GUYATU ROBA JIRMO

SCT221-0851/2022

ASSIGNMENT

DESIGN AND ANALYSIS OF ALGORITHM

Q1) Write an efficient recursive algorithm that takes a sentence, starting index and ending index. The algorithm should then return a sentence that contain words between the starting and ending indices. Write recurrence relation of your algorithm and find time complexity using tracing tree method

```
#include <iostream>

#include <string>

using namespace std;


string getWordsInRange(const string& sentence, int start, int end) {

   if (start > end || start < 0 || end >= sentence.length()) {

      return "";

   }


   while (start < sentence.length() && sentence[start] == ' ') {

      start++;  // Skip leading spaces

   }


   if (start > end) {

      return "";

   }


   int wordEnd = start;

   while (wordEnd <= end && sentence[wordEnd] != ' ') {

      wordEnd++;  // Find the end of the current word

   }
```

```
    return sentence.substr(start, wordEnd - start) + " " + getWordsInRange(sentence, wordEnd + 1, end);
}

int main() {
    string sentence = "This is a sample sentence for testing purposes.";
    int start = 5;  // Starting index
    int end = 18;   // Ending index

    string result = getWordsInRange(sentence, start, end);
    cout << "Words in range: " << result << endl;

    return 0;
}
```

RECURRENCE RELATION

$T(n) = T(n/2) + O(n)$

The time complexity of the algorithm can be calculated using the tracing tree method:

```
            T(n)
           /    \
       T(n/2)    O(n)
      /    \
  T(n/4)  O(n/2)
  /   \
O(n/8)  O(n/4)
```

The tree has log(n) levels, and at each level, the work done is O(n). So, the overall time complexity is O(n log n).

Q2) Write an efficient algorithm that takes an array A[n1...nn) of sorted integers and return an array with elements that have been circularly shifted k positions to the right. For example a sorted array A=[5, 15, 29, 35, 42] is converted to A[35, 42, 5, 15, 27, 29] after circularly shifted 2 positions, while the same array A =[5, 15, 29, 35, 42] is converted to A[27, 29, 35, 42, 5, 15] after circularly shifted 4 positions. Write the recurrence relation of your solution and find the time complexity of your algorithm using iterative method

```
#include <iostream>

#include <vector>

using namespace std;


vector<int> circularShift(vector<int>& nums, int k) {

   vector<int> result(nums.size());


   int n = nums.size();

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

      int newPos = (i + k) % n;

      result[newPos] = nums[i];

   }


   return result;

}


int main() {

   vector<int> nums = {5, 15, 29, 35, 42};

   int k = 2;


   vector<int> shifted = circularShift(nums, k);


   cout << "Original Array: ";

   for (int num : nums) {
```

```
        cout << num << " ";
    }
    cout << endl;

    cout << "Shifted Array: ";
    for (int num : shifted) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}
```

RECURRENCE RELATIONAL

T(n)=O(n)

The time complexity of the algorithm is O(n) because it iterates through each element of the array once.

The iterative method for finding the time complexity involves counting the number of basic operations performed in the algorithm, which is proportional to the size of the input array. Therefore, the time complexity is linear, O(n).