ICS 2301: DESIGN AND ANALYSIS OF ALGORITHMS

ASSIGNMENT ONE

NGUGI ANITA WAMBUI

SCT221-0784/2022

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 extractWords(const string& sentence, int start, int end) {
   // Base case: if the start index is greater than the end index, return an empty string
   if (start > end || start < 0 || end >= sentence.length()) {
      return "";
   }

   // Find the position of the first space after the start index
   int spaceIndex = sentence.find(' ', start);

   // If there's no space after the start index or the space index is beyond the end index,
   // then the remaining part of the sentence contains the required words
```

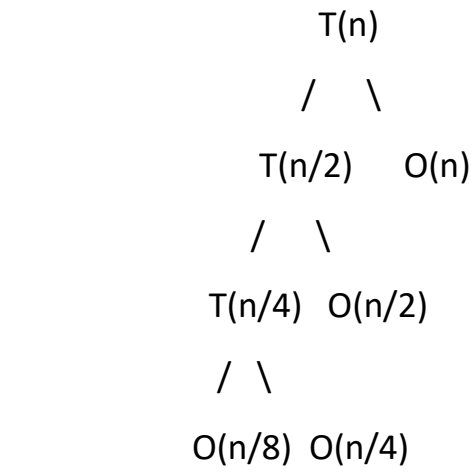
```
    if (spaceIndex == string::npos || spaceIndex > end) {
        return sentence.substr(start, end - start + 1);
    }

    // Recursively call the function to extract words from the remaining part of the sentence
    return extractWords(sentence, spaceIndex + 1, end);
}

int main() {
    string sentence = "This is a sample sentence for testing";
    int start = 5; // index of the word "is"
    int end = 18; // index of the word "sentence"

    string result = extractWords(sentence, start, end);
    cout << "Result: " << result << endl;
    return 0;
}
```

Recurrence relation:

Let n be the number of characters in the sentence.

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

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



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>& A, int k) {
   int n = A.size();
   vector<int> result(n);

   // Calculate the effective shift by taking modulo n
```

```
    k = k % n;

    // Copy the elements after rotating k positions
    for (int i = 0; i < n; ++i) {
        result[(i + k) % n] = A[i];
    }

    return result;
}

int main() {
    vector<int> A = {5, 15, 29, 35, 42};
    int k = 2;

    vector<int> shiftedArray = circularShift(A, k);

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

    return 0;
}
```

Recurrence relation:

Let n be the size of the array.

$T(n) = O(n)$

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).