Recursion 2

In this lecture, we are going to understand how to solve different kinds of problems using recursion. It is strictly advised to complete the assignments for the earlier topics, if due.

Recursion and Strings

Here, we are going to discuss the different problems that can be solved using recursion on strings:

- Finding the length of the string

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

int length(char s[]) {
        if (s[0] == '\0') {                                    // Base case
                return 0;
        }
        int smallStringLength = length(s + 1);                 // Recursive call
        return 1 + smallStringLength;                          // Small calculation
}


int main() {
        char str[100];
        cin >> str;

        int l = length(str);
        cout << l << endl;
}
```

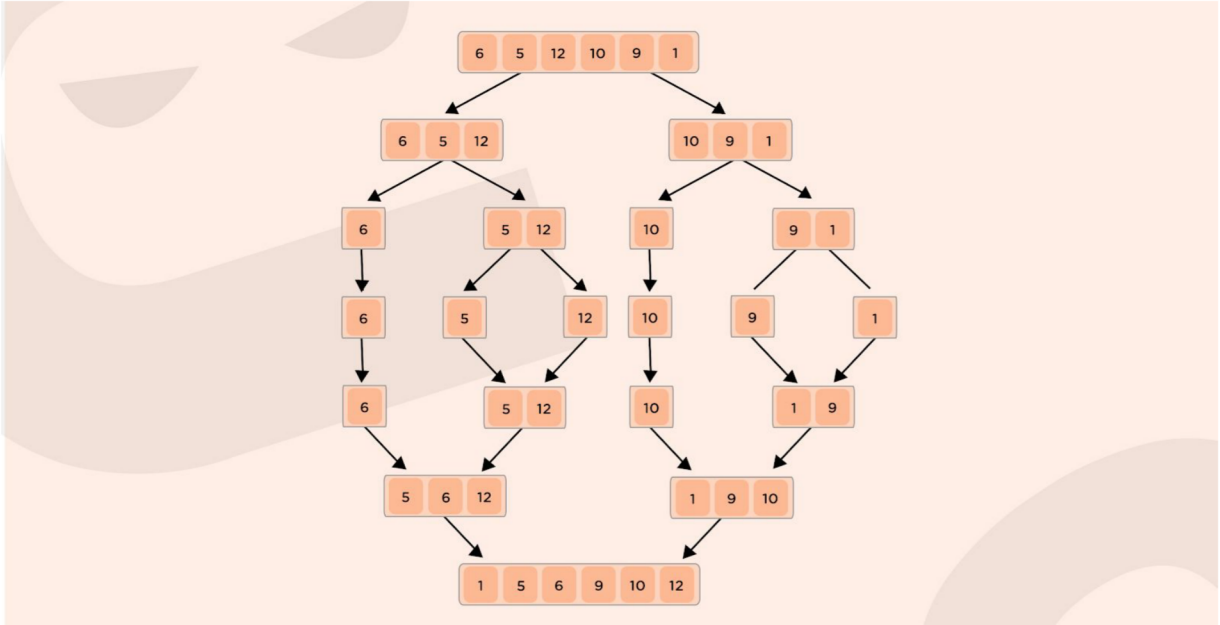
- Remove X from a given string

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

void removeX(char s[]) {
        if (s[0] == '\0') {                          // Base case
                return;
        }

        if (s[0] != 'x') {                           // Small calculation
                removeX(s + 1);                      // Recursive call
        } else {
                int i = 1;
                for (; s[i] != '\0'; i++) {
                        s[i - 1] = s[i];
                }
                s[i - 1] = s[i];
                removeX(s);
        }
}

int main() {
        char str[100];
        cin >> str;
        removeX(str);
        cout << str << endl;
}
```

Sorting Technique

Merge Sort

Consider the example below that shows the working of merge sort over the given array and also how it **divides and conquers** the array.



Let's now look at the algorithm associated with it...

Algorithm for Merge Sort:

Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines the smaller sorted lists, keeping the new list sorted too.

Step 1 − If it is only one element in the list it is already sorted, return.

Step 2 − Divide the list recursively into two halves until it can no more be divided.

Step 3 − Merge the smaller lists into new list in sorted order.

Now, you can code it easily by following the above three steps.

It has just one disadvantage and it is that it's **not an in-place sorting** technique, which means it creates a copy of the array and then works on that copy.

Time Complexity : O(nlogn)

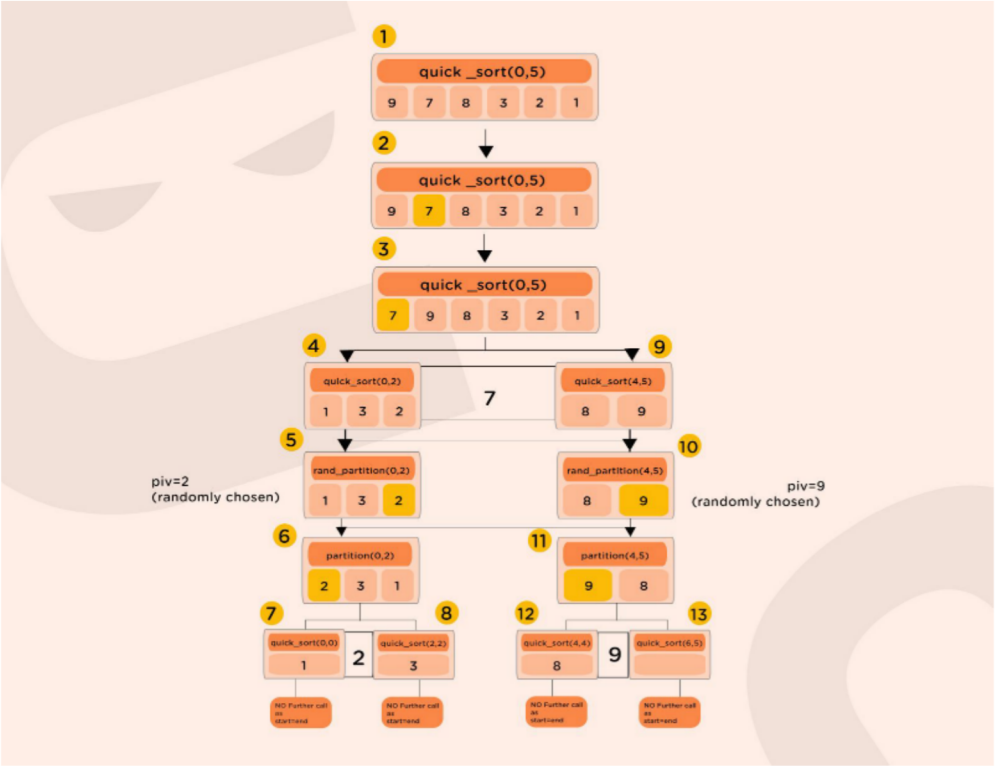
Extra Space Complexity : O(n)

Quick sort

Quick sort is based on the **divide-and-conquer approach** based on the idea of choosing one element as a pivot element and partitioning the array around it, such that: Left side of pivot contains all the elements that are less than the pivot element and Right side contains all elements greater than the pivot.

It reduces the space complexity and removes the use of the auxiliary array that is used in merge sort. Selecting a random pivot in an array results in an improved time complexity in most of the cases.

To get a better understanding of the approach consider the following example where a pictorial representation of the quick sort on an array is shown.



Algorithm for Quick Sort:

On the basis of Divide and Conquer approach, quicksort algorithm can be explained as:

- **Divide**

  The array is divided into subparts, taking pivot as the partitioning point. The elements smaller than the pivot are placed to the left of the pivot and the elements greater than the pivot are placed to the right side.

- **Conquer**

  The left and right sub-parts are again partitioned by selecting pivot elements for them. This can be achieved by recursively passing the subparts into the algorithm.

- **Combine**

This part does not play a significant role in quicksort. The array is already sorted at the end of the conquer step.

Advantage of quicksort over merge sort is that it does not require any extra space to sort the given array, therefore, it is an in-place sorting technique.

Time Complexity : O(nlogn)

Extra Space Complexity : O(1)

Strings

Let us think of string as a class, instead of a data type. We all already know that strings are the character arrays that ends with a NULL character itself.

Syntax for declaring a string:

```
string str;
```

Syntax for declaring a string dynamically:

```
string *str = new string;
```

To take input of the string, we can use the **getline()** function. Newline is the delimiter for the getline() function. Follow the syntax below:

```
getline(cin, str);          // where str is the name of the string
```

You can treat the string like any other character array as well. You can go to any specific index (indexing starts from 0), assign any string value by using equals to (=) operator, etc.

To concatenate two strings you can use '+' operator. For example, to concatenate two strings s1 and s2 and put it in another string str, syntax is as below:

```
string str = s1 + s2;
```

To calculate the length of the string you can use the in-built function (.length()) for that:

```
int len = str.length();
```

You can do the same using the following syntax also:

```
int len = str.size();
```

To extract a particular segment of the string, we can use .substr() function.

```
string s = str.substr(beginning_index, length_ahead_of_starting_index);
```

Here, if you omit the second argument, then automatically the function takes the complete string, after the specified starting index.

You can also find any particular substring in the given string using .find() function. It will return the starting index of the substring to be found in the given string.

```
int index = str.find(name_of_the_pattern_to_be_checked);
```

Note: .find() function finds the first occurring index of the given pattern.

These concepts would be much clearer after studying OOPs.

Let's practice some of the questions over the topics covered in this lecture...

Questions:

Visit the following links for practicing some questions...

- https://www.hackerearth.com/practice/algorithms/sorting/merge-sort/practice-problems/
- https://www.hackerearth.com/practice/algorithms/sorting/quick-sort/practice-problems/
- https://www.techiedelight.com/recursion-practice-problems-with-solutions (On this webpage, visit the subheading **String**)