Project

Anh Huynh

Task 1

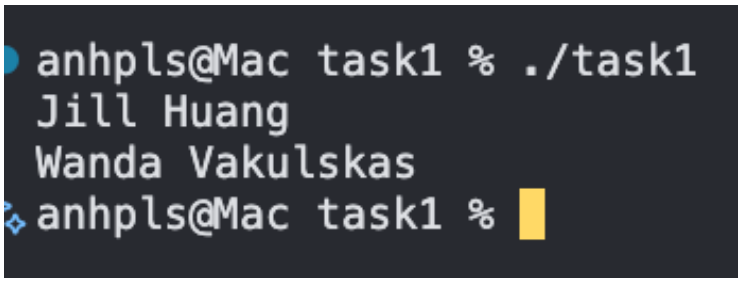
> You're working on software that analyzes sports players. Following are two arrays of players of different sports:
>
> ```
> basketball_players = [
>    {first_name: "Jill", last_name: "Huang", team: "Gators"},
>    {first_name: "Janko", last_name: "Barton", team: "Sharks"},
>    {first_name: "Wanda", last_name: "Vakulskas", team: "Sharks"},
>    {first_name: "Jill", last_name: "Moloney", team: "Gators"},
>    {first_name: "Luuk", last_name: "Watkins", team: "Gators"}
> ]
>
> football_players = [
>    {first_name: "Hanzla", last_name: "Radosti", team: "32ers"},
>    {first_name: "Tina", last_name: "Watkins", team: "Barleycorns"},
>    {first_name: "Alex", last_name: "Patel", team: "32ers"},
>    {first_name: "Jill", last_name: "Huang", team: "Barleycorns"},
>    {first_name: "Wanda", last_name: "Vakulskas", team: "Barleycorns"}
> ]
> ```
>
> If you look carefully, you'll see that some players participate in more than one sport. Jill Huang and Wanda Vakulskas play both basketball *and* football.
>
> You are to write a function that accepts two arrays of players and returns an array of the players who play in *both* sports. In this case, that would be:
>
> ```
> ["Jill Huang", "Wanda Vakulskas"]
> ```
>
> While there are players who share first names and players who share last names, we can assume there's only one person who has a particular *full* name (meaning first *and* last name).
>
> We can use a nested-loops approach, comparing each player from one array against each player from the other array, but this would have a runtime of O(N∗M).
>
> **Your job is to optimize the function so that it can run just O(N+M).**

```cpp
#include <iostream>
#include <string>
#include <unordered_set>
#include <vector>
using namespace std;
```

```cpp

// structure for each person's info
struct Person {
    string first_name;  // first name
    string last_name;   // last name
    string team;    // what team they're on
};

// function to find players who appear in both lists
vector<string> playsBoth(const vector<Person>& a, const vector<Person>& b){
    unordered_set<string> fullNames;   // set to store all full names from list a

    for (const auto& x : a){    // for each person in first list
        fullNames.insert(x.first_name + " " + x.last_name); // put "first + last" into set
    }

    vector<string> result;  // for names that are in both lists

    for (const auto& x: b){ // for each person in second list
        string fullNames2 = x.first_name + " " + x.last_name;   // "first + last"
        if (fullNames.count(fullNames2)){   // check if this name existed in the first list
            result.push_back(fullNames2);   // if yes then add to result list
        }
    }

    return result;  // return the list for names that are in both lists
}


int main(){
    // list of basketball players using the Person struct
    vector<Person> basketball = {
        {"Jill", "Huang", "Gators"},
        {"Janko", "Barton", "Sharks"},
        {"Wanda", "Vakulskas", "Sharks"},
        {"Jill", "Moloney", "Gators"},
        {"Luuk", "Watkins", "Gators"}
    };

    // list of football players
    vector<Person> football = {
        {"Hanzla", "Radosti", "32ers"},
        {"Tina", "Watkins", "Barleycorns"},
        {"Alex", "Patel", "32ers"},
        {"Jill", "Huang", "Barleycorns"},
        {"Wanda", "Vakulskas", "Barleycorns"}
    };

    vector<string> both = playsBoth(basketball, football);  // list to store players that play in both sports

```

```
    // print each name in both list to show which players
    for (const string& x : both){
        cout << x << endl;
    }

}
```

Output:



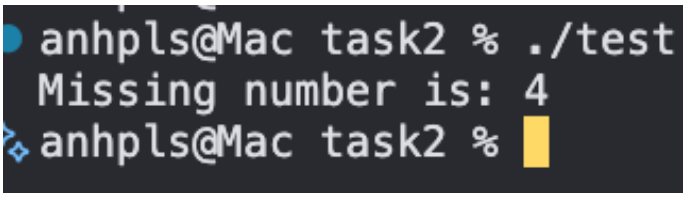
Task 2

> You're writing a function that accepts an array of distinct integers from 0, 1, 2, 3...up to N. However, the array will be missing one integer, and your function is to *return the missing one.*
>
> For example, this array has all the integers from 0 to 6, but is missing the 4:
>
> ```
> [2, 3, 0, 6, 1, 5]
> ```
>
> Therefore, the function should return 4.
>
> The next example has all the integers from 0 to 9, but is missing the 1:
>
> ```
> [8, 2, 3, 9, 4, 7, 5, 0, 6]
> ```
>
> In this case, the function should return the 1.
>
> Using a nested-loops approach would take up to O(N2).
>
> **Your job is to optimize the code so that it has a runtime of O(N).**

```
#include <iostream>

using namespace std;


int returnMissingNum(const vector<int>& listOfNums){
    int n = listOfNums.size();  // size of array 0-n
    int totalSum = n * (n + 1) / 2; // sum of all numbers from 0 to n

```

```
    int testSum = 0;
    for (int x : listOfNums){
        testSum += x;   // sum all numbers inside the array
    }

    return totalSum - testSum; // this gives the difference between the two arrays (missing number)
}



int main(){
    vector<int> list = {2, 3, 0, 6, 1, 5};  // create populated array for testing
    cout << "Missing number is: " << returnMissingNum(list) << endl;    // output

    vector<int> list2 ={8, 2, 3, 9, 4, 7, 5, 0, 6}; // second example
    cout << "Missing number is: " << returnMissingNum(list2) << endl; // output
}
```

Output:



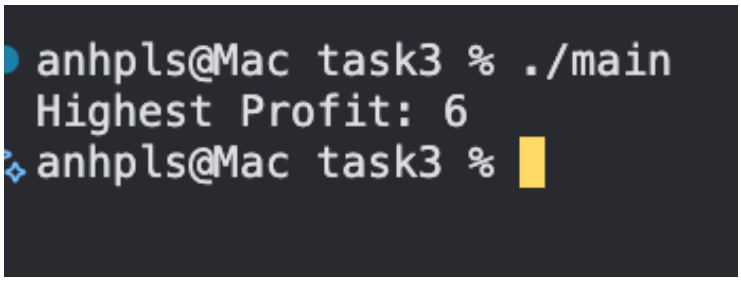
Task 3

> You're working on some more stock-prediction software. The function you're writing accepts an array of predicted prices for a particular stock over the course of time.
>
> For example, this array of seven prices:
>
> ```
> [10, 7, 5, 8, 11, 2, 6]
> ```
>
> predicts that a given stock will have these prices over the next seven days. (On Day 1, the stock will close at $10; on Day 2, the stock will close at $7; and so on.)
>
> Your function should calculate the greatest profit that could be made from a single "buy" transaction followed by a single "sell" transaction.
>
> In the previous example, the most money could be made if we bought the stock when it was worth $5 and sold it when it was worth $11. This yields a profit of $6 per share.
>
> Note that we could make even more money if we buy and sell multiple times, but for now, this function focuses on the most profit that could be made from just *one* purchase followed by *one* sale.
>
> Now, we could use nested loops to find the profit of every possible buy and sell combination. However, this would be $O(N^2)$ and too slow for our hotshot trading platform.

> **Your job is to optimize the code so that the function clocks in at just O(N).**

```cpp
#include <iostream>
using namespace std;

int highestReturn(const vector<int>& stocks){
    if (stocks.empty()) return 0;   // edge: handle an empty list

    int minPrice = stocks[0];   // assume first day is smallest price
    int highestProfit = 0;  // initalize to 0 profit

    for(int stockPrice : stocks){   // for each stock in the stocks list
        int currProfit = stockPrice - minPrice; // for each day, take difference between that day's stock price and minimum stock price

        if (currProfit > highestProfit){    // if this profit is more profitable
            highestProfit = currProfit; // set as new highest profit
        }

        if (stockPrice < minPrice){     // if the stockPrice is lower than the smallest stock price
            minPrice = stockPrice;  // set as new smallest stock price
        }
    }

    return highestProfit;   // return highest profit

}


int main(){
    vector<int> stocks = {10, 7, 5, 8, 11, 2, 6};   // test stocks list
    cout << "Highest Profit: " << highestReturn(stocks) << endl;    // output
}
```

Output:



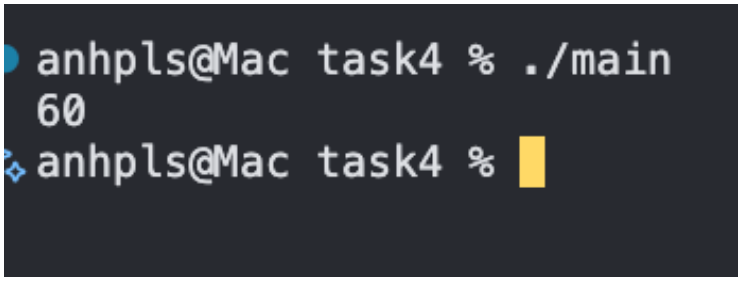
Task 4

> You're writing a function that accepts an array of numbers and computes the highest product of any two numbers in the array. At first glance, this is easy, as we can just find the two greatest numbers and multiply them. However, our array can contain negative numbers and look like this:
>
> ```
> [5, -10, -6, 9, 4]
> ```
>
> We could use nested loops to multiply every possible pair of numbers, but this would take O(N^2) time. **Your job is to optimize the function so that it's a speedy O(N).**

```cpp
#include <iostream>
#include <climits>
using namespace std;


int biggestProduct(const vector<int>& nums) {

    int maxA = INT_MIN, maxB = INT_MIN;   // 2 largest numbers
    int minA = INT_MAX, minB = INT_MAX;   // 2 smallest numbers

    // iterate through the list
    for (int x : nums) {
        //  update largest numbers
        if (x > maxA) {     // if x is bigger than the current biggest
            maxB = maxA;    // move old biggest to second biggest
            maxA = x;       // set new biggest
        }
        else if (x > maxB) {    // otherwise it's the second biggest
            maxB = x;           // set to second biggest
        }

        // update smallest numbers
        if (x < minA) {     // if x is smaller than the current smallest
            minB = minA;    // move old smallest to second smallest
            minA = x;       // set new smallest
        }
        else if (x < minB) {    // otherwise it's the second smallest
            minB = x;       // set to second smallest
        }
    }

    int prodLargest = maxA * maxB;      // multiply two biggest
    int prodSmallest = minA * minB;     // multiply two smallest

    return max(prodLargest, prodSmallest);  // return whichever product is bigger
}

int main(){
    vector<int> list = {5, -10, -6, 9, 4};  // test list
```

```
    cout << biggestProduct(list) << endl;   // output
}
```

Output:



Task 5

> You're creating software that analyzes the data of body temperature readings taken from hundreds of human patients. These readings are taken from healthy people and range from 97.0 degrees Fahrenheit to 99.0 degrees Fahrenheit. An important point: within this application, *the decimal point never goes beyond the tenth place.*
>
> Here's a sample array of temperature readings:
>
> ```
> [98.6, 98.0, 97.1, 99.0, 98.9, 97.8, 98.5, 98.2, 98.0, 97.1]
> ```
>
> You are to write a function that sorts these readings from lowest to highest.
>
> Using a classic sorting algorithm such as Quicksort would take O(NlogN). However, in this case, writing a faster sorting algorithm is possible.
>
> Yes, that's right. Even though you've learned that the fastest sorts are O(NlogN), this case is different. Why? In this case, there are limited possibilities for the readings. In such a case, we can sort these values in O(N). It may be N multiplied by a constant, but that's still considered O(N).

```
#include <iostream>
#include <vector>
using namespace std;

vector<double> sortTemp(const vector<double>& temps) {

    const int RANGE = 21;                    // total possible temps from 97.0 to 99.0
    vector<vector<double>> buckets(RANGE);   // 21 buckets to hold each temp group

    // drop temps directly into their matching bucket
    for (double t : temps) {
        int index = int((t - 97.0) * 10);    // temp like 97.3 becomes index 3
        buckets[index].push_back(t);         // throw the value into that bucket
    }

```

```cpp
    vector<double> sortedList;                  // final sorted temps

    for (int i = 0; i < RANGE; i++) {           // go bucket by bucket in order
        for (double t : buckets[i]) {           // grab everything inside the bucket
            sortedList.push_back(t);            // push it into the sorted list
        }
    }

    return sortedList;                          // return the sorted temps
}

int main() {
    vector<double> data = {98.6, 98.0, 97.1, 99.0, 98.9, 97.8, 98.5, 98.2, 98.0, 97.1}; // example data
    vector<double> sorted = sortTemp(data);     // created the sorted list

    for (double x : sorted) {                   // print sorted temps
        cout << x << " ";                       // print each temp with a space
    }
}
```

Output

```
anhpls@Anhs-MacBook-Pro task5 % g++ -std=c++17 task5.cpp -o main
anhpls@Anhs-MacBook-Pro task5 % ./main
97.1 97.1 97.8 98 98 98.2 98.6 98.5 98.9 99 %
```

Task 6

> You're writing a function that accepts an array of unsorted integers and returns the length of the *longest consecutive sequence* among them. The sequence is formed by integers that increase by 1. For example, in the array:
>
> ```
> [10, 5, 12, 3, 55, 30, 4, 11, 2]
> ```
>
> the longest consecutive sequence is 2-3-4-5. These four integers form an increasing sequence because each integer is one greater than the previous one. While there's also a sequence of 10-11-12, it's only a sequence of three integers. In this case, the function should return 4, since that's the length of the *longest* consecutive sequence that can be formed from this array.
>
> One more example:
>
> ```
> [19, 13, 15, 12, 18, 14, 17, 11]
> ```
>
> This array's longest sequence is 11-12-13-14-15, so the function would return 5.

> **Your job is to optimize the function so that it takes O(N) time.**

```cpp
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;

// function to find the largest consecutive sequence
int consecutive(vector<int>& list){
    unordered_set<int> a;   // init a hash set to store all nums

    for (int x : list){     // loop through the vector
        a.insert(x);        // insert each number into the set
    }

    int longest = 0;        // init the longest sequence to 0

    for (int x : list){     // check each number in the list
        if (a.find(x-1) == a.end()){    // only start counting if x-1 is not in the set
            int curr = x;   // current number in the sequence
            int length = 1; // length of the current sequence

            while (a.find(curr + 1) != a.end()){    // while the next number exists
                curr++;     // move to the next consecutive number
                length++;   // increase the sequence length
            }
            longest = max(longest, length); // update the longest if this sequence is bigger
        }
    }
    return longest; // return the longest sequence length
}

int main(){
    vector<int> list = {10, 5, 12, 3, 55, 30, 4, 11, 2};    // first test list
    cout << consecutive(list) << endl;  // should print 4

    vector<int> list2 = {19, 13, 15, 12, 18, 14, 17, 11};   // second test list
    cout << consecutive(list2) << endl; // should print 5
}
```

Output

```
anhpls@Anhs-MacBook-Pro task6 % g++ -std=c++17 task6.cpp -o main
anhpls@Anhs-MacBook-Pro task6 % ./main
4
5
```

Video Link:

https://youtu.be/dYYB-7yYxlM