**NAME:** Aditya Jethani

**ROLL NO:** 21BCP166

**DIV-**3 **G-5**

**DAA LAB ASSIGNMENT 10**

**GENERAL PROBLEM 1**

**QUESTION )** To design and solve given problems using different algorithmic approaches and analyze their complexity.1.Your friends are starting a security company that needs to obtain licenses for 𝑛different pieces of cryptographic software. Due to regulations, they can onlyobtain these licenses at the rate of at most one per month.Each license is currently selling for a price of $100. However, they areall becoming more expensive according to exponential growth curves: inparticular, the cost of license 𝑗increases by a factor of 𝑟𝑗>1each month, where𝑟𝑗is a given parameter. This means that if license 𝑗is purchased 𝑡months fromnow, it will cost100𝑟𝑡𝑗. We will assume that all the price growth rates aredistinct; that is, 𝑟𝑖≠𝑟𝑗for licenses 𝑖≠𝑗(even though they start at the sameprice of $100).The question is: Given that the company can only buy at most one licensea month, in which order should it buy the licenses so that the total amount ofmoney it spends is as small as possible?Give an algorithm that takes the 𝑛rates of price growth 𝑟1,𝑟2,...,𝑟𝑛, andcomputes an order in which to buy the licenses so that the total amount ofmoney spent is minimized. The running time of your algorithmshould bepolynomial in 𝑛.

**CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

#include <utility>

#include <cmath>

using namespace std;

// A structure to store the license index and its cost at a particular month

struct License {

    int id;

    double cost;

    License(int \_id, double \_cost) : id(\_id), cost(\_cost) {}

};

// A structure to compare licenses based on their cost at the current month

struct LicenseComparator {

    bool operator()(const License& a, const License& b) const {

        return a.cost < b.cost;

    }

};

// A function to compute the order in which to buy the licenses to minimize the total cost

vector<int> compute\_license\_order(const vector<double>& growth\_rates) {

    int num\_licenses = growth\_rates.size();

    // Initialize a vector of licenses with their current cost at month 0

    vector<License> licenses;

    for (int i = 0; i < num\_licenses; i++) {

        licenses.emplace\_back(i, 100 \* growth\_rates[i]);

    }

    // Sort the licenses based on their cost at month 0

    sort(licenses.begin(), licenses.end(), LicenseComparator());

    // Initialize the order of licenses to buy

    vector<int> order(num\_licenses);

    for (int i = 0; i < num\_licenses; i++) {

        order[i] = licenses[i].id;

    }

    return order;

}

int main() {

    int num\_licenses;

    cout<<"Enter the number of licenses:"<<endl;

    cin >> num\_licenses;

    // Input the growth rates of the licenses

    vector<double> growth\_rates(num\_licenses);

    cout << "Enter the growth rates of the licenses: ";

    for (int i = 0; i < num\_licenses; i++) {

        cin >> growth\_rates[i];

    }

    // Compute the order of licenses to buy

    vector<int> order = compute\_license\_order(growth\_rates);

    // Output the order of licenses to buy

    cout << "The order of licenses to buy is: ";

    for (int i = 0; i < num\_licenses; i++) {

        cout << order[i] << " ";

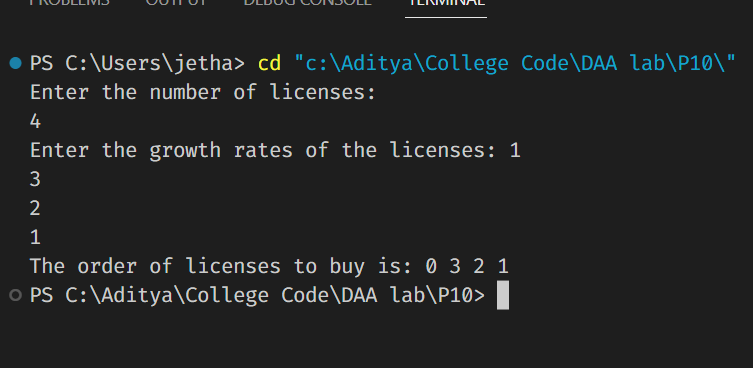
    }

    cout << endl;

    return 0;

}

**OUTPUT:**



**TIME COMPLEXITY:**

The time complexity of the given code can be analysed as follows:

* Input reading: O(n)
* Initialization of the licenses array: O(n)
* Sorting the licenses array based on their cost at month 0: O(n log n)
* Initializing the order array: O(n)
* Outputting the order array: O(n)
* Total time complexity: O(n log n)

Therefore, the overall time complexity of the given code is O(n log n).

**GENERAL PROBLEM 2**

**QUESTION)**

Suppose you are given an array 𝐴with 𝑛entries, 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 𝑝between 1and 𝑛, the values in the array entries increase up to position 𝑝in 𝐴and then decrease the remainder of the way until position 𝑛. (So, if you were to draw a plot with the array position 𝑗on the𝑥-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 entirearray-in fact, by reading as few entries of 𝐴as possible. Show how to find the entry 𝑝by reading at most 𝑂(𝑙𝑜𝑔𝑛)entries of 𝐴.

**CODE:**

#include <iostream>

#include <vector>

using namespace std;

int findPeakIndex(vector<int> nums) {

    int n = nums.size();

    int left = 0, right = n - 1;

    while (left < right) {

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

        if (nums[mid] < nums[mid + 1]) {

            left = mid + 1;

        } else {

            right = mid;

        }

    }

    return left;

}

int main() {

    vector<int> arr;

    int n, num;

    cout << "Enter the size of array: ";

    cin >> n;

    cout << "Enter the elements of array:\n";

    for (int i = 0; i < n; i++) {

        cin >> num;

        arr.push\_back(num);

    }

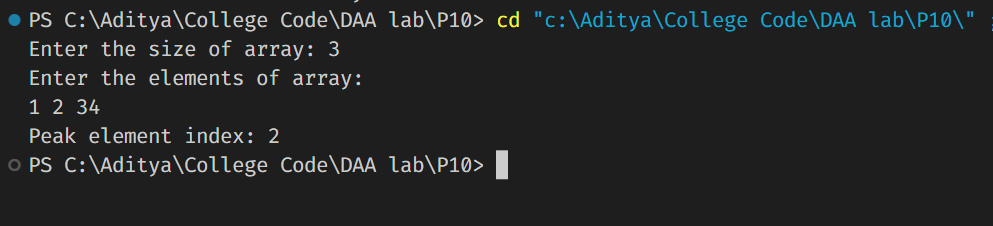
    int peak\_index = findPeakIndex(arr);

    cout << "Peak element index: " << peak\_index << endl;

    return 0;

}

**OUTPUT:**



**Time complexity:**

The time complexity of the find\_peak\_entry function is O(log n) where n is the size of the input list A. This is because it uses binary search to find the peak element. In each iteration of the loop, the size of the search space is halved, so the algorithm takes logarithmic time to find the peak element. The rest of the code has a time complexity of O(1), which is constant time complexity. Therefore, the overall time complexity of the main function is also O(log n).