General Problems

Q1. Your friends are starting a security company that needs to obtain licenses for n different pieces of cryptographic software. Due to regulations, they can only obtain these licenses at the rate of at most one per month. Each license is currently selling for a price of $100. However, they are all becoming more expensive according to exponential growth curves: in particular, the cost of license j increases by a factor of $r_j > 1$ each month, where r_j is a given parameter. This means that if license j is purchased t months from now, it will cost $100r_j^t$. We will assume that all the price growth rates are distinct; that is, $r_i \neq r_j$ for licenses $i \neq j$ (even though they start at the same price of $100).
The question is: Given that the company can only buy at most one license a month, in which order should it buy the licenses so that the total amount of money it spends is as small as possible?

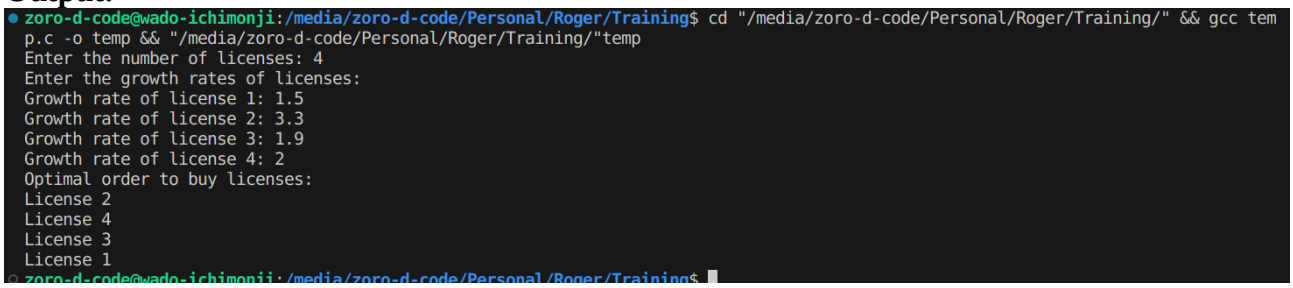
Solution:
To tackle this problem efficiently, we adopt a greedy strategy, prioritizing licenses based on their growth rates in descending order. This approach minimizes overall costs by acquiring licenses with the highest growth potential first.

Code:

```
#include <stdio.h>
#include <stdlib.h>

struct License {
    int index;
    double growth_rate;
};

// Function to compare growth rates for sorting
int compare(const void *a, const void *b) {
    struct License *license1 = (struct License *)a;
    struct License *license2 = (struct License *)b;
    return (license2->growth_rate > license1->growth_rate) - (license2->growth_rate < license1->growth_rate);
}

// Function to find the optimal order to buy licenses
void findOptimalOrder(double growth_rates[], int n) {
    struct License licenses[n];
    for (int i = 0; i < n; i++) {
        licenses[i].index = i + 1;
        licenses[i].growth_rate = growth_rates[i];
    }
    qsort(licenses, n, sizeof(struct License), compare);
    printf("Optimal order to buy licenses:\n");
    for (int i = 0; i < n; i++) {
        printf("License %d\n", licenses[i].index);
    }
}

int main() {
    int n;
    printf("Enter the number of licenses: ");
```

```
    scanf("%d", &n);
    if (n <= 0) {
        printf("Invalid number of licenses!\n");
        return 1;
    }
    double growth_rates[n];
    printf("Enter the growth rates of licenses:\n");
    for (int i = 0; i < n; i++) {
        printf("Growth rate of license %d: ", i + 1);
        scanf("%lf", &growth_rates[i]);
        if (growth_rates[i] <= 1) {
            printf("Growth rate must be greater than 1!\n");
            return 1;
        }
    }
    findOptimalOrder(growth_rates, n);
    return 0;
}
```

Output:



Q2. Suppose you are given an array A with n entries, with each entry holding a distinct number. You are told that the sequence of values A[1],A[2],...,A[n] is unimodal. That is, for some index p between 1 and n, the values in the array entries increase up to position p in A and then decrease the remainder of the way until position n. (So if you were to draw a plot with the array position j on the x-axis and the value of the entry A[j] on the y-axis, the plotted points would rise until x-value p, where they'd achieve their maximum value, and then fall from there on). You'd like to find the "peak entry" p without having to read the entire array - in fact, by reading as few entries of A as possible. Show how to find the entry p by reading at most O(log n) entries of A.

Solution:

To efficiently find the peak entry p in a unimodal array A of size n by examining at most $O(\log n)$ entries, we can utilize a modified binary search algorithm. The approach involves checking the middle element $A[\text{mid}]$ and comparing it with its neighboring elements. Based on this comparison, we determine whether the peak lies in the left or right half of the array and continue the search accordingly.

Code:

```
#include <stdio.h>

// Function to find the peak element in a unimodal array
int find_peak(int A[], int n) {
```

```
    int left = 0;
    int right = n - 1;

    while (left < right) {
        int mid = left + (right - left) / 2;

        if ((mid == 0 || A[mid] >= A[mid - 1]) && (mid == n - 1 || A[mid] >= A[mid + 1])) {
            return mid; // Peak element found
        }

        if (mid > 0 && A[mid - 1] > A[mid]) {
            right = mid - 1; // Peak is in the left half
        }
        else {
            left = mid + 1; // Peak is in the right half
        }
    }

    return left; // Return the index of the peak element
}

int main() {
    int n;
    printf("Enter the size of the unimodal array: ");
    scanf("%d", &n);

    int A[n];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < n; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &A[i]);
    }

    int peak_index = find_peak(A, n);
    printf("The peak element is at index %d with value %d\n", peak_index, A[peak_index]);
    return 0;
}
```

Output:

```
zoro-d-code@wado-ichimonji:/media/zoro-d-code/Personal/Roger/Training$ cd "/media/zoro-d-code/Personal/Roger/Training/" && gcc tem
p.c -o temp && "/media/zoro-d-code/Personal/Roger/Training/"temp
Enter the size of the unimodal array: 10
Enter the elements of the array:
Element 1: 5
Element 2: 10
Element 3: 15
Element 4: 20
Element 5: 25
Element 6: 30
Element 7: 35
Element 8: 40
Element 9: 45
Element 10: 50
The peak element is at index 9 with value 50
```