問題 1

オペレーティングシステムに関して以下の問いに答えよ．

(1) 5つのプロセス P_0, P_1, P_2, P_3, P_4 に対するスケジューリングについて，各プロセス P_i の到着時間（ms）と処理時間（ms）をそれぞれ A_i と C_i で表すものとする．また，同時に実行可能なプロセスは1つのみで，コンテキストスイッチのオーバーヘッドは無視できるものとする．$A_0 = 35$, $A_1 = A_2 = A_3 = 25$, $A_4 = 0$, $C_0 = 10$, $C_1 = 15$, $C_2 = 20$, $C_3 = 30$, $C_4 = 50$ である場合に，5つのプロセスを Preemptive Shortest Job First アルゴリズムでスケジュールしたときの平均ターンアラウンド時間と平均応答時間を求めよ．ここで，ターンアラウンド時間とはプロセスの到着から実行完了までの時間とし，応答時間とはプロセスの到着から実行開始までの時間とする．

(2) 問い (1) と同じ到着時間と処理時間をもつ5つのプロセスを Non-Preemptive Shortest Job First アルゴリズムでスケジュールしたときの平均ターンアラウンド時間と平均応答時間を求めよ．

(3) 問い (1) と同じ到着時間と処理時間をもつ5つのプロセスをタイムスライスが 10 ms の Round Robin アルゴリズムでスケジュールしたときの平均ターンアラウンド時間と平均応答時間を求めよ．プロセスがタイムスライスを使い切らなかった場合は即座に次のプロセスのタイムスライスが開始されるものとする．また，新たに到着したプロセスは Round Robin キューの末尾に追加されるものとし，複数のプロセスが同時にキューの末尾に到着した場合には残りの処理時間の短いプロセスの方が優先されて追加されるものとする．

(4) 現実のオペレーティングシステムではコンテキストスイッチのオーバーヘッドは無視できない．このオーバーヘッドを考慮した場合，Round Robin アルゴリズムの利点と欠点について CPU スケジューリングとメモリ管理の観点から説明せよ．

(5) 現実のオペレーティングシステムではプロセス優先度を決定するためにしばしば Aging 方式が使われる．Aging 方式の基本概念と古典的な静的優先度方式に対する優位性について説明せよ．

問題 2

C言語で書かれた以下のプログラムは整数配列 a の a[i] から a[j-1] までを昇順に整列する関数 mysort(a, i, j) を定義している (i<j). プログラム中の関数 multfrac(k, l, m) は k, l, m が正の整数であるとき $k \times \frac{l}{m}$ 以上の最小の整数を求める関数であり, w, x, y, z は正の整数定数とする. 整数の演算はオーバーフローしないものとする.

```
int multfrac(int k, int l, int m) {
    return (k * l + (m-1))/m;
}

void compare_swap(int *p, int *q) {
    if (*p > *q) {
        int tmp = *p;
        *p = *q;
        *q = tmp;
    }
}

void mysort(int a[], int i, int j) {
    int k = j - i;
    if (k < 4) {
        [ X ]
    }
    else {
        mysort(a, i, i + multfrac(k, x, w));
        mysort(a, j - multfrac(k, y, w), j);
        mysort(a, i, i + multfrac(k, z, w));
    }
}
```

以下の問いに答えよ.

(1) (w, x, y, z) が (4, 3, 3, 3) である場合, 空欄 [X] に入れるべき適切なコードを答えよ. ただし, compare_swap 以外の関数呼び出しは不可とする. なお, コードは複数行にわたっても良い.

(2) mysort(a, 0, n) が呼び出された時にコード断片 [X] が実行される回数の合計を $T(n)$ と表記する. (w, x, y, z) が (4, 3, 3, 3) である場合, $T(n)$ の n に関するオーダーを与えよ.

(3) (w, x, y, z) が (4, 2, 3, 3), (4, 3, 2, 3), (4, 3, 3, 2), (4, 2, 3, 2) である場合のそれぞれについて, mysort が常に正しく動作するか否かを答えよ.

(4) mysort が常に正しく動作するために w, x, y, z が満たすべき必要十分条件を答えよ.

問題 3

$\Sigma_1 = \{\mathtt{a}, \mathtt{b}\}$, $\Sigma_2 = \{\mathtt{t}, \mathtt{f}\}$ とする．語 $w \in \Sigma_1^*$ について，w の長さを $|w|$ と書く．また，空語（すなわち長さ 0 の語）を ϵ と書く．語 $w \in \Sigma_1^*$ について，関数 $f_w \in \Sigma_1^* \to \Sigma_2^*$ を以下のように定義する．

$$f_w(w') = \{x_1 \cdots x_{|w'|} \in \Sigma_2^* \mid x_i = \begin{cases} \mathtt{t} & \text{if } w' = uwv \text{ for some } u, v \in \Sigma_1^* \text{ such that } |u| = i-1 \\ \mathtt{f} & \text{otherwise} \end{cases} \text{ for each } i \in \{1, \ldots, |w'|\}\}.$$

すなわち，$f_w(w')$ は w' から，w と一致する部分文字列の開始位置を $\mathtt{t}$ で置き換え，それ以外の位置を $\mathtt{f}$ で置き換えて得られるものである．例えば，$f_{\mathtt{aa}}(\mathtt{baaab}) = \mathtt{fttff}$，$f_\epsilon(\mathtt{abbab}) = \mathtt{ttttt}$ である．さらに，関数 f_w を，以下の定義によって Σ_1 上の言語を Σ_2 上の言語に写像する関数 f_w^* に拡張する．

$$f_w^*(L) = \{f_w(w') \mid w' \in L\}.$$

例えば，$f_{\mathtt{ab}}^*(\{(\mathtt{abb})^n \mid n \geq 0\}) = \{(\mathtt{tff})^n \mid n \geq 0\}$ である．

以下の問いに答えよ．

(1) $f_{\mathtt{aba}}(\mathtt{babababa})$ を求めよ．

(2) $f_{\mathtt{aba}}^*(\Sigma_1^*)$ を正規表現を用いて表せ．

(3) 語 $w \in \Sigma_1^*$ (ただし $|w| > 0$) および決定性有限オートマトン $\mathcal{A} = (Q, \Sigma_1, \delta, q_0, F)$ が与えられたとし，$\mathcal{A}$ が受理する言語を L とする．ただし Q, δ, q_0, F は，それぞれ $\mathcal{A}$ の状態集合，遷移関数，初期状態，受理状態の集合を表すものとし，遷移関数 $\delta \in Q \times \Sigma_1 \to Q$ は全域関数であるとする．$f_w^*(L)$ を受理する非決定性有限オートマトンを，簡単な説明とともに与えよ．ϵ 遷移を用いてもよい．

(4) 以下の命題が真ならばその証明の概略（$f_w^*(L)$ を受理するプッシュダウンオートマトンまたは $f_w^*(L)$ を生成する文脈自由文法を簡単な説明とともに示せばよい）を，偽ならば反例を示せ．
命題：「すべての語 $w \in \Sigma_1^*$ について，$L \subseteq \Sigma_1^*$ が文脈自由言語ならば，$f_w^*(L)$ も文脈自由言語である」

問題 4

コンピュータアーキテクチャに関する以下の問いに答えよ.

(1) 命令 i が生成する結果を命令 j が利用する可能性がある場合に，命令 i に対する命令 j のデータ依存が存在するといい，$j \rightarrow i$ と表す．以下のプログラムに存在するデータ依存をすべて示せ．各命令の動作はプログラム中のコメント（#以降の記述）の通りである．プログラムを実行するプロセッサは x0 から x31 までの 32 個のレジスタを持ち，x0 は常に 0 を保持するゼロレジスタとする．コメント中，メモリの addr 番地へのアクセスを "memory[addr]" と表す．

```
命令 0)        addi x3, x0, 64     # x3 <- x0 + 64
命令 1)        addi x4, x0, 0      # x4 <- x0 + 0
命令 2)        addi x5, x0, 0      # x5 <- x0 + 0
命令 3) Loop:  lw x6, 0 (x4)       # x6 <- memory[x4 + 0]
命令 4)        add x5, x5, x6      # x5 <- x5 + x6
命令 5)        addi x4, x4, 4      # x4 <- x4 + 4
命令 6)        blt x4, x3, Loop    # if x4 < x3, goto Loop
命令 7)        sw x5, 4096 (x0)    # memory[x0 + 4096] <- x5
```

(2) 毎クロックサイクル最大 1 命令を発行する，5 段ステージのパイプラインプロセッサを考える．このプロセッサは，命令フェッチ (IF) ステージ，命令デコードとレジスタフェッチ (ID) ステージ，実行 (EX) ステージ，メモリアクセス (MA) ステージ，レジスタ書き込み (WB) ステージの 5 つのステージで構成される．各レジスタのビット幅は 32 である．1 クロックサイクルでアクセス可能な命令メモリとデータメモリを持つものとし，ロードワード命令 lw およびストアワード命令 sw が MA ステージでストールすることはない．命令 lw に関する被ロード・データハザードが発生する場合，IF ステージ，ID ステージ，EX ステージを 1 クロックサイクルの間ストールさせる．分岐命令 blt (branch if less than) は，分岐結果が EX ステージで確定するまで，IF ステージおよび ID ステージをストールさせる．つまり，分岐命令のフェッチ後，2 クロックサイクルの間，後続の命令をフェッチしない．EX ステージでの実行結果および MA ステージでのロード結果は，EX ステージに適切にフォワーディングされるものとする．

被ロード・データハザードとは何かを説明せよ．また，このプロセッサ上で問い (1) のプログラムを実行する際に，被ロード・データハザードがどのように発生するかを説明せよ．

(3) 問い (2) のプロセッサ上で，問い (1) のプログラムを実行する際に要するクロックサイクル数を求めよ．また，平均 IPC (instructions per cycle) を小数第 2 位まで求めよ．

(4) 問い (1) のプログラムおよび問い (2) のプロセッサを例に，動的分岐予測の原理と役割を説明せよ．

Problem 1

Answer the following questions on operating systems.

(1) For the scheduling of five processes P_0, P_1, P_2, P_3, and P_4, the arrival time (ms) and the computation time (ms) of each process P_i are denoted by A_i and C_i, respectively. Also, assume that only one process is allowed to execute at any instant, and the overhead of context switches can be ignored. Obtain the average turnaround time and the average response time when the five processes are scheduled by the Preemptive Shortest Job First algorithm, where $A_0 = 35$, $A_1 = A_2 = A_3 = 25$, $A_4 = 0$, $C_0 = 10$, $C_1 = 15$, $C_2 = 20$, $C_3 = 30$, and $C_4 = 50$. Here, the turnaround time refers to the time interval from the arrival of the process to the completion of its execution, and the response time refers to the time interval from the arrival of the process to the beginning of its execution.

(2) Obtain the average turnaround time and the average response time when the five processes with the same arrival and computation times as those given in question (1) are scheduled by the Non-Preemptive Shortest Job First algorithm.

(3) Obtain the average turnaround time and the average response time when the five processes with the same arrival and computation times as those given in question (1) are scheduled by the Round Robin algorithm with the time slice 10 ms. The next time slice starts immediately when the current process does not exhaust its time slice. Also, a new process is added to the end of the Round Robin queue upon its arrival, and ties are broken in favor of the processes with shorter remaining computation times if multiple processes arrive at the end of the queue simultaneously.

(4) In real-world operating systems, the overhead of context switches cannot be ignored. Explain the pros and cons of the Round Robin algorithm from the viewpoint of CPU scheduling and memory management, when this overhead is considered.

(5) The Aging scheme is often used to determine process priorities in real-world operating systems. Explain the basic concept of the Aging scheme and its advantage over the classical static-priority scheme.

Problem 2

The following program written in C language defines a function `mysort(a, i, j)`, which sorts an integer array `a` from `a[i]` to `a[j-1]` in ascending order (where `i<j`). The function `multfrac(k, l, m)` used in the program returns the least integer that is greater than or equal to $k \times \frac{l}{m}$, when `k`, `l`, and `m` are positive integers. Assume that `w`, `x`, `y`, and `z` are positive integer constants. Assume also that integer operations never overflow.

```
int multfrac(int k, int l, int m) {
    return (k * l + (m-1))/m;
}

void compare_swap(int *p, int *q) {
    if (*p > *q) {
        int tmp = *p;
        *p = *q;
        *q = tmp;
    }
}

void mysort(int a[], int i, int j) {
    int k = j - i;
    if (k < 4) {
        [      X      ]
    }
    else {
        mysort(a, i, i + multfrac(k, x, w));
        mysort(a, j - multfrac(k, y, w), j);
        mysort(a, i, i + multfrac(k, z, w));
    }
}
```

Answer the following questions.

(1) Answer appropriate code to fill the blank [X] when (`w`, `x`, `y`, `z`) is (4, 3, 3, 3). You are not allowed to use a function call except for `compare_swap`. The code may consist of multiple lines.

(2) Let $T(n)$ denote the number of times that the code fragment [X] is executed while `mysort(a, 0, n)` is called. Give the order of $T(n)$ on n when (`w`, `x`, `y`, `z`) is (4, 3, 3, 3).

(3) Answer whether or not `mysort` always works correctly for each case where (`w`, `x`, `y`, `z`) is (4, 2, 3, 3), (4, 3, 2, 3), (4, 3, 3, 2), and (4, 2, 3, 2).

(4) Give a necessary and sufficient condition on `w`, `x`, `y`, and `z` that guarantees `mysort` to always work correctly.

Problem 3

Let $\Sigma_1 = \{\mathtt{a}, \mathtt{b}\}$ and $\Sigma_2 = \{\mathtt{t}, \mathtt{f}\}$. For a word $w \in \Sigma_1^*$, we write $|w|$ for the length of w. We also write ϵ for the empty word (i.e., the word of length 0). For a word $w \in \Sigma_1^*$, we define the function $f_w \in \Sigma_1^* \to \Sigma_2^*$ by:

$$f_w(w') = \{x_1 \cdots x_{|w'|} \in \Sigma_2^* \mid x_i = \begin{cases} \mathtt{t} & \text{if } w' = uwv \text{ for some } u, v \in \Sigma_1^* \text{ such that } |u| = i - 1 \\ \mathtt{f} & \text{otherwise} \end{cases} \text{ for each } i \in \{1, \ldots, |w'|\}\}.$$

In other words, $f_w(w')$ is the word obtained from w' by replacing the start position of each subword that matches w with $\mathtt{t}$ and any other position with $\mathtt{f}$. For example, $f_{\mathtt{aa}}(\mathtt{baaab}) = \mathtt{fttff}$ and $f_\epsilon(\mathtt{abbab}) = \mathtt{ttttt}$. Furthermore, we extend the function f_w to the function f_w^* that maps a language over Σ_1 to a language over Σ_2 by the following definition:

$$f_w^*(L) = \{f_w(w') \mid w' \in L\}.$$

For example, $f_{\mathtt{ab}}^*(\{(\mathtt{abb})^n \mid n \geq 0\}) = \{(\mathtt{tff})^n \mid n \geq 0\}$.

Answer the following questions.

(1) Compute $f_{\mathtt{aba}}(\mathtt{babababa})$.

(2) Express $f_{\mathtt{aba}}^*(\Sigma_1^*)$ by using a regular expression.

(3) Suppose that a word $w \in \Sigma_1^*$ (where $|w| > 0$) and a deterministic finite automaton $\mathcal{A} = (Q, \Sigma_1, \delta, q_0, F)$ are given, and that L is the language accepted by $\mathcal{A}$. Here, Q, δ, q_0, F are respectively the set of states, the transition function, the initial state, and the set of accepting states of $\mathcal{A}$. Assume that the transition function $\delta \in Q \times \Sigma_1 \to Q$ is total. Give a nondeterministic finite automaton that accepts $f_w^*(L)$, with a brief explanation. You may use ϵ-transitions.

(4) If the following proposition is true, then give a proof sketch (it suffices to show a pushdown automaton that accepts $f_w^*(L)$ or a context-free grammar that generates $f_w^*(L)$, with a brief explanation). Otherwise, give a counterexample.
Proposition: "For every word $w \in \Sigma_1^*$, if $L \subseteq \Sigma_1^*$ is a context-free language, then $f_w^*(L)$ is also a context-free language."

Problem 4

Answer the following questions on computer architecture.

(1) When the instruction j may use the result generated by the instruction i, we say there is a data dependency from the instruction j to the instruction i, and write $j \rightarrow i$. Give all data dependencies in the program below. The behavior of each instruction is described as a comment (the description after #) in the program. The processor which executes the program has 32 registers from `x0` to `x31`, and `x0` is the zero register that always keeps the value 0. In the comment, we represent a memory access to the address `addr` on the memory as "`memory[addr]`".

```
instruction 0)        addi x3, x0, 64     # x3 <- x0 + 64
instruction 1)        addi x4, x0, 0      # x4 <- x0 + 0
instruction 2)        addi x5, x0, 0      # x5 <- x0 + 0
instruction 3) Loop:  lw x6, 0 (x4)       # x6 <- memory[x4 + 0]
instruction 4)        add x5, x5, x6      # x5 <- x5 + x6
instruction 5)        addi x4, x4, 4      # x4 <- x4 + 4
instruction 6)        blt x4, x3, Loop    # if x4 < x3, goto Loop
instruction 7)        sw x5, 4096 (x0)    # memory[x0 + 4096] <- x5
```

(2) Consider a 5-stage pipeline processor that issues up to one instruction per clock cycle. The processor consists of 5 stages: instruction fetch (IF) stage, instruction decode and register fetch (ID) stage, execution (EX) stage, memory access (MA) stage, and register write back (WB) stage. The bit width of each register is 32. The processor has the instruction and data memories that can be accessed in one clock cycle, and load-word `lw` and store-word `sw` instructions do not stall on the MA stage. If there is a load-use data hazard for `lw` instruction, the IF, ID, and EX stages are stalled for one clock cycle. The branch instruction `blt` (branch if less than) stalls the IF and ID stages until the branch result is determined in the EX stage. Thus, the processor does not fetch subsequent instructions for two clock cycles after a branch instruction is fetched. Execution results in the EX stage and load results in the MA stage are properly forwarded to the EX stage.

Explain what the load-use data hazard is. Explain also how load-use data hazards occur when the program in question (1) is executed on the processor.

(3) Calculate the number of clock cycles required for the execution of the program in question (1) on the processor in question (2). Calculate also the average IPC (instructions per cycle) up to two places of decimals.

(4) Using the program in question (1) and the processor in question (2) as an example, explain the mechanism and role of dynamic branch prediction.