \right\rangle \leftarrow$ Alt-Prefix-Sum$\left(\left\langle x_1, x_2, \ldots, x_{\frac{n}{2}} \right\rangle, \oplus \right)$

  $\left\{ \textit{sets } s_i = x_1 \oplus x_2 \oplus \ldots \oplus x_i \textit{ for } 1 \leq i \leq \frac{n}{2} \right\}$

  $\left\langle s_{\frac{n}{2}+1}, s_{\frac{n}{2}+2}, \ldots, s_n \right\rangle \leftarrow$ Alt-Prefix-Sum$\left(\left\langle x_{\frac{n}{2}+1}, x_{\frac{n}{2}+2}, \ldots, x_n \right\rangle, \oplus \right)$

  $\left\{ \textit{sets } s_{\frac{n}{2}+i} = x_{\frac{n}{2}+1} \oplus x_{\frac{n}{2}+2} \oplus \ldots \oplus x_{\frac{n}{2}+i} \textit{ for } 1 \leq i \leq \frac{n}{2} \right\}$

  sync

  parallel for $i \leftarrow 1$ ***to*** $\frac{n}{2}$ ***do***

    $s_{\frac{n}{2}+i} \leftarrow s_{\frac{n}{2}} \oplus s_{\frac{n}{2}+i}$ $\left\{ \textit{extends } s_{\frac{n}{2}+i} = x_{\frac{n}{2}+1} \oplus x_{\frac{n}{2}+2} \oplus \ldots \oplus x_{\frac{n}{2}+i} \right.$

    $\left. \textit{to } s_{\frac{n}{2}+i} = s_{\frac{n}{2}} \oplus x_{\frac{n}{2}+1} \oplus x_{\frac{n}{2}+2} \oplus \ldots \oplus x_{\frac{n}{2}+i} = x_1 \oplus x_2 \oplus \ldots \oplus x_{\frac{n}{2}+i} \right\}$

return $\langle s_1, s_2, \ldots, s_n \rangle$

Figure 1: An alternate parallel prefix sum algorithm.

[1] assuming the span of a ***parallel for*** loop with n iterations to be $\mathcal{O}(\log n + k)$, where k is the maximum span of a single iteration

1(a) [**7 Points**] Write down a recurrence relation describing the work done (i.e., T_1) by ALT-PREFIX-SUM, and solve it.

Solution. Let $T_1(n)$ be the work done by the algorithm for a sequence of n elements.

Clearly, the algorithm performs $\Theta(1)$ work (in line 2) when $n = 1$. Otherwise (i.e., if $n > 1$), it makes two recursive calls to itself (in lines 4 and 5) each for a sequence of length $\frac{n}{2}$, and then executes a ***for*** loop (in lines 6 and 7) that iterates $\frac{n}{2}$ times. Hence, T_1 can be described using the following recurrence relation.

$$T_1(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T_1\left(\frac{n}{2}\right) + \Theta(n) & \text{otherwise.} \end{cases}$$

Using Master Theorem (case 2), we obtain: $T_1(n) = \Theta(n \log n)$.

1(*b*) [**7 Points**] Write down a recurrence relation describing the span (i.e., T_∞) of ALT-PREFIX-SUM, and solve it.

Solution. Let $T_\infty(n)$ be the span of the algorithm for an input of size n.

Clearly, the algorithm has $\Theta\left(1\right)$ span (in line 2) for $n = 1$. Otherwise (i.e., if $n > 1$), it makes two parallel recursive calls to itself (in lines 4 and 5) each for a sequence of length $\frac{n}{2}$, and then executes a ***parallel for*** loop (in lines 6 and 7) that iterates $\frac{n}{2}$ times and thus has $\Theta\left(\log n\right)$ span. Hence, T_∞ can be described using the following recurrence relation.

$$T_\infty(n) = \begin{cases} \Theta\left(1\right) & \text{if } n = 1, \\ T_\infty\left(\frac{n}{2}\right) + \Theta\left(\log n\right) & \text{otherwise.} \end{cases}$$

Using Master Theorem (case 2), we obtain: $T_\infty(n) = \Theta\left(\log^2 n\right)$.

1(c) [**6 Points**] Find the parallel running time (i.e., T_p) and parallelism of ALT-PREFIX-SUM.

Solution.

Parallel running time with p processing elements on an input of size n (using Graham / Brent theorem): $T_p(n) = \Theta\left(\frac{T_1(n)}{p} + T_\infty(n)\right) = \Theta\left(\left(\frac{n}{p} + \log n\right)\log n\right)$.

Parallelism for an input of size n: $P(n) = \frac{T_1(n)}{T_\infty(n)} = \Theta\left(\frac{n}{\log n}\right)$.

1(d) [**5 Points**] Is ALT-PREFIX-SUM work-optimal? Why or why not?

Solution. ALT-PREFIX-SUM is work-optimal provided $pT_p(n) = \Theta\left(T_S(n)\right)$, where $T_S(n)$ is the work performed by an optimal serial prefix sum algorithm.

We know that for an input of lenth n, any prefix sum algorithm must perform $\Omega\left(n\right)$ work (since it must read the input at least once). We also know that one can compute the prefix sums in $\mathcal{O}\left(n\right)$ time simply by scanning the input sequence once from left to right once and always keeping track of the running sum. Hence, $T_S(n) = \Theta\left(n\right)$.

On the other hand, for ALT-PREFIX-SUM, $pT_p = \Theta\left(n\log n + p\log^2 n\right) = \omega\left(T_S(n)\right)$.

Hence, the algorithm is not work-optimal.

Use this page if you need additional space for your answers.

Question 2. [**30 Points**] **ϵ-Approximate Frequency.** Let $A[\,1:n\,]$ be an array of length n containing both positive and negative numbers. Let m be the number of positive numbers in A, and let $p = \frac{m}{n}$. We are interested in estimating the value of m fast. Clearly, one can find the exact value of m in $\Theta(n)$ time simply by scanning A once and counting the number of positive numbers.

For any $\epsilon \in (0, p]$, we say that $\hat{m}$ is an ϵ-approximation[2] of m provided $m - \epsilon n < \hat{m} < m + \epsilon n$.

Approx-Freq($A[\,1:n\,]$, ϵ)

(Inputs are an array $A[\,1:n\,]$ of n numbers, and a floating point parameter $\epsilon \in (0,1]$. This routine chooses a sample of size $\lceil \frac{6}{\epsilon^2} \ln n \rceil$ from A uniformly at random (with replacement), and uses that sample to estimate the number of entries of A that are positive.)

1. $s \leftarrow \lceil \frac{6}{\epsilon^2} \ln n \rceil$ {*size of the sample*}
2. $c \leftarrow 0$ {*a counter that keeps track of the frequency of v in the chosen sample*}
3. ***for*** $i \leftarrow 1$ ***to*** s ***do*** {*sample s items (with replacement) from A*}
4. $\quad j \leftarrow$ Random($1, n$) {*choose an integer uniformly at random from $[\,1, n\,]$*}
5. $\quad$ ***if*** $A[\,j\,] > 0$ ***then*** $c \leftarrow c + 1$ {*choose $A[\,j\,]$ as the next sample from A*}
6. ***return*** $\frac{c}{s} \times n$ {*return the estimate*}

Figure 2: Estimate the number of entries of $A[\,1:n\,]$ that are positive.

This problem asks you to show that the function Approx-Freq given in Figure 2 which runs in $\Theta\left(\frac{1}{\epsilon^2} \ln n\right)$ worst-case time returns an ϵ-approximation of m w.h.p. in n. While analyzing the algorithm we will drop the ceiling in line 1 for simplicity, i.e., we will assume that $s = \frac{6}{\epsilon^2} \ln n$.

2(*a*) [**5 Points**] Let μ be the expected value of c right after the loop in lines 3–5 completes execution. Show that $\mu = \left(\frac{6}{\epsilon^2}\right)\left(\frac{m}{n}\right) \ln n$.

Solution. For $1 \le i \le s$, let X_i be a 0-1 random variable defined as follows.

$$X_i = \begin{cases} 1 & \text{if iteration } i \text{ samples a positive number,} \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, $\Pr[X_i = 1] = \frac{m}{n} = p$, and $E[X_i] = 1 \times \Pr[X_i = 1] + 0 \times \Pr[X_i = 0] = p$.
Then $\mu = \sum_{i=1}^{s} E[X_i] = \sum_{i=1}^{s} p = sp = \left(\frac{6}{\epsilon^2}\right)\left(\frac{m}{n}\right) \ln n$.

[2] for simplicity, we have used '$<$' instead of '$\le$' in the definition of ϵ-approximation

2(b) **[12 Points]** Let $\hat{c}$ be the exact value of c right after the loop in lines 3–5 completes execution. Prove that for $0 < \epsilon < p$ and $\delta = \frac{\epsilon}{p}$,

$$\Pr[\ \hat{c} \le (1-\delta)\mu\ ] < \frac{1}{n^3} \quad \text{and} \quad \Pr[\ \hat{c} \ge (1+\delta)\mu\ ] < \frac{1}{n^2}.$$

Solution. Given $0 < \epsilon < p \Rightarrow 0 < \frac{\epsilon}{p} < 1 \Rightarrow 0 < \delta < 1$.

Observe from part 1(a) that $\hat{c} = \sum_{i=1}^{s} X_i$, where each X_i is a Poisson random variable.

We also have from part 1(a), $\mu = \left(\frac{6}{\epsilon^2}\right)\left(\frac{m}{n}\right)\ln n = \left(\frac{6}{\epsilon^2}\right)p\ln n$.

Hence, we can use the following Chernoff bound for the lower tail:

$$\begin{aligned}
\Pr[\ \hat{c} \le (1-\delta)\mu\ ] &\le e^{-\frac{\mu\delta^2}{2}} \\
&= e^{-\left(\frac{6}{\epsilon^2}\right)p\ln n\times\left(\frac{\epsilon}{p}\right)^2\times\frac{1}{2}} \\
&= e^{-\frac{3}{p}\ln n} \\
&= n^{-\frac{3}{p}} \\
&< n^{-3} && \left\{\because p < 1 \Rightarrow \frac{1}{p} > 1 \Rightarrow -\frac{3}{p} < -3 \Rightarrow n^{-\frac{3}{p}} < n^{-3}\right\} \\
&= \frac{1}{n^3}
\end{aligned}$$

For the upper tail we can use the following Chernoff bound:

$$\begin{aligned}
\Pr[\ \hat{c} \ge (1+\delta)\mu\ ] &\le e^{-\frac{\mu\delta^2}{3}} \\
&= e^{-\left(\frac{6}{\epsilon^2}\right)p\ln n\times\left(\frac{\epsilon}{p}\right)^2\times\frac{1}{3}} \\
&= e^{-\frac{2}{p}\ln n} \\
&= n^{-\frac{2}{p}} \\
&< n^{-2} && \left\{\because p < 1 \Rightarrow \frac{1}{p} > 1 \Rightarrow -\frac{2}{p} < -2 \Rightarrow n^{-\frac{2}{p}} < n^{-2}\right\} \\
&= \frac{1}{n^2}
\end{aligned}$$

2(c) [**5 Points**] let $\hat{m}$ be the estimate of m returned by APPROX-FREQ. Argue that for $0 < \epsilon < p$, the results from part 2(b) imply the following:

$$\Pr[\ \hat{m} \le m - \epsilon n\ ] < \frac{1}{n^3} \text{ and } \Pr[\ \hat{m} \ge m + \epsilon n\ ] < \frac{1}{n^2}.$$

Solution. Observe from line 6 of APPROX-FREQ that $\hat{m} = \frac{\hat{c}}{s} \times n$. Then

$$\begin{aligned}
\Pr[\ \hat{m} \le m - \epsilon n\ ] &= \Pr\left[\ \tfrac{\hat{c}}{s} \times n \le (1 - \epsilon \times \tfrac{n}{m})m\ \right] \\
&= \Pr\left[\ \hat{c} \le \left(1 - \epsilon \times \tfrac{n}{m}\right) s \left(\tfrac{m}{n}\right)\ \right] \\
&= \Pr\left[\ \hat{c} \le \left(1 - \tfrac{\epsilon}{p}\right) sp\ \right] && \left\{\because \tfrac{m}{n} = p \text{ (given)}\right\} \\
&= \Pr[\ \hat{c} \le (1 - \delta)\mu\ ] && \left\{\because \tfrac{\epsilon}{p} = \delta \text{ (from 2(}b\text{)) and } sp = \mu \text{ (from 2(}a\text{))}\right\} \\
&< \tfrac{1}{n^3} && \{\text{from part 2(}b\text{)}\}
\end{aligned}$$

$$\begin{aligned}
\text{and}\quad \Pr[\ \hat{m} \ge m + \epsilon n\ ] &= \Pr\left[\ \tfrac{\hat{c}}{s} \times n \ge (1 + \epsilon \times \tfrac{n}{m})m\ \right] \\
&= \Pr\left[\ \hat{c} \ge \left(1 + \epsilon \times \tfrac{n}{m}\right) s \left(\tfrac{m}{n}\right)\ \right] \\
&= \Pr\left[\ \hat{c} \ge \left(1 + \tfrac{\epsilon}{p}\right) sp\ \right] && \left\{\because \tfrac{m}{n} = p \text{ (given)}\right\} \\
&= \Pr[\ \hat{c} \ge (1 + \delta)\mu\ ] && \left\{\because \tfrac{\epsilon}{p} = \delta \text{ (from 2(}b\text{)) and } sp = \mu \text{ (from 2(}a\text{))}\right\} \\
&< \tfrac{1}{n^2} && \{\text{from part 2(}b\text{)}\}
\end{aligned}$$

2(d) [**8 Points**] Use your results from part 2(c) to argue that for $0 < \epsilon < p$, APPROX-FREQ returns an ϵ-approximation of m w.h.p. in n.

Solution. We have:

$$\begin{aligned}
\Pr[\ m - \epsilon n < \hat{m} < m + \epsilon n\ ] &= \Pr[\ (\hat{m} > m - \epsilon n)\ \cap\ (\hat{m} < m + \epsilon n)\ ] \\
&= \Pr[\ \hat{m} > m - \epsilon n\ ] + \Pr[\ \hat{m} < m - \epsilon n\ ] \\
&\quad - \Pr[\ (\hat{m} > m - \epsilon n)\ \cup\ (\hat{m} < m + \epsilon n)\ ] \\
&= \Pr[\ \hat{m} > m - \epsilon n\ ] + \Pr[\ \hat{m} < m - \epsilon n\ ] - 1 \\
&\quad \{\because \Pr[\ (\hat{m} > m - \epsilon n)\ \cup\ (\hat{m} < m + \epsilon n)\ ] = 1 \text{ for } \epsilon > 0\} \\
&= (1 - \Pr[\ \hat{m} \le m - \epsilon n\ ]) + (1 - \Pr[\ \hat{m} \ge m - \epsilon n\ ]) - 1 \\
&= 1 - \Pr[\ \hat{m} \le m - \epsilon n\ ] - \Pr[\ \hat{m} \ge m - \epsilon n\ ] \\
&> 1 - \tfrac{1}{n^3} - \tfrac{1}{n^2}\ \ \{\text{from part 2(}c\text{)}\} \\
&\ge 1 - \tfrac{2}{n^2}\ \ \left\{\because n^3 \ge n^2 \Rightarrow -\tfrac{1}{n^3} \ge -\tfrac{1}{n^2}\right\}
\end{aligned}$$

Hence, APPROX-FREQ returns an ϵ-approximation of m w.h.p. in n.

Use this page if you need additional space for your answers.

Question 3. [**20 Points**] **Matrix Rotation.** The *rotation* of an $n \times n$ matrix X is another $n \times n$ matrix X^R obtained by writing the i-th row of X as the $n - i + 1$-th column of X^R for $1 \le i \le n$. An example is given below.

$$X = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix} \quad \Rightarrow \quad X^R = \begin{bmatrix} d_1 & c_1 & b_1 & a_1 \\ d_2 & c_2 & b_2 & a_2 \\ d_3 & c_3 & b_3 & a_3 \\ d_4 & c_4 & b_4 & a_4 \end{bmatrix}$$

In this problem we will analyze the cache complexity of a couple of algorithms for rotating square matrices. We will assume that all matrices are stored in row-major order.

$3(a)$ [**5 Points**] Analyze the cache complexity of Iter-Matrix-Rotate given in Figure 3.

```
ITER-MATRIX-ROTATE( X, Y, n )
(Input is an n × n square matrix X[ 1 : n, 1 : n ]. This function generates the rotation of X in Y.)
   1. for i ← 1 to n do
   2.     for j ← 1 to n do
   3.         Y[ i, j ] ← X[ n − j + 1, i ]
```

Figure 3: Iterative matrix rotation.

Solution. Let us assume for simplicity that each cache block can hold B entries of a matrix, and $M \ge 2B$ (i.e., one cache block for writing to Y, and at least one block for reading from X).

Observe that the algorithm accesses X row-by-row and Y column-by-column. Since Y is laid out in row-major order, writing to Y will incur only $\Theta\left(1 + \frac{n^2}{B}\right)$ cache misses in total as long as there is one block dedicated to Y. Since X is also laid out in row-major order but accessed column-by-column, reading the entire matrix will incur $\Theta\left(n^2\right)$ cache misses if there are fewer than n cache blocks dedicated to reading X. However, if there are at least n cache blocks for X, then reading entire X will incur only $\mathcal{O}\left(n + \frac{n^2}{B}\right)$ cache misses. Moreover, reading X can incur as few as $\Theta\left(1 + \frac{n^2}{B}\right)$ misses provided at least $2n$ cache blocks are dedicated to the task. Overall, the algorithm will incur $\Theta\left(1 + \frac{n^2}{B} + n^2\right) = \Theta\left(n^2\right)$ cache misses provided the cache has fewer than $n + 1$ cache blocks, and $\mathcal{O}\left(1 + \frac{n^2}{B} + n + \frac{n^2}{B}\right) = \mathcal{O}\left(n + \frac{n^2}{B}\right)$ cache misses provided there are between $n + 1$ and $2n$ cache blocks, and $\Theta\left(1 + \frac{n^2}{B}\right)$ misses otherwise.

Hence, the number of cache misses incurred by the algorithm for $n \times n$ input matrices is:

$$Q_{iter}(n) = \begin{cases} \Theta\left(1 + \frac{n^2}{B}\right) & \text{if } M \ge (2n+1)B, \\ \mathcal{O}\left(n + \frac{n^2}{B}\right) & \text{if } (2n+1)B > M \ge (n+1)B, \\ \Theta\left(n^2\right) & \text{otherwise.} \end{cases}$$

3(b) **[10 Points]** Complete the recursive divide-and-conquer algorithm (Rec-Matrix-Rotate) for rotating a square matrix given in Figure 4. Analyze its cache complexity assuming a *tall* cache (i.e., $M = \Omega\left(B^2\right)$, where M is the cache size and B is the cache block size).

Rec-Matrix-Rotate(X, Y, n)

(Input is an $n \times n$ square matrix $X[\ 1 : n,\ 1 : n\ ]$. This function recursively generates the rotation of X in Y. We assume $n = 2^k$ for some integer $k \geq 0$. If $n > 1$, let X_{11}, X_{12}, X_{21} and X_{22} denote the top-left, top-right, bottom-left and bottom-right quadrants of X, respectively. Similarly for Y.)

1. ***if*** $n = 1$ ***then*** $Y \leftarrow X$ {*base case: the rotation of a* 1×1 *matrix is the matrix itself*}
2. ***else*** {*divide* X *and* Y *into quadrants, and generate the rotation of* X *recursively.*}
3. Rec-Matrix-Rotate(, ,) {*fill out*}
4. Rec-Matrix-Rotate(, ,) {*fill out*}
5. Rec-Matrix-Rotate(, ,) {*fill out*}
6. Rec-Matrix-Rotate(, ,) {*fill out*}

Figure 4: Recursive matrix rotation.

Solution. Rec-Matrix-Rotate can be completed as follows.

3. Rec-Matrix-Rotate(X_{11}, Y_{12}, $\frac{n}{2}$)
4. Rec-Matrix-Rotate(X_{12}, Y_{22}, $\frac{n}{2}$)
5. Rec-Matrix-Rotate(X_{21}, Y_{11}, $\frac{n}{2}$)
6. Rec-Matrix-Rotate(X_{22}, Y_{21}, $\frac{n}{2}$)

Let $Q(n)$ be the number of cache misses incurred by the algorithm for $n \times n$ input matrices. Then for some suitable constant γ,

$$Q(n) = \begin{cases} \mathcal{O}\left(n + \frac{n^2}{B}\right) & \text{if } n^2 \leq \gamma M, \\ 4Q\left(\frac{n}{2}\right) & \text{otherwise.} \end{cases}$$

Let $n^2 > \gamma M$ and let k be the smallest positive integer such that $\left(\frac{n}{2^k}\right)^2 \leq \gamma M$. Then expanding the recurrence for $Q(n)$, we obtain:

$$Q(n) = 4^k Q\left(\frac{n}{2^k}\right) = \mathcal{O}\left(\left(\frac{n^2}{M}\right)\left(\sqrt{M} + \frac{M}{B}\right)\right) = \mathcal{O}\left(\frac{n^2}{\sqrt{M}} + \frac{n^2}{B}\right) = \mathcal{O}\left(\frac{n^2}{B}\right) \text{ for } M = \Omega\left(B^2\right).$$

However, if the matrices are small enough to fit into the cache then $Q(n) = \mathcal{O}\left(1 + \frac{n^2}{B}\right)$.

Combining the two cases above, we obtain: $Q(n) = \mathcal{O}\left(\frac{n^2}{B} + 1 + \frac{n^2}{B}\right) = \mathcal{O}\left(1 + \frac{n^2}{B}\right)$.

$3(c)$ [**5 Points**] Is the cache complexity result of part $3(b)$ optimal? Why or why not?

Solution. Observe that any algorithm that tries to rotate matrix X must read X at least once. Since X is an $n \times n$ matrix stored in n^2 contiguous memory locations, one must incur $\Theta\left(1 + \frac{n^2}{B}\right)$ cache misses for reading X once. So, no algorithm for rotating X can incur fewer than $\Theta\left(1 + \frac{n^2}{B}\right)$ cache misses. Since the algorithm in part $3(b)$ incurs only $\mathcal{O}\left(1 + \frac{n^2}{B}\right)$ cache misses it shows optimal cache performance.

Use this page if you need additional space for your answers.

Appendix I: Some Elementary Probability Results

Given an event A, $\Pr[\ A\ ]$ denotes the probability of occurrence of A. By $\overline{A}$ we denote the opposite or complement of event A. Then $\Pr[\ \overline{A}\ ]$ denotes the probability of event A not occurring. Clearly,

$$0 \leq \Pr[\ A\ ], \Pr[\ \overline{A}\ ] \leq 1 \quad \text{and} \quad \Pr[\ \overline{A}\ ] = 1 - \Pr[\ A\ ].$$

Given two events A and B,

- $A \cap B$ is the event of both A and B occurring, and
- $A \cup B$ is the event of at least one of A and B occurring.

Then the corresponding complements are as follows:

$$\overline{A \cap B} = \overline{A} \cup \overline{B} \quad \text{and} \quad \overline{A \cup B} = \overline{A} \cap \overline{B}.$$

If A and B are mutually exclusive (i.e., both cannot occur simultanesouly[3]), then $\Pr[\ A \cap B\ ] = 0$. You might find the following relationship useful:

$$\Pr[\ A \cup B\ ] = \Pr[\ A\ ] + \Pr[\ B\ ] - \Pr[\ A \cap B\ ].$$

Observe that if A and B are mutually exclusive, the relationship given above reduces to:

$$\Pr[\ A \cup B\ ] = \Pr[\ A\ ] + \Pr[\ B\ ].$$

[3]e.g., if A is the event $(x < 5)$ and B is the event $(x > 5)$ then both A and B cannot be true (i.e., cannot occur) at the same time

Appendix II: Useful Tail Bounds

Markov's Inequality. Let X be a random variable that assumes only nonnegative values. Then for all $\delta > 0$, $Pr\left[X \geq \delta\right] \leq \frac{E[X]}{\delta}$.

Chebyshev's Inequality. Let X be a random variable with a finite mean $E[X]$ and a finite variance $Var[X]$. Then for any $\delta > 0$, $Pr\left[|X - E[X]| \geq \delta\right] \leq \frac{Var[X]}{\delta^2}$.

Chernoff Bounds. Let $X_1, \ldots, X_n$ be independent Poisson trials, that is, each X_i is a 0-1 random variable with $Pr[X_i = 1] = p_i$ for some p_i. Let $X = \displaystyle\sum_{i=1}^{n} X_i$ and $\mu = E[X]$. Following bounds hold:

Lower Tail:

- for $0 < \delta < 1$, $Pr\left[X \leq (1-\delta)\mu\right] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mu}$
- for $0 < \delta < 1$, $Pr\left[X \leq (1-\delta)\mu\right] \leq e^{-\frac{\mu\delta^2}{2}}$
- for $0 < \gamma < \mu$, $Pr\left[X \leq \mu - \gamma\right] \leq e^{-\frac{\gamma^2}{2\mu}}$

Upper Tail:

- for any $\delta > 0$, $Pr\left[X \geq (1+\delta)\mu\right] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}$
- for $0 < \delta < 1$, $Pr\left[X \geq (1+\delta)\mu\right] \leq e^{-\frac{\mu\delta^2}{3}}$
- for $0 < \gamma < \mu$, $Pr\left[X \geq \mu + \gamma\right] \leq e^{-\frac{\gamma^2}{3\mu}}$

Appendix III: The Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = \begin{cases} \Theta\left(1\right), & \text{if } n \leq 1, \\ aT\left(\frac{n}{b}\right) + f(n), & \text{otherwise,} \end{cases}$$

where, $\frac{n}{b}$ is interpreted to mean either $\left\lfloor \frac{n}{b} \right\rfloor$ or $\left\lceil \frac{n}{b} \right\rceil$. Then $T(n)$ has the following bounds:

Case 1: If $f(n) = \mathcal{O}\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.

Case 2: If $f(n) = \Theta\left(n^{\log_b a} \log^k n\right)$ for some constant $k \geq 0$, then $T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right)$.

Case 3: If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta\left(f(n)\right)$.