**Fundaments of Algorithms – Practice 2: Formal Specification**

**Names and surnames of the two members of the team**:

-Jose Ignacio Daguerre Garrido

-Ramón Costales de Ledesma

**Number of the team (the same used in the judge)**:

1. Provide the formal specification of a function that calculates whether a vector of numbers is palindrome. A vector is palindrome if the first element is the same as the last, the second is the same as the second to the last, and so on until reaching to the middle of the vector.

Examples:

|  |  |
| --- | --- |
| Vector X | Is palindrome X? |
| 1 7 9 15 21 15 9 7 1 | true |
| 2 8 6 2 | false |
| 9 | true |

Precondition:

|  |
| --- |
| P≡{0≤numElements<10000} |

Name of the Function:

|  |
| --- |
| fun isPalindrome(int sequence[], int numElements) return bool palindrome |

Postcondition:

|  |
| --- |
| Q≡{ palindrome=∀i: 0≤i<numElements/2: sequence[i] = sequence[numElements -1 -i] } |

1. Write a function in C++ that implements the previous specification of palindrome efficiently (i.e. with the minimum time complexity order). Please, test the function in Visual Studio or similar environment with several cases to make sure it compiles and has the expected behaviour. Use the template from the virtual campus for testing it on the judge. Then, upload the solution in the judge (http://fal.fdi.ucm.es) for the problem “Palindrome Sequences”, and make it sure it works. Then copy below only the function “isPalindrome”. Determine whether it passed the judge.

|  |
| --- |
| bool isPalindrome(int sequence[], int numElements) {  bool palindrome = true;  for (int i = 0; i < numElements / 2 && palindrome; ++i) {  if (sequence[i] != sequence[numElements - 1 - i])  palindrome = false;  }  return palindrome;  }  It passes.☺ |

1. Determine the execution time function t(n) considering n as the size of the vector, in the worst case, considering different constant times (e.g. tassign, tcompare, tincrement, tretrieve) like in the previous practice.

|  |
| --- |
| Palindrome:  tassign ·(1 + 1) + tcompare·(2(n/2 + 1) + (n/2)) + tincrement ·(n/2) + tretrieve ·(2(n/2)) |

1. Determine the complexity order of time execution of your function about palindrome sequences, and prove it with simplifications from the function t(n):

|  |
| --- |
| function t(n): 3n/2· tcompare+ n/2 · tincrement + n · tretrieve ∈ O(n) -> linear complexity |

1. Provide the formal specification of a function that returns the lowest positive (remember zero is not considered to be zero) value from a vector of integer values. If all the elements are negative or zero, the function returns -1.

Examples:

|  |  |
| --- | --- |
| Vector X | Lowest Positive function |
| 4 7 -9 15 -21 15 -9 8 3 | 3 |
| 2 8 9 6 | 2 |
| -7 -5 -3 -4 0 | -1 |

Precondition:

|  |
| --- |
| P≡{0≤numElements<10000} |

Name of the Function:

|  |
| --- |
| fun lowestPostive(int sequence[], int numElements) return int lowest |

Postcondition:

|  |
| --- |
| Q≡{ lowest= ∃i: ∀j : 0≤i≤j<numElements: sequence[i] > 0 AND sequence[i] <= sequence[j]} |

1. Write a function in C++ that implements the previous specification of the function lowest positive efficiently (i.e. with the minimum possible complexity order of execution time). Please, test the function in Visual Studio or similar environment with several cases to make sure it compiles and has the expected behaviour. Use the template from the virtual campus for testing it on the judge. Upload the solution in the judge (http://fal.fdi.ucm.es) for the problem “Lowest Positive”, and make it sure it works. Then copy the function “lowestPositive” below only if it works. Determine whether the function worked on the judge.

|  |
| --- |
| int lowestPostive(int sequence[], int numElements) {  int lowest = -1, var;  for (int i = 0; i < numElements; ++i) {  var = sequence[i];  if (var > 0) {  if (lowest == -1)  lowest = var;  else if (var < lowest)  lowest = var;  }  }  return lowest;  }  It passes.☺ |

1. Determine the execution time function t(n) considering n as the size of the vector, in the worst case, considering different constant times (e.g. tassign, tcompare, tincrement, tretrieve) like in the previous practice.

|  |
| --- |
| All positive and decrecent:  tassign·(2n + 2) + tincrement ·(n) + tcompare ·(n + 1 + 2 +3(n - 1)) + tretrieve ·(n) |

1. Determine the complexity order of time execution of your function about the lowest positive, and prove it with simplifications from the function t(n):

|  |
| --- |
| function t(n): 2n · tassign + 4n· tcompare+ n · tincrement + n · tretrieve ∈ O(n) -> linear complexity |