Comprehensive: Analysis of Algorithms (80 points) **Autumn 1989**

The list of functions that are considered closed form includes $\lfloor x \rfloor$, $\lceil x \rceil$, F_n, H_n, $n!$, $n^{\underline{k}}$, $n^{\overline{k}}$, $\binom{n}{k}$.

Problem 1 (20 points). The generating function for the elements of Pascal's triangle along a line of slope $-1/2$ is given by

$$g_n(z) = \sum_k \binom{n-k}{n-2k} z^k,$$

where $n \geq 0$.

1a. (7 points). Give a recurrence for $g_n(z)$ in terms of $g_{n-1}(z)$ and $g_{n-2}(z)$.

1b. (13 points). Evaluate $g_n(3)$ in closed form.

Problem 2 (20 points). Let $d(k)$ be the number of divisors of k (e.g., $d(10) = 4$).

2a. (7 points). Express

$$\sum_{k=1}^{n} d(k)$$

as $g(n) + O(n)$, with $g(n)$ in closed form.

2b. (13 points). Reduce the $O(n)$ error term from part (a) to $O(\sqrt{n})$. [*Hint:* Consider divisors smaller than $\sqrt{n}$ and greater than $\sqrt{n}$ separately.]

Problem 3 (11 points). Given n, exhibit an arithmetic progression $ai + b$ ($a \neq 0$) such that $\gcd(ai + b, aj + b) = 1$ for all $0 \leq i < j < n$.

Problem 4 (29 points). A binary search tree T is constructed with the elements of the set $[n] = \{1, 2, \ldots, n\}$. The elements of this set are inserted in T one by one, in the order given by a permutation of $[n]$ chosen uniformly at random from the set of all such permutations. [To insert a new element i in a tree: if the tree is empty, let i be the root. Otherwise compare i with the root j, and insert i in the left subtree or in the right subtree depending on whether $i < j$ or $i > j$.]

4a. (13 points). Show that the probability that element i is compared with element j when i is inserted in T is

$$p_{ij} = \frac{1}{|i-j|+1}.$$

[*Hint:* How does i reach j?]

4b. (7 points). Determine the expected depth of element i in T.

4c. (9 points). What is the expected depth in T of an element chosen uniformly at random from $[n]$?

Comprehensive Solutions: Analysis of Algorithms (80 points) **Autumn 1989**

What you needed to know to get a passing score on the Analysis of Algorithms comprehensive:

For Problem 1a, in addition to the meaning of generating functions, and the additive recurrence for binomial coefficients, you needed to know that $\sum_{P(k)} f(k) = \sum_{P(k+c)} f(k+c)$. This is a routine transformation on sums, emphasized in *Concrete Mathematics*.

For Problem 2a, you needed to know another basic summation transform: Instead of

$$\sum_i [i \le n] \sum_d [d \text{ divides } i],$$

reverse the summation order as

$$\sum_d \sum_i [d \text{ divides } i \le n],$$

and the inner sum is then $\lfloor n/i \rfloor$.

For Problem 3, to get substantial partial credit and have a good chance of recognizing one of the many solutions, you needed to know that $p \perp q \equiv p \perp q + mp$. Then

$$aj + b \perp ak + b \equiv aj + b \perp a(k - j).$$

Some particular cases impose substantial constraints:

$$(j = 0) \quad b \perp ak \Rightarrow b \perp k$$

so b has no prime factors $\le n$. Then candidate values for b might be large primes, $\text{lcm}(2..n) + 1$, $n! + 1$, or simply 1.

For Problem 4b, you needed to know that the expected number of p_j that are true is the sum of the probabilities that each is true. This is a very important special case of the law that the expected value of a sum is the sum of the expected values. You also needed to recognize that a sum of reciprocals is a harmonic number.

For Problem 4c, you needed to know the meaning of "expected value", and how to sum harmonic numbers. Since harmonic numbers are themselves sums, expanding into a bivariate sum and reversing the summation order is routine.

Using the above knowledge, you could get

1a	7
1b	–
2a	7
2b	–
3	4
4a	–
4b	7
4c	9
	—
	34

which is a passing grade, by turning the crank.

Computer Science Department
Stanford University
Comprehensive Examination in Automata, Languages, and Mathematical Theory of Computation
Autumn 1989

November 27, 1989

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. Be sure you have all the pages of this exam. There are 2 pages.

3. The number of POINTS for each problem indicates how elaborate an answer is expected. For example, an essay-type question worth 6 points or less doesn't deserve an extremely detailed answer, even though a person can expound at length on just about any topic in computer science.

4. The total number of points is 60, and the exam takes 60 minutes. This "coincidence" can help you plan your time.

5. This exam is CLOSED BOOK. You may not use any notes, calculators, computers, or outside help.

6. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why you think something is true when it is actually false. But no partial credit can be given if you write nothing.

Comprehensive:

Automata, Languages, and Mathematical Theory of Computation (60 points) **Autumn 1989**

Problem 1 (20 points). Given two strings x and y over an alphabet Σ, define *shuffle*(x, y) to be the set of all strings over Σ, which can be obtained by shuffling x and y together in an arbitrary way.

For instance, *shuffle*$(a, bc) = \{abc, bac, bca\}$.

shuffle can be defined recursively as follows:

$$\textit{shuffle}(x, \epsilon) = \{x\}$$
$$\textit{shuffle}(\epsilon, x) = \{x\}$$
$$\textit{shuffle}(ax, by) = \{az : z \in \textit{shuffle}(x, by)\} \bigcup \{bz : z \in \textit{shuffle}(ax, y)\}.$$

Let L be an arbitrary regular language over Σ.

1a. Let $L_1 = \{x \in \Sigma^* : \exists y \in \Sigma^* (\textit{shuffle}(x, y) \cap L \neq \phi)\}$.
Is L_1 necessarily regular?
Is it necessarily context-free?
Justify your answers briefly.

1b. Let $L_2 = \{xy : x, y \in \Sigma^* \text{ and } \textit{shuffle}(x, y) \cap L \neq \phi\}$.
Is L_2 necessarily regular?
Is it necessarily context-free?
Justify your answers briefly.

Problem 2 (10 points). Assume that there is no polynomial-time program that, given a graph, prints a Hamiltonian cycle, when one exists. Prove that the decision problem "HAMILTONIAN CYCLE" (does the given graph have a Hamiltonian cycle?) is not in the class P.

Problem 3 (20 points).

3a. Give the strongest possible Hoare-style verification rule to prove the partial correctness of the statement repeat P until E (where P is a program and E an expression).

3b. Consider the following program Q to compute the quotient and remainder of a nonnegative integer x and a positive integer y:

$\{x \geq 0 \wedge y > 0\}$
quot $:= 0$; *rem* $:= x$;
while $\{p(x, y, quot, rem)\}$ *rem* $\geq y$ **do**
begin
$i := 1$; $z := y$;
repeat
quot $:=$ *quot* $+ i$; *rem* $:=$ *rem* $- z$;
$i := i + i$; $z := z + z$
until $\{q(x, y, quot, rem, i, z)\}$ *rem* $< z$
end
$\{x = y \cdot quot + rem \wedge 0 \leq rem < y\}$.

Sketch a proof of the partial correctness of Q, using the rule you gave in (a). In particular, give loop invariants p and q such that all verification conditions have trivial proofs (you need not prove them).

3c. Define a variant function into a well-founded set, and sketch a proof of the termination of Q.

Problem 4 (10 points). Prove that the set of all Turing machines that accept infinitely many inputs is not r.e.

Hint:: Compare the problem of whether a Turing machine accepts infinitely many inputs with the complement of the halting problem.

Comprehensive Solutions:
Automata, Languages, and Mathematical Theory of Computation (60 points) **Autumn 1989**

Problem 1 (20 points).

1a. L_1 consists of all the subsequences of the strings in L. Let M be a finite state automaton accepting L. Construct M_1 from M by adding an ϵ-edge between every pair (s,t) of states of M such that there is an edge from s to t labeled with some symbol of Σ. The language accepted by M_1 is L_1. So L_1 is necessarily regular (and hence, context-free).

1b. We will show that L_2 is not necessarily connext-free (hence not regular). Take $L = (ab)^*(cd)^*$. Let $L_3 = \{a^m c^n b^m d^n : m, n \geq 0\}$. L_3 is not context-free. We will show that $L_2 \cap a^*c^*b^*d^* = L_3$. Since any string in L_2 has equal number of a's and b's, and equal number of c's and d's, it follows that $L_2 \cap a^*c^*b^*d^* \subseteq L_3$. Furthermore, for any $m, n \geq 0$, $(ab)^m(cd)^n \in \mathit{shuffle}(a^m c^n, b^m d^n)$. Hence $L_3 \subseteq L_2 \cap a^*c^*b^*d^*$.
Since context-free languages are closed under intersection with regular sets, it follows that L_2 is not context-free.

Problem 2 (10 points). We will assume that there exists a polynomial-time program Q which solves the decision problem and use it to construct a polynomial time program for printing a Hamiltonian cycle, thus contradicting the assumption.

Let G be a given graph.

If Q when given G as the input returns "no", we are done. If not, use the following procedure to find a Hamiltonian cycle. Assume that the edges of G are enumerated in some fixed order. Initially let $G' = G$. Repeat the following step for every edge e one by one in the order of enumeration:

> If Q, when given $G' - \{e\}$ as the input, says "yes",
> then set G' to $G' - \{e\}$, else leave G' unchanged.

At the end G' will be a subgraph of G composed solely of a Hamiltonian cycle. The time taken by this program equals the product of the number of edges of G by the time taken by Q, and thus is polynomial in the size of G.

Problem 3 (20 points).

3a.

$$\frac{\{\phi\}\, P\, \{\psi\} \qquad \{\psi \wedge \neg E\}\, P\, \{\psi\}}{\{\phi\}\ \textbf{repeat}\ P\ \textbf{until}\ E\ \{\psi \wedge E\}}.$$

3b. Let

$p(x, y, quot, rem)$: $x = y \cdot quot + rem \wedge rem \geq 0$,
$q(x, y, quot, rem, i, z)$: $p(x, y, quot, rem) \wedge z = i \cdot y$,
$r(x, y, quot, rem, i, z)$: $q(x, y, quot, rem, i, z) \wedge rem \geq z$.

The assertions ϕ and ψ of (a) correspond to $r(x, y, quot, rem, i, z)$ and $q(x, y, quot, rem, i, z)$, respectively. All verification conditions are easy to check:

(i) $x \geq 0 \wedge y > 0 \rightarrow p(x, y, 0, x)$,
(ii) $p(x, y, quot, rem) \wedge rem \geq y \rightarrow r(x, y, quot, rem, 1, y)$,
(iii) $r(x, y, quot, rem, i, z) \rightarrow q(x, y, quot + i, rem - z, i + i, z + z)$,
(iv) $q(x, y, quot, rem, i, z) \wedge rem \not< z \rightarrow q(x, y, quot + i, rem - z, i + i, z + z)$,
(v) $q(x, y, quot, rem, i, z) \wedge rem < z \rightarrow p(x, y, quot, rem)$,
(vi) $p(x, y, quot, rem) \wedge rem \not\geq y \rightarrow x = y \cdot quot + rem \wedge 0 \leq rem < y$.

3c. The value of *rem* is a suitable variant function into $(N, <)$. To prove termination, show that:

(i) $rem \geq 0$ is an invariant for both loops (which follows from (b)).

(ii) $z > 0$ is an invariant of the repeat-loop (which requires stronger invariants than (b)).

While (i) guarantees that $rem \in N$, condition (ii) establishes that the repeat-loop reduces *rem* with every execution ($rem > rem - z$). So does the while-loop, because every execution of the while-loop includes at least one execution of the repeat-loop.

Problem 4 (20 points). The set of programs that accept infinitely many inputs is not r.e.

Take an instance of the no-input halting problem: Does program P_1 halt, when started with null input? From P_1, construct a program P_2 that accepts n if P_1 has no computation that halts in at most n steps. If P_1 halts, P_2 accepts finitely many inputs; if no, P_2 accepts everything.

Suppose the set of programs that accept infinitely many inputs were r.e., i.e., the range of a computable function f. To decide whether P_1 halts, for each n, test whether P_1 halts in n steps (then P_1 halts) and whether $P_2 = f(n)$ (then P_1 does not halt) until one is true.

This is an application of the argument of the second form of Rice's theorem.

Computer Science Department
Stanford University
Comprehensive Examination in Computer Architecture
Autumn 1989

November 29, 1989

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. Be sure you have all the pages of this exam. There are 3 pages.

3. The number of POINTS for each problem indicates how elaborate an answer is expected. For example, an essay-type question worth 6 points or less doesn't deserve an extremely detailed answer, even though a person can expound at length on just about any topic in computer science.

4. The total number of points is 60, and the exam takes 60 minutes. This "coincidence" can help you plan your time.

5. This exam is CLOSED BOOK. You may not use any notes, calculators, computers, or outside help.

6. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why you think something is true when it is actually false. But no partial credit can be given if you write nothing.

Comprehensive: Computer Architecture (60 points) Autumn 1989

Problem 1 (17 points). Assume that we are given a 32-bit, pipelined processor. This processor is register-register (load/store) machine. The processor is configured to always stall the next instruction to execute if the current instruction takes more than a single cycle to execute. The following table gives the execution cycle counts and frequency distribution of different operations within a Floating-Point (FP) benchmark.

Operation	CPI	Frequency
Call/Return	2	5%
Branches	2	20%
ALU	1	45%
Load/Store	2	25%
FP Operation	5	5%

1a. (2 points). What is the CPI for this benchmark assuming a perfect memory system?

1b. (3 points). Now assume that we remove the hardware logic that automatically inserted the stall cycles for branches, calls/returns, and load/store operations. Now our compiler must insert NOP instructions in the delay slots if the delay slots cannot be filled with useful instructions. We find that the compiler can fill 70% of the branch slots, 100% of the call slots, and 40% of the load/store slots with useful instructions. What is the new CPI still assuming a perfect memory system?

1c. (4 points). The FP engineers say that they can increase the speed of the FP operations by 20%. What is the fraction of execution time spent doing FP operations in the original machine? Also, what speedup results from adding these FP improvements to our original machine?

1d. (3 points). Now the memory architects, not to be outdone, claim that they can also speedup the double-precision FP programs by implementing load double (LD) and store double (SD) instructions. These instructions replace two load word (LW) or store word (SW) instructions each, but they take 3 cycles to execute instead of 2. If we can replace 20% of the LW/SW instructions in our benchmark with LD/SD instructions, how much faster will the original machine run with just this enhancement?

1e. (5 points). Finally, a customer wants to know how much the CPI increases for the original machine if we use a split instruction and data caches. The caches are Write-Back (copy-back) and have a miss rate of 2% and 4% respectively. Assume that the miss penalty and the time to write a cache block to main memory are both 10 cycles. In addition, 40% of the cache blocks are dirty at any one time.

Problem 2 (13 points). You are given a computer with a 48-bit virtual address and a 32-bit physical address. The processor uses a 16KB unified cache with a 16B line.

2a. (8 points). What is the smallest page size you can use to allow overlapped TLB and cache access if the cache is direct-mapped? If the cache is 4-way set associative? Also give the number of bits of physical address used for cache tag, index, and offset in each case.

2b. (5 points). If you buy 32MB of main memory with this machine, and you are told the machine has the following characteristics:

CPI = 1.6 cycles
1.3 references/instruction are sent to TLB
bus can transfer 4B/cycle
page fault every 100 * (number of pages in memory) references

how much bus bandwidth is used for the page transfer from disks to main memory by each page size in part (2a)?

Problem 3 (4 points). What is the Boolean function, expressed as a sum of products, for the following Karnaugh map?

AB \ *CD*	00	01	11	10
00	1	0	0	1
01	1	0	1	1
11	1	1	1	1
10	1	1	1	1

Problem 4 (4 points). Synthesize the following equation using only two-input nand gates. (All signals are active high).

$$A = \overline{(B \cdot C) + (D \cdot E)}.$$

Problem 5 (22 points). Build a specialized processor that evaluates the following polynomial:

$$A_6 x^6 + A_4 x^4 + A_2 x^2 + A_0 .$$

The expression can be calculated using Horner's rule:

$$(((A_6)x^2 + A_4)x^2 + A_2)x^2 + A_0 .$$

You can use multipliers, adders, registers, muxes and any other logic gates. The specification of the computation is given by the following RTL (register transfer level) description.

Note on RTL notation: operations separated by commas (,) are executed in the same clock; operations in different clocks are separated by semi-colons (;).

Input registers	X (64-bit), RESET (1-bit)
Output registers	P (64-bit)
Internal registers	XSQ, A6, A4, A2, A0 (all 64-bit)

```
S0:  A6 <- X;
S1:  A4 <- X;
S2:  A2 <- X;
S3:  A0 <- X;
S4:  XSQ <- X*X, SUM <- A6;
S5:  SUM <- SUM*XSQ+A4;
S6:  SUM <- SUM*XSQ+A2;
S7:  P <- SUM*XSQ+A0, if RESET then goto S0 else goto S4;
```

5a. (4 points). What is the smallest number of buses needed?

5b. (7 points). Draw a logic diagram for the data processing unit (also known as the data path).

5c. (7 points). Design a hardwired control unit for the machine. Show only the logic to implement the state transitions. (You do not need to show how the control signals are generated.)

5d. (4 points). Describe how you would modify the data processing unit and the control unit of the machine if the polynomial is of degree 100:

$$A_{100} x^{100} + A_{98} x^{98} + \cdots + A_0 .$$

Comprehensive Solutions: Computer Architecture (60 points) **Autumn 1989**

Problem 1 (17 points).

Operation	CPI	Frequency
Call/Return	2	5%
Branches	2	20%
ALU	1	45%
Load/Store	2	25%
FP Operation	5	5%

1a. (2 points).

$$CPIe = 5\% \times 2 + 20\% \times 2 + 45\% \times 1 + 25\% \times 2 + 5\% \times 5 = 1.7 \text{ cycles.}$$

1b. (3 points).

$$\begin{aligned} CPIe = {} & 5\% \times (1 \times 100\% + 2 \times 0\%) + 20\% \times (1 \times 70\% \\ & + 2 \times 30\%) + 45\% \times 1 + 25\% \times (1 \times 40\% + 2 \times 60\%) + 5\% \times 5 \\ = {} & 1.41 \text{ cycles.} \end{aligned}$$

1c. (4 points).

Fraction of execution doing FP = $5\% \times 5/1.7 = 14.7\%$

Speedup = $1/[(1 - 14.7\%) + 14.7\%/1.2] = 1.025 \Rightarrow 2.5\%$ faster.

1d. (3 points). Replace 20% of 25 lw/sw instructions $\Rightarrow$ 20 lw/sw plus 5/2 = 2.5 new ld/sd instructions.

New instruction distribution:

LW/SW	20.0%	2 cycles
LD/SD	2.5%	3
B/Call	25.0%	2
ALU	45.0%	1
FP	5.0%	5
total	97.5%	

$$CPIe = 2 \times 20/97.5 + 3 \times 2.5/97.5 + 2 \times 25/97.5 + 1 \times 45/97.5 + 5 \times 5/97.5 = 1.72 \text{ cycles}$$

perf = IC × CPI × CT $\Rightarrow$ CTold = CTnew

ratio of perf = (ICold × CPIold)/(ICnew × CPInew)

$= (1 \times 1.7)/(.975 \times 1.72) = 1.0137 \Rightarrow 1.37\%$ faster.

1e. (5 points).

$$CPIc = 1 \times 0.02 \times 10 + 0.25 \times 0.04 \times (10 + 0.4 \times 10) = 0.34 \text{ cycles.}$$

Problem 2 (13 points).

2a. (8 points).

dm: index all blocks ⇒ 2^{14} = 16KB page
tag = 18 index = 10 offset = 4

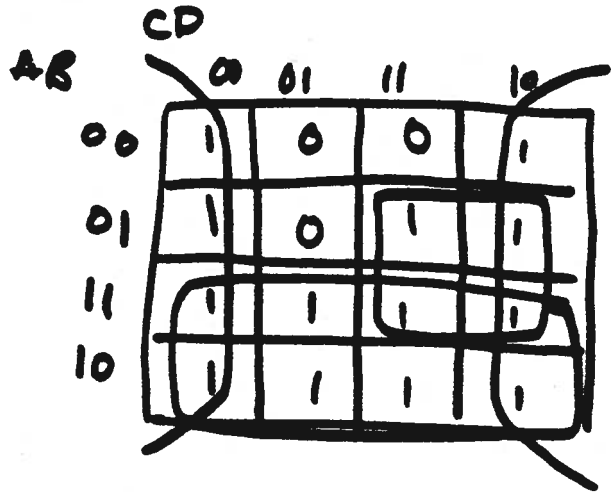
4-way: $2^{14}/4 \Rightarrow 2^{12}$ = 4KB page
tag = 20 index = 8 offset = 4

2b. (5 points).

dm: 16KB page → 2K pages in mm → 204800 refs/pf
204800 r/pf × 1.6 c/i/1.3 r/i = 252K c/pf ⇒ 3.97u pf/c
3.97u pf/c × 16 KB/pf/4 B/c = 0.016 B/c ⇒ 1.6%

4-way: 4KB page → 8K pages in mm → 809600 refs/pf
809600 r/pf × 1.6 c/i/1.3 r/i = 1.01M c/pf ⇒ 0.99u pf/c
0.99u pf/c × 4 KB/pf/4 B/c = 0.0010 B/c ⇒ 0.10%

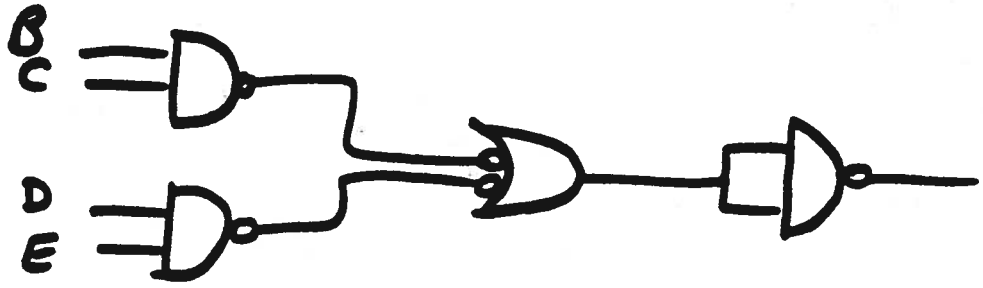
Problem 3 (4 points).



$A + \bar{D} + BC$

Problem 4 (4 points).

$$\overline{(B \cdot C) + (D \cdot E)}$$

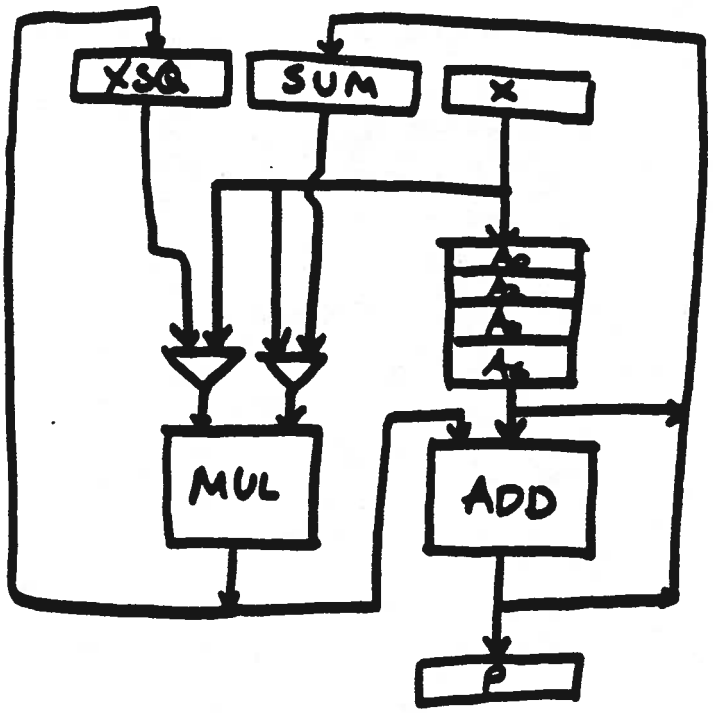


Problem 5 (22 points).

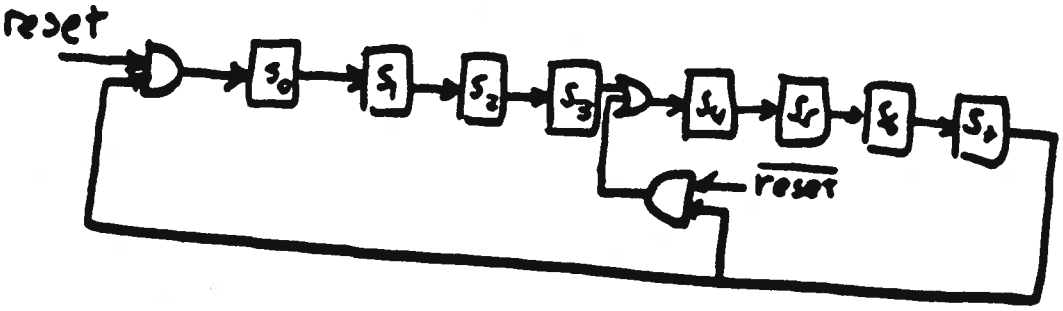
5a. (4 points).

5 buses, to carry values: SUM
XSQ
Ai
SUM * XSQ
SUM * XSQ + Ai

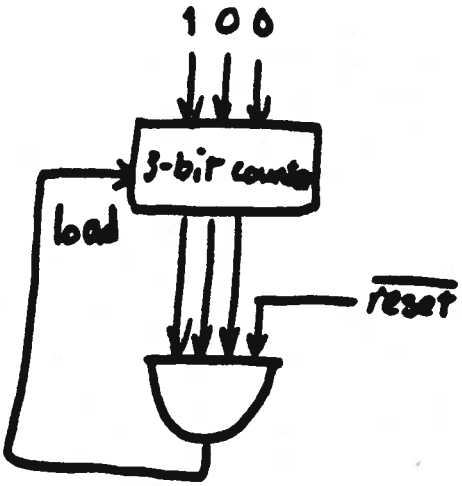
5b. (7 points).



5c. (7 points). Simplest solution:



OR



5d. (4 points).

Data processing unit:

More registers to hold A_i, otherwise unchanged.

Control unit:

Use the counter approach.

Need to modify the LOAD condition

Add a reset signal for the counter.

Computer Science Department
Stanford University
Comprehensive Examination in Numerical Analysis
Autumn 1989

November 29, 1989

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. The number of POINTS for each problem indicates how elaborate an answer is expected. For example, an essay-type question worth 6 points or less doesn't deserve an extremely detailed answer, even though a person can expound at length on just about any topic in computer science.

3. The total number of points is 30, and the exam takes 30 minutes. This "coincidence" can help you plan your time.

4. This exam is CLOSED BOOK. You may not use any notes, calculators, computers, or outside help.

5. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why you think something is true when it is actually false. But no partial credit can be given if you write nothing.

Comprehensive Solutions: Numerical Analysis (30 points) Autumn 1989

Problem 1 (14 points).

1a. (4 points). Writing equations in matrix form:

$$Au = f.$$

Since

$$\det|A| = 1 - ab,$$

$\det|A| \neq 0$ when $ab \neq 1$. Thus the LU factorization exists.

1b. (5 points). We can write the iteration as:

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = C \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \tag{1}$$

where

$$C = \begin{pmatrix} 0 & -a \\ -b & 0 \end{pmatrix}.$$

In addition, we have

$$\begin{pmatrix} \xi \\ \eta \end{pmatrix} = C \begin{pmatrix} \xi \\ \eta \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix}. \tag{2}$$

Subtracting (2) from (1), yields

$$e_{k+1} = Ce_k \tag{3}$$

where

$$e_k = \begin{pmatrix} x_k - \xi \\ y_k - \eta \end{pmatrix}.$$

Thus,

$$e_{k+2} = C^2 e_k.$$

Finally, taking norms gives

$$\| e_{k+1} \| \leq \| C^2 \| \ \| e_k \|.$$

Since $|ab| < 1$, $\| C^2 \| < 1$ and thus the iteration converges.

1c. (5 points). Once again we can write the iteration in matrix form as:

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = C \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} 1 \\ 2 - b \end{pmatrix}.$$

where this time

$$C = \begin{pmatrix} 0 & -a \\ 0 & ab \end{pmatrix}.$$

It is possible to show that for this matrix C, (3) holds and that the norm of $\| C \| < 1$.

Problem 2 (16 points).

2a. (4 points). Expand $f(x)$ about midpoint yields:

$$f(x) = f(y) + (x-y)f'(y) + \frac{1}{2}(x-y)^2 f''(y) + \cdots .$$

where

$$y = a + \frac{h}{2}.$$

Integrate series and simplify:

$$\int_a^{a+h} f(x)\,dx = hf(y) + \int_a^{a+h} (x-y)f'(y)\,dx + \frac{1}{2}\int_a^{a+h} (x-y)^2 f''(y)\,dx + \cdots .$$

Substitute $z = x - a - \frac{h}{2}$ on the righthand side:

$$\int_a^{a+h} f(x)\,dx = hf(y) + \int_{-\frac{h}{2}}^{\frac{h}{2}} zf'(y)\,dz + \frac{1}{2}\int_{-\frac{h}{2}}^{\frac{h}{2}} z^2 f''(y)\,dz + \cdots .$$

Finally, we get:

$$\int_a^{a+h} f(x)\,dx = hf(y) + \frac{h^3}{24}f''(\eta)$$

where $\eta \in [a, a+h]$.

2b. (4 points). Summing the error terms:

$$\begin{aligned} E &= \frac{1}{24}\sum_{i=1}^{n} h^3 f''(\eta_i) \quad \eta_i \in [a+ih-h, a+ih+h] \\ &= \frac{1}{24}\sum_{i=1}^{n} h^3 f''(\eta) \quad \eta \in [a, a+b] \\ &= \frac{1}{24} nh^3 f''(\eta) \\ &= \frac{(b-a)}{24} h^2 f''(\eta). \end{aligned}$$

2c. (4 points). Must show that the error expression is positive

$$\begin{aligned} f''(x) &= 6x^{-4} > 0. \\ h^2 &> 0. \\ b - a &= 1 > 0. \end{aligned}$$

2d. (4 points). Determine n such that error expression above is less than 10^{-3}.

$$\max_{\eta \in [1,2]} f''(\eta) = 6.$$

So

$$n = \sqrt{\frac{6}{24 \times 10^{-3}}}.$$

Computer Science Department
Stanford University
Comprehensive Examination in Software Systems
Autumn 1989

December 1, 1989

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. The number of POINTS for each problem indicates how elaborate an answer is expected. For example, an essay-type question worth 6 points or less doesn't deserve an extremely detailed answer, even though a person can expound at length on just about any topic in computer science.

3. The total number of points is 60, and the exam takes 60 minutes. This "coincidence" can help you plan your time.

4. This exam is CLOSED BOOK. You may not use any notes, calculators, computers, or outside help.

5. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why you think something is true when it is actually false. But no partial credit can be given if you write nothing.

Comprehensive: Software Systems (60 points) Autumn 1989

Problem 1 (15 points). Consider what happens when a page fault occurs. Briefly answer the following questions in a sentence or two.

1a. (3 points). How does the system decide which page should be brought in?

1b. (3 points). How does the system decide which page fraime should be used for the new page?

1c. (3 points). What happens to the page that used to be in that page frame?

1d. (3 points). What system tables must be updated as a result of this operation?

1e. (3 points). When does paging work? That is, when is the time cost of paging neglible compared to the execution of a program?

Problem 2 (10 points). Suppose a multi-level feedback scheduling algorithm can be parameterized by N, the number of levels, and $Q[1..N]$, an array of priorities indexed by level.

2a. (5 points). What parameter values would give the same effect as a first-come-first-serve algorithm?

2b. (5 points). What parameter values would give the same effect as a round-robin algorithm with quantum T?

Problem 3 (35 points). A certain freight train can hold a maximum of B boxes. The train travels across country once a day, stopping to drop off and pick up boxes at each of N stations along the way.

You are to write two functions, *train* and *sendbox*, that simulate the operation of the train with respect to loading and unloading boxes. The *train* function is called by a process simulating the train; *sendbox* is called by a process simulating a box that is sent on the train. You are only to write these functions; do not be concerned with the other code for the processes.

You are to use general semaphores to synchronize the train process and the box processes. The *train* function is passed a single integer parameter s that identifies a station. This function is called when the train stops at the specified station. To synchronize the unloading of boxes, it is necessary to wake up a box process and then wait until the box is taken off the train. Similarly, it is necessary to wake up box processes at the station and wait for them to be loaded on the train.

The *sendbox* function is passed two integer parameters identifying the stations where the box is sent from and to, respectively. This function should wait for the train to come to the originating station, load the box on the train, wait for the train to get to the destination station, and then unload the box. A box may not be loaded on the train if it is full (has B boxes already). If the destination of a box is in the opposite direction the train is traveling, the box should be loaded anyway and will be delivered the next day.

You can program these functions in Pascal or C. Be sure to specify initial values for all semaphores and global variables. Try to maximize concurrency while avoiding busy-waiting and the possibility of deadlock.

Problem 1 (15 points).

1a. (3 points). It looks at which address caused the fault (demand page selection).

1b. (3 points). It uses a page replacement algorithm, such as LRU, which hopefully picks a page frame containing a page that will not be used for a while.

1c. (3 points). If it has been changed since it was written on the disk, the latest version must be written to disk.

1d. (3 points). Both the maps for the old page (mark it invalid) and the new page (mark it valid), as well as the table that says which pages are in real memory.

1e. (3 points). Paging is infrequent if real memory is large enough to hold the working sets of all running processes.

Problem 2 (10 points).

2a. (5 points). $N = 1$, $Q[1] = \infty$.

2b. (5 points). $N = 1$, $Q[1] = T$.

Problem 3 (35 points). Solution in C++:

```
/*
 * We need two arrays indexed by station:  one for the boxes waiting
 * to be loaded on the train at a station and one for the boxes on the train
 * that should be unloaded at a particular station.  We also need to keep
 * track of the number of boxes on the train at any time.
 *
 * For each station, we need a mutual exclusion semaphore for accessing
 * the numWaiting arrays, a wait semaphore for loading, and a wait semaphore
 * for unloading.  We also need two semaphores for the train,
 * one for loaded boxes and one for unloaded boxes.
*/
```

```
    int numWaitingAtStation[N], numWaitingForStation[N], boxes;
    Semaphore mutex(1)[N];
    Semaphore waitingAtStation(0)[N], waitingForStation(0)[N];
    Semaphore load(0), unload(0);
    void train(int s) {
        int unloading, loading;
        mutex[s].P();
        unloading = numWaitingForStation[s];
        numWaitingForStation[s] = 0;
        boxes -= unloading;
        if (boxes + numWaitingAtStation[s] > B) {
          loading = B - boxes;
        } else {
          loading = numWaitingAtStation[s];
        }
        numWaitingAtStation[s] -= loading;
        mutex[s].V();
        boxes += loading;
        /* unload boxes destined for this station */
        for (int i = 0; i < unloading; i++) {
          waitingForStation[s].V();
          unload.P();
        }
        /* load boxes waiting at station */
        for (i = 0; i < loading; i++) }
          waitingAtStation[s].V();
          load.P();
        }
    }
    void sendbox(int fromStation, int toStation) {
        /* add to waiting at station */
        mutex[fromStation].P();
        ++numWaitingAtStation[fromStation];
        mutex[fromStation].V();
        /* wait for the train */
        waitingAtStation[fromStation].P();
        /* load */
        load.V();
        /* add to waiting for toStation */
        mutex[toStation].P();
        ++numWaitingForStation[toStation];
        mutex[toStation].V();
        /* wait for the train to reach toStation */
        waitingForStation[toStation].P();
        /* unload */
        unload.V();
    }
```