I. IPv4 address

1. Please give three implementations of the following function, one uses `getnameinfo`, one uses `inet_ntop`, `ntohs` and one uses only `ntohs`:

   `void print_sin(const struct sockaddr_in *addr);`

   given an IPv4 address struct, print it as "x.x.x.x:port"

   Note: For simplicity, you do NOT need to take care of error handling.

```
void print_sin1(const struct sockaddr_in *addr) {
    char host[100], port[100];
    getnameinfo(addr, sizeof(*addr), host, sizeof(host),
            port, sizeof(port), NI_NUMERICHOST |
            NI_NUMERICSERV);
    printf("%s:%s\n", host, port);
}
void print_sin2(const struct sockaddr_in *addr) {
    char host[100];
    inet_ntop(addr->sin_family, &addr->sin_addr, host,
            sizeof(host));
    printf("%s:%u\n", host, ntohs(addr->sin_port));
}
void print_sin3(const struct sockaddr_in *addr) {
    unsigned char *p = (unsigned char *) &addr->sin_addr;
    printf("%u.%u.%u.%u:%u\n", p[0], p[1], p[2], p[3],
            ntohs(addr->sin_port));
}
```

2. Assume we initialize addr as following:

```
struct sockaddr_in addr;
memset(&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = 0x13784293;
addr.sin_port = 12387;
```

   What's the output of `print_sin`? Why the port number is not 12387?

   147.66.120.19:25392

   Because the assign will store 12387 to sin_port in little-endian order, while socket related functions parse it as big-endian.

   12387 = 0x3063 = 63 30 (stored in little-endian) = 0x6330 (loaded in big-endian) = 25392

II.Socket programming

Assume we want to write a tick program which prints a "BEEP" in console every second . If there is any client connected, the BEEP will be sent to client instead of being printed on server. When client closes the connection, "BEEP" should be printed in console again. Here is parts of the program:

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <errno.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>
#include "csapp.h"

void handler(int sig) {
    Write(1, "BEEP\n", 5);
    Alarm(1);
}

// This function will block and return after the client
// close the connection
void wait_disconnect(int fd) {
    char c;
    while (read(fd, &c, 1) > 0 || errno == EINTR)
        ;
}

int main() {
    int listenfd, connfd;
    Close(2); // You are not allowed to use stderr
    Signal(SIGALRM, handler);
    Alarm(1);

    listenfd = Open_listenfd("2333");
    while (1) {
        connfd = Accept(listenfd, NULL, NULL);
        int stdout_backup = dup(1);
        Dup2(connfd, 1);
        wait_disconnect(connfd);
        Close(connfd);
        Dup2(stdout_backup, 1);
        Close(stdout_backup);
    }
    exit(0);
}
```

1. Complete the program
2. Please write a command to test your program, without writing any client program.