

INTRODUCTION TO ADTs

SPECIFICATION

```
spec StringHandler
  genre StringHandler
  operations
    empty: string -> bool
    first: string -> char
    iesm: string,int -> char
    middle: string -> char
    addAtBeg: string,string -> string
    addAtEnd: string,string -> string
    concatenate: string,string -> string
    reverse: string -> string
    rotate: string,int -> string
endspec
```

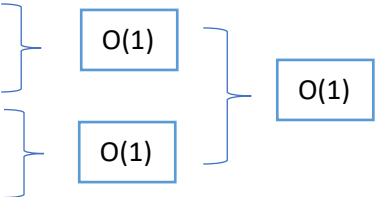
IMPLEMENTATION

```
class StringHandler
  public bool empty(string)
  public char first(string)
  public char iesm(string,int)
  public char middle(string)
  public string addAtBeg(string,string)
  public string addAtEnd(string,string)
  public string concatenate(string,string)
  public string reverse(string)
  public string rotate(string,int)
endclass
```

```

public bool StringHandler::empty(s:string)
    if (s.length()==0)
        return true
    else
        return false
endmethod

```



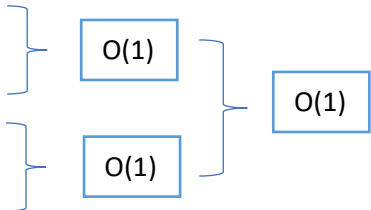
The diagram shows the complexity analysis for the `empty` method. It consists of two branches: an `if` branch and an `else` branch. The `if` branch contains the statement `s.length()==0` and the `return true` statement. The `else` branch contains the `return false` statement. Brackets on the right side of the code group the `if` branch and the `else` branch together, with a final bracket indicating the overall complexity is $O(1)$.

- The running time of “empty” method is $O(1)$ because it only has operations with simple statements.

```

public char StringHandler::first(s:string)
    if (empty(s))
        return ("Error")
    else
        return s[0]
endmethod

```



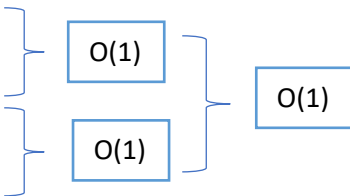
The diagram shows the complexity analysis for the `first` method. It has two branches: an `if` branch and an `else` branch. The `if` branch contains the call `empty(s)` and the `return ("Error")` statement. The `else` branch contains the `return s[0]` statement. Brackets on the right side of the code group the `if` branch and the `else` branch together, with a final bracket indicating the overall complexity is $O(1)$.

- The running time of “first” method is $O(1)$ because it only has operations with simple statements.

```

public char StringHandler::iesm(s:string,i:int)
    if (empty(s))
        return ("Error")
    else
        return s[i]
endmethod

```



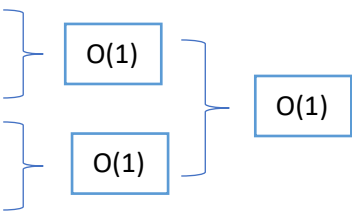
The diagram shows the complexity analysis for the `iesm` method. It has two branches: an `if` branch and an `else` branch. The `if` branch contains the call `empty(s)` and the `return ("Error")` statement. The `else` branch contains the `return s[i]` statement. Brackets on the right side of the code group the `if` branch and the `else` branch together, with a final bracket indicating the overall complexity is $O(1)$.

- The running time of “iesm” method is $O(1)$ because it only has operations with simple statements.

```

public char StringHandler::middle(s:string)
    if (empty(s))
        return ("Error")
    else
        return s[s.length\2]
endmethod

```



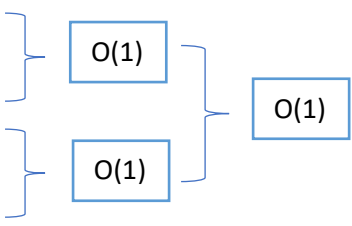
The diagram shows the complexity analysis for the `middle` method. It has two branches: an `if` branch and an `else` branch. The `if` branch contains the call `empty(s)` and the `return ("Error")` statement. The `else` branch contains the `return s[s.length\2]` statement. Brackets on the right side of the code group the `if` branch and the `else` branch together, with a final bracket indicating the overall complexity is $O(1)$.

- The running time of “middle” method is $O(1)$ because it only has operations with simple statements.

```

public string StringHandler::addAtBeg(s:string,s1:string)
    if (empty(s))
        newString=s1
    else
        newString=s1+s
    endmethod

```



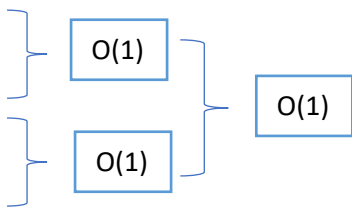
The diagram shows the complexity of the `addAtBeg` method. It consists of two branches: an `if` branch and an `else` branch. The `if` branch contains the statement `newString=s1`, and the `else` branch contains the statement `newString=s1+s`. Both branches are enclosed in curly braces, each labeled with $O(1)$. A larger curly brace on the right groups both branches, also labeled with $O(1)$, indicating that the overall time complexity of the method is constant.

- The running time of “addAtBeg” method is $O(1)$ because it only has operations with simple statements.

```

public string StringHandler::addAtEnd(s:string,s1:string)
    if (empty(s))
        newString=s1
    else
        newString=s+s1
    endmethod

```



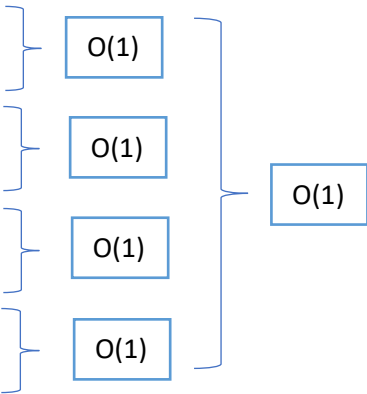
The diagram shows the complexity of the `addAtEnd` method. It consists of two branches: an `if` branch and an `else` branch. The `if` branch contains the statement `newString=s1`, and the `else` branch contains the statement `newString=s+s1`. Both branches are enclosed in curly braces, each labeled with $O(1)$. A larger curly brace on the right groups both branches, also labeled with $O(1)$, indicating that the overall time complexity of the method is constant.

- The running time of “addAtEnd” method is $O(1)$ because it only has operations with simple statements.

```

public string StringHandler::concatenate(s:string,s1:string)
    if (empty(s))
        newString=s1
    else if (empty(s1))
        newString=s
    else if (empty(s)&&empty(s1))
        printf("%s","Both strings are empty")
    else
        newString=s+s1
    endmethod

```



The diagram shows the complexity of the `concatenate` method. It consists of four branches: an `if` branch, two `else if` branches, and an `else` branch. The `if` branch contains `newString=s1`, the first `else if` branch contains `newString=s`, the second `else if` branch contains `printf("%s","Both strings are empty")`, and the `else` branch contains `newString=s+s1`. Each of these four branches is enclosed in a curly brace labeled with $O(1)$. A large curly brace on the right groups all four branches, also labeled with $O(1)$, indicating that the overall time complexity of the method is constant.

- The running time of “concatenate” method is $O(1)$ because it only has operations with simple statements.

```
public string StringHandler::reverse(s:string)
```

```
    if (s.length() == 1)
```

```
        return s;
```

```
    else
```

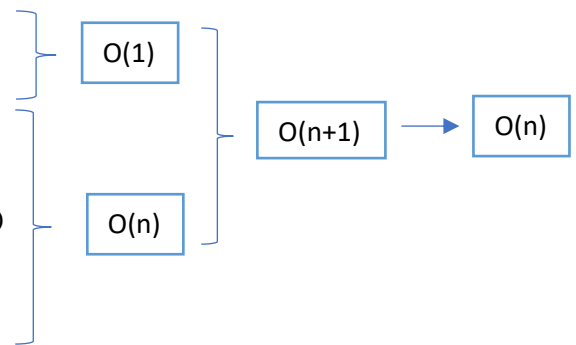
```
        s_reverse=""
```

```
        for (int i=s.length()-1;i>=0;i--)
```

```
            s_reverse+=s[i]
```

```
        return s_reverse
```

```
endmethod
```



- The running time of “reverse” method is $O(n)$ because it has a conditional statement in which the running time of the largest branch is $O(n)$.
- This largest branch is a `for` loop that is being executing n times.

```
public string StringHandler::rotate(s:string,n:int,s1:string)
```

```
    suffix=""
```

```
    prefix=""
```

```
    if (s.length() == 1)
```

```
        newString=s
```

```
    else
```

```
        if (s1.compare("left")==0)
```

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

```
                suffix+=s[i]
```

```
            for (int j=n;j<s.length();j++)
```

```
                prefix+=s[j]
```

```
        else
```

```
            for (int i=0;i<(s.length()-n);i++)
```

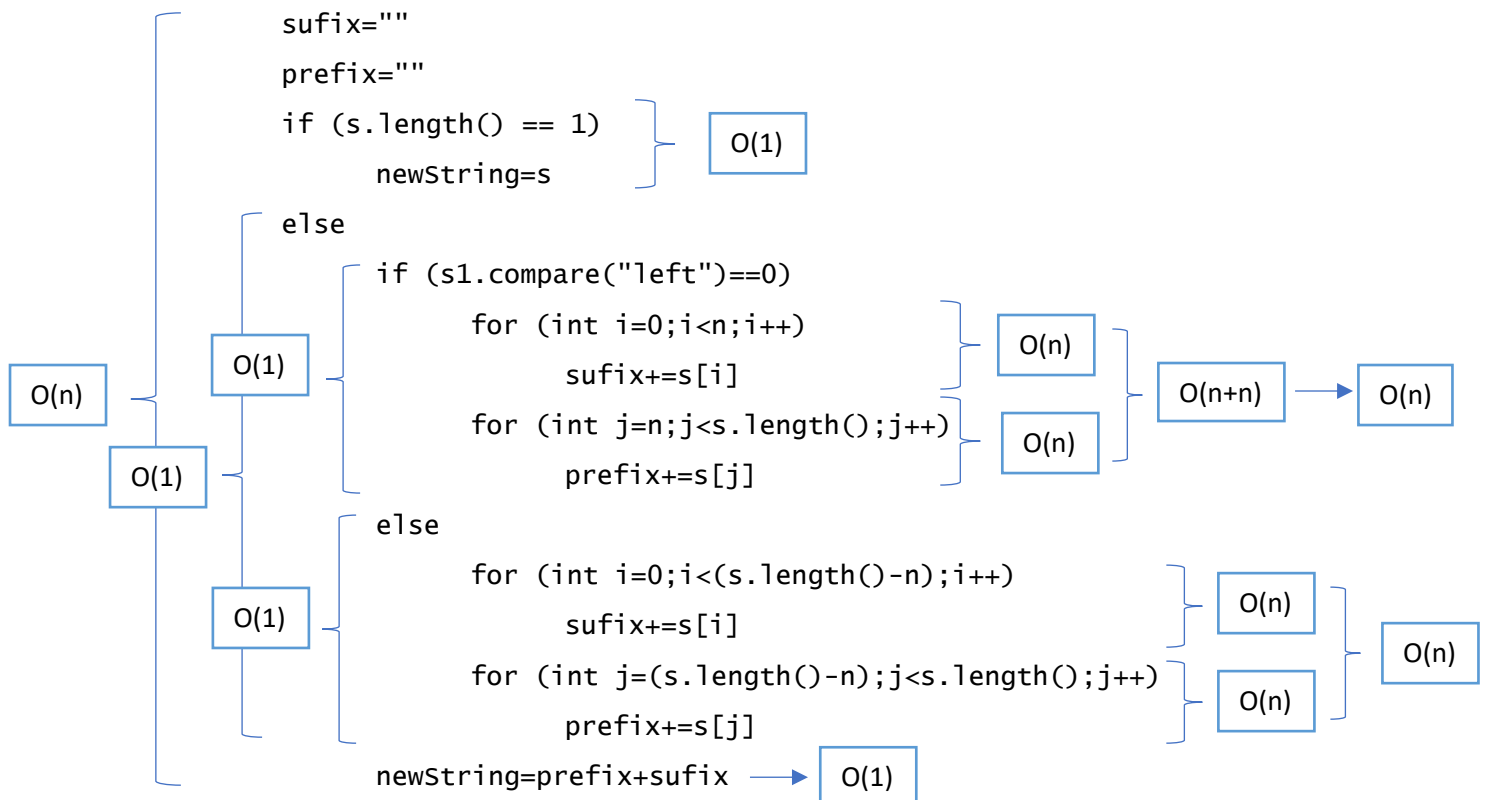
```
                suffix+=s[i]
```

```
            for (int j=(s.length()-n);j<s.length();j++)
```

```
                prefix+=s[j]
```

```
        newString=prefix+suffix
```

```
endmethod
```



- The running time of “rotate” method is $O(n)$ because it has an `if-else` statement in which the running time of the largest branch is $O(n)$.
- This largest branch are some `for` loops that are being executing n times at the same time.

- Is it possible to get a better running time with alternative data structures and/or implementation/s?

No, because almost all the method has $O(1)$ that can't have a better running time. Then, the two ones that has a running time of $O(n)$ I think it can't be converted to a better running time, due to the for loop that is obligatory for the correct behaviour function of the method.