Status	Finished
Started	Monday, 30 September 2024, 3:44 PM
Completed	Monday, 30 September 2024, 5:05 PM
Duration	1 hour 21 mins
Marks	6.87/9.00
Grade	**7.63** out of 10.00 (**76.3**%)

Question 1

Correct

Mark 1.00 out of 1.00

The prices of all cars of a car shop have been saved as an array called N. Each element of the array N is the price of each car in shop. A person, with the amount of money k want to buy as much cars as possible.

Request: Implement function

buyCar(int* nums, int length, int k);

Where `nums` is the array N, `length` is the size of this array and `k` is the amount of money the person has. Find the maximum cars this person can buy with his money, and return that number.

Example:

`nums=[90, 30, 20, 40, 50]; k=90;`

The result is 3, he can buy the cars having index 1, 2, 3 (first index is 0).

Note: The library `iostream`, `'algorithm'` and `using namespace std` have been used. You can add other functions but you are not allowed to add other libraries.

For example:

Test	Result

`int nums[] = {90,30,40,90,20};`
`int length = sizeof(nums)/sizeof(nums[0]);`
`cout << buyCar(nums, length, 90) << "\n";`
	3

Answer: (penalty regime: 0 %)

Reset answer

```cpp
int buyCar(int* nums, int length, int k) {
    sort(nums, nums + length);

    int count = 0; // Counter to keep track of how many cars we can buy
    int totalCost = 0; // Sum of the car prices we've bought so far

    // Step 2: Buy cars while we have enough money
    for (int i = 0; i < length; i++) {
        if (totalCost + nums[i] <= k) {
            totalCost += nums[i]; // Add the price of the car to the total cost
            count++; // Increase the count of cars bought
        } else {
            break; // Stop if we can't afford the next car
        }
    }

    return count;
}
```

	Test	Expected	Got	
✓	`int nums[] = {90,30,40,90,20};` `int length = sizeof(nums)/sizeof(nums[0]);` `cout << buyCar(nums, length, 90) << "\n";`	3	3	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Partially correct

Mark 0.90 out of 1.00

Given an array of integers.
Your task is to implement a function with the following prototype:

```
bool consecutiveOnes(vector<int>& nums);
```

The function returns if all the `1`s appear consecutively in `nums`. If `nums` does not contain any elements, please return `true`

Note:
- The `iostream` and `vector` libraries have been included and `namespace std` are being used. No other libraries are allowed.
- You can write helper functions.

For example:

Test	Result

`vector<int> nums {0, 1, 1, 1, 9, 8};`
`cout << consecutiveOnes(nums);`
	1

Answer: (penalty regime: 0 %)

Reset answer

```
bool consecutiveOnes(vector<int>& nums) {
    // STUDENT ANSWER
        if (nums.empty()) {
        return true;
    }

    bool foundOne = false;
    bool encounteredZeroAfterOne = false;

    for (int num : nums) {
        if (num == 1) {
            if (encounteredZeroAfterOne) {
                return false;
            }
            foundOne = true;
        } else if (num == 0) {
            if (foundOne) {
                encounteredZeroAfterOne = true;
            }
        }
    }

    return true;
}
```

	Test	Expected	Got	

✓	`vector<int> nums {0, 1, 1, 1, 9, 8};`
	`cout << consecutiveOnes(nums);`
		1	1	✓

✓	`vector<int> nums {};`
	`cout << consecutiveOnes(nums);`
		1	1	✓

Your code failed one or more hidden tests.

Partially correct

Marks for this submission: 0.90/1.00.

Question **3**

Correct

Mark 1.00 out of 1.00

Given an array of integers.
Your task is to implement a function with following prototype:

```
int equalSumIndex(vector<int>& nums);
```

The function returns the smallest index `i` such that the sum of the numbers to the left of `i` is equal to the sum of the numbers to the right.
If no such index exists, return `-1`.

Note:

- The `iostream` and `vector` libraries have been included and `namespace std` is being used. No other libraries are allowed.
- You can write helper functions.

For example:

Test	Result
`vector<int> nums {3, 5, 2, 7, 6, 4};` `cout << equalSumIndex(nums);`	3

Answer: (penalty regime: 0 %)

Reset answer

```
int equalSumIndex(vector<int>& nums) {
    // STUDENT ANSWER
        int totalSum = 0;
    for (int num : nums) {
        totalSum += num;
    }

    int leftSum = 0; // Initialize left

    for (int i = 0; i < nums.size(); i++) {
        int rightSum = totalSum - leftSum - nums[i];

        if (leftSum == rightSum) {
            return i; // Return the index if equal
        }

        leftSum += nums[i];
    }

    return -1;
}
```

	Test	Expected	Got	
✓	`vector<int> nums {3, 5, 2, 7, 6, 4};` `cout << equalSumIndex(nums);`	3	3	✓
✓	`vector<int> nums {3};` `cout << equalSumIndex(nums);`	0	0	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Incorrect

Mark 0.00 out of 1.00

Given an array of strings.
Your task is to implement a function with following prototype:

int longestSublist(vector<string>& words);

The function returns the length of the longest subarray where all words share the same first letter.

Note:

- The `iostream` and `vector` libraries have been included and `namespace std` is being used. No other libraries are allowed.
- You can write helper functions.

For example:

Test	Result
`vector<string> words {"faction", "fight", "and", "are", "attitude"};` `cout << longestSublist(words);`	3

Answer: (penalty regime: 0 %)

Reset answer

```
int longestSublist(vector<string>& words) {
    // STUDENT ANSWER
        if (words.empty()) {
        return 0;
    }

    int maxLength = 1;
    int currentLength = 1;

    for (int i = 1; i < words.size(); i++) {
        // Compare
        if (words[i][0] == words[i - 1][0]) {
            currentLength++;
        } else {
            maxLength = max(maxLength, currentLength);
            currentLength = 1;
        }
    }

    maxLength = max(maxLength, currentLength);

    return maxLength
}
```

Syntax Error(s)

```
__tester__.cpp: In function 'int longestSublist(std::vector<std::__cxx11::basic_string<char> >&)':
__tester__.cpp:28:21: error: expected ';' before '}' token
   28 |     return maxLength
      |                     ^
      |                     ;
   29 | }
      | ~
```

Incorrect

Marks for this submission: 0.00/1.00.

Question **5**

Correct

Mark 1.00 out of 1.00

The array N contains positive integers (including n elements) and positive integer k (k <= n). Divide array N into sub-arrays satisfying the following rules:

- Each sub-array contains contiguous elements in array N.
- Each element in array N belongs to only one sub-array.
- Number of elements in each sub-array is less than or equal k.

Let S-value of each sub-array is the product of the largest element in this sub-array and the size of this sub-array. A way W, following these above rules, divides this array N into sub-arrays. S(W) is the sum of all S-values from all sub-arrays created by the way W. The way having the largest value S(W) is called W_{max}.

Request: Implement function

int maxSum(int* nums, int n, int k)

Where `nums` is array N, `n` is the size of array N and `k` is described above; return the result is the S(W) of the way W_{max}.

Example:

`nums[]={1,6,3,2,2,5,1}; k=3;`

The result is 35. The way W_{max} to divide the array is: {1,6,3}, {2}, {2,5,1}; the S-values of each sub-arrays is 6 * 3 = 18, 2 * 1 = 2 and 5 * 3 = 15, respectively; therefore, the sum of all S-values is 35.

Note: In this exercise, library `iostream` and `using namespace std` has been used. You can add other functions but not libraries.

For example:

Test	Result

`int arr[] = {1,6,3,2,2,5,1};`
`int length = sizeof(arr) / sizeof(arr[0]);`
`cout << maxSum(arr, length, 3) << "\n";`
	35

Answer: (penalty regime: 0 %)

Reset answer

```
int maxSum(int* nums, int n, int k) {
    int* dp = new int[n + 1]();

    for (int i = 1; i <= n; i++) {
        int maxElem = 0;

        for (int j = 1; j <= k && (i - j) >= 0; j++) {
            maxElem = max(maxElem, nums[i - j]);
            int size = j;
            int currentSValue = maxElem * size;

            dp[i] = max(dp[i], dp[i - j] + currentSValue);
        }
    }

    int result = dp[n];
    delete[] dp;
    return result;
}
```

	Test	Expected	Got	

✓	`int arr[] = {1,6,3,2,2,5,1};`
	`int length = sizeof(arr) / sizeof(arr[0]);`
	`cout << maxSum(arr, length, 3) << "\n";`
		35	35	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **6**

Partially correct

Mark 0.90 out of 1.00

Given an array of integers and an non-negative integer `k`.
Your task is to implement a function with following prototype:

int minimumAmplitude(vector<int>& nums, int k);

The function returns the minimum value of `max(nums) - min(nums)` after removing a sublist with length `k` from array `nums`.

Note:

- The `iostream`, `vector` and `climits` libraries have been included and `namespace std` is being used. No other libraries are allowed.
- You can write helper functions.

For example:

Test	Result

`vector<int> nums {9, 6, 2, 1, 10, 7};`
`int k = 2;`
`cout << minimumAmplitude(nums, k);`
	4

Answer: (penalty regime: 0 %)

Reset answer

```
int minimumAmplitude(vector<int>& nums, int k) {
    // STUDENT ANSWER
    int n = nums.size();
    if (n == 0 || k >= n) return 0; // No amplitude if removing all or more elements

    // Step 1: Calculate the initial maximum and minimum values
    int overallMax = INT_MIN;
    int overallMin = INT_MAX;

    for (int num : nums) {
        overallMax = max(overallMax, num);
        overallMin = min(overallMin, num);
    }

    // Step 2: Initialize the minimum amplitude with the overall amplitude
    int minAmplitude = overallMax - overallMin;

    // Step 3: Sliding window to find the minimum amplitude after removing sublists
    for (int i = 0; i <= n - k; i++) {
        // Determine the max and min of the remaining elements after removing nums[i] to nums[i + k - 1]
        int leftMax = INT_MIN;
        int leftMin = INT_MAX;

        int rightMax = INT_MIN;
        int rightMin = INT_MAX;

        // Find max and min for the left side
        for (int j = 0; j < i; j++) {
            leftMax = max(leftMax, nums[j]);
            leftMin = min(leftMin, nums[j]);
        }

        // Find max and min for the right side
        for (int j = i + k; j < n; j++) {
            rightMax = max(rightMax, nums[j]);
            rightMin = min(rightMin, nums[j]);
        }

        // Determine the new amplitude
        int newMax = max(leftMax, rightMax);
        int newMin = min(leftMin, rightMin);

        // Update the minimum amplitude
        if (newMax != INT_MIN && newMin != INT_MAX) {
            minAmplitude = min(minAmplitude, newMax - newMin);
        }
    }

    return minAmplitude;
}
```

	Test	Expected	Got	

✓	`vector<int> nums {9, 6, 2, 1, 10, 7};`
	`int k = 2;`
	`cout << minimumAmplitude(nums, k);`
		4	4	✓

✓	`vector<int> nums {375, 8734, 7366, 433, 1063, 371, 412, 6424, 3680, 4100};`
	`int k = 3;`
	`cout << minimumAmplitude(nums, k);`
		6053	6053	✓

Your code failed one or more hidden tests.

Partially correct

Marks for this submission: 0.90/1.00.

Question **7**

Partially correct

Mark 0.07 out of 1.00

You are given a list of integers `positions` with `n` elements (`1 ≤ n ≤ 100000`), each element represents the position of a person at equally spaced intervals of time.

Request: Implement function:

```
int steadySpeed(vector<int>& p);
```

Where `positions` is the list of position of a person. This function returns the length of the longest sublist where the person was traveling at a constant speed.

Example:

The list of position is `{5, 4, 3, 5, 4, 5, 1, 3, 5, 3}`. Therefore, the length of the longest sublist where the person was traveling at a constant speed is `4` (It is `{1, 3, 5, 3}`, with constant speed is `2`).

Note:

In this exercise, the libraries `iostream, string, cstring, climits, utility, vector, list, stack, queue, map, unordered_map, set, unordered_set, functional, algorithm` has been included and `namespace std` are used. You can write helper functions and classes. Importing other libraries is allowed, but not encouraged, and may result in unexpected errors.

For example:

Test	Result

`vector<int>positions{5,4,3,5,4,5,1,3,5,3};`
`cout << steadySpeed(positions);`
	4

`vector<int> positions{0, 3, 6, 3, 0};`
`cout << steadySpeed(positions);`
	5

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
int steadySpeed(vector<int>& positions) {
    int n = positions.size();
    if (n < 2) return n;
    int maxLength = 1;
    int currentLength = 1;
    int currentSpeed = positions[1] - positions[0];

    for (int i = 1; i < n - 1; i++) {
        int speed = positions[i + 1] - positions[i];

        if (speed == currentSpeed) {
            currentLength++;
        } else {
            maxLength = max(maxLength, currentLength + 1);
            currentLength = 1;
            currentSpeed = speed;
        }
    }

    maxLength = max(maxLength, currentLength + 1);

    return maxLength;
}
```

	Test	Expected	Got	
✗	`vector<int>positions{5,4,3,5,4,5,1,3,5,3};` `cout << steadySpeed(positions);`	4	3	✗

Some hidden test cases failed, too.

Show differences

Partially correct

Marks for this submission: 0.07/1.00.

Question **8**

Correct

Mark 1.00 out of 1.00

Given an array of integers sorted in ascending order and an integer `target`.
Your task is to implement a function with following prototype:

int sumLessThanTarget(vector<int>& nums, int target);

The function returns the largest sum of the pair of the numbers in `nums` whose sum is less than `target`.
The testcases ensure that a solution exists.

Note:

- The `iostream`, `vector` and `climits` libraries have been included and `namespace std` is being used. No other libraries are allowed.
- You can write helper functions.

For example:

Test	Result

`vector<int> nums {1, 2, 3, 5, 6, 9};`
`int target = 7;`
`cout << sumLessThanTarget(nums, target);`
	6

Answer: (penalty regime: 0 %)

Reset answer

```
int sumLessThanTarget(vector<int>& nums, int target) {
    // STUDENT ANSWER
    int left = 0;
    int right = nums.size() - 1;
    int maxSum = 0; // Initialize maximum sum found

    while (left < right) {
        int currentSum = nums[left] + nums[right];

        if (currentSum < target) {
            // Update maxSum if the current sum is less than the target
            maxSum = max(maxSum, currentSum);
            left++; // Move the left pointer to the right
        } else {
            // If currentSum is greater than or equal to target, move the right pointer to the left
            right--;
        }
    }

    return maxSum;
}
```

	Test	Expected	Got	

✓	`vector<int> nums {1, 2, 3, 5, 6, 9};`
	`int target = 7;`
	`cout << sumLessThanTarget(nums, target);`
		6	6	✓

✓	`vector<int> nums {18392640, 447224685};`
	`int target = 765618120;`
	`cout << sumLessThanTarget(nums, target);`
		465617325	465617325	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 9

Correct

Mark 1.00 out of 1.00

Given an array of integers `nums` and a two-dimension array of integers `operations`.
Each operation in `operations` is represented in the form `{L, R, X}`. When applying an operation, all elements with index in range `[L, R]` (include `L` and `R`) increase by `X`.
Your task is to implement a function with following prototype:

vector<int> updateArrayPerRange(vector<int>& nums, vector<vector<int>>& operations);

The function returns the array after applying all operation in `operations`.

Note:

- The `iostream`, and `vector` libraries have been included and `namespace std` is being used. No other libraries are allowed.
- You can write helper functions.

For example:

Test	Result

`vector<int> nums {13, 0, 6, 9, 14, 16};`
`vector<vector<int>> operations {{5, 5, 16}, {3, 4, 0}, {0, 2, 8}};`
`printVector(updateArrayPerRange(nums, operations));`
	`[21, 8, 14, 9, 14, 32]`

Answer: (penalty regime: 0 %)

Reset answer

```
vector<int> updateArrayPerRange(vector<int>& nums, vector<vector<int>>& operations) {
    // STUDENT ANSWER
        int n = nums.size();
    vector<int> diff(n + 1, 0); // Difference array of size n + 1

    // Apply operations to the difference array
    for (const auto& op : operations) {
        int L = op[0];
        int R = op[1];
        int X = op[2];

        diff[L] += X; // Start adding X from index L
        if (R + 1 < n) {
            diff[R + 1] -= X; // Stop adding X after index R
        }
    }

    // Calculate the final values
    int currentAdd = 0; // To keep track of cumulative sum from the difference array
    for (int i = 0; i < n; i++) {
        currentAdd += diff[i]; // Update the cumulative sum
        nums[i] += currentAdd; // Apply the cumulative sum to nums
    }

    return nums;
}
```

	Test	Expected	Got	

✓	`vector<int> nums {13, 0, 6, 9, 14, 16};`
	`vector<vector<int>> operations {{5, 5, 16}, {3, 4, 0}, {0, 2, 8}};`
	`printVector(updateArrayPerRange(nums, operations));`
		`[21, 8, 14, 9, 14, 32]`	`[21, 8, 14, 9, 14, 32]`	✓

	Test	Expected	Got	

✓	`vector<int> nums {19, 4, 3, 2, 16, 3, 17, 8, 18, 12};`
	`vector<vector<int>> operations {{0, 3, 4}, {2, 5, 12}, {3, 6, 6}, {5, 8, 5}, {8, 9, 8}, {0, 5, 9}, {1, 7, 8}, {1, 1, 3}, {5, 5, 18}};`
	`printVector(updateArrayPerRange(nums, operations));`
		[32, 28, 36, 41, 51, 61, 36, 21, 31, 20]	[32, 28, 36, 41, 51, 61, 36, 21, 31, 20]	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.