CSE225

*Assignment 01*

# DATA STRUCTURE AND ALGORITHM

Summer 2023

**Submitted By:**
Md. Misbah Khan
ID: 2132089642
**Section:** 04

**Submitted To:**
Rifat Ahmed Hassan

**Submitted On:**
23 September, 2023

## Answer to Question 01(I):

In our given code, there are two loops. Except these two loops, runtime for other part of this code will be O(1). For these two loops:

```
while (location < length)
{
        if(item > info[location])
            location++;
        else
            break;
}
```

Time complexity for this part will be O(N). Because for worst case, we need to go index to length-1, one index increasing per iteration.

For the second loop:

```
for (int index = length-1; index >= location; index--)
    info[index+1] = info[index];
info[location] = item;
length++;
```

Time complexity for this part will be also O(N). Because for worst case, we need to go length-1 to index 0, one index increasing per iteration.

## Answer to Question 01(II):

Dynamic memory allocation is allocating memory during runtime. On the other hand, Static memory allocation is allocating memory during compile time. Dynamic memory allocation allocate memory only when needed, so no memory gets wasted. But when we allocate memory statically, there might be some empty spaces that come to no use, so these memory spaces get wasted.

## Answer to Question 02:

After ordering the given functions by growth rate (smallest to largest), we get:

$2/N < 37 < \sqrt{N} < N < N \log \log N < N \log N <= N \log (N^2) < N \log^2 N < N^{1.5} < N^2 < N^2 \log N < N^3 < 2^{N/2} < 2^N$

## Answer to Question 03:

Time complexity of given code fragments are:

| Time Complexity | Explanation |
|---|---|
| O(N) | One for loop. In worst case, it will run n times. |
| $O(N^2)$ | Two for loops. In worst case, outer for loop will run n times, and for each iteration in outer loop, inner loop will run n times. So, time complexity: $n * n = n^2$ |
| $O(N^3)$ | Two for loops. In worst case, outer for loop will run n times, and for each iteration in outer loop, inner loop will run n*n times. So, time complexity: $n * n * n = n^3$ |
| $O(N^2)$ | Two for loops. In worst case, outer for loop will run n times, and for each iteration in outer loop, inner loop will run i times. So, time complexity: $n * i = n * n = n^2$ |
| $O(N^5)$ | Three for loop. In worst case, 1st for loop will run n times, 2nd for loop will run $j = i^2 = n^2$ times and third loop will run $k = j = i^2 = n^2$ times. So, time complexity: $n * n^2 * n^2 = n^5$ |

## Answer to Question 04:

a) This function takes a parameter key of type ItemType and return True if the key is found on this sorted list and return False if not found.
b) bool IsThere(ItemType item)
c) The definition of isThere function using binary search algorithm will be:

```
template <class ItemType>
bool SortedType<ItemType>::isThere(ItemType item)
{
    int mid, first = 0, last = length - 1;
    while (first <= last)
    {
        mid = (first + last) / 2;
        if (info[mid] == item)
            return true;
    }

    if(info[mid]<key)
       First = mid + 1;
    else
        last = mid - 1;

    return false;
}
```

d) There is one while loop, which decrease it’s loop time by half after each iteration. So time complexity will be O(log n)

**Answer to Question 05:** A point to be noted that we used StackType.cpp and StackType.h from our slide and keep them unchanged, the main.cpp file is modified by me.

::::::::::::::StackType.h::::::::::::::

```
#ifndef STACKTYPE_H
#define STACKTYPE_H
const int MAX_ITEMS = 5;

class FullStack
{};
class EmptyStack
{};

template <class ItemType>
class StackType
{
    public:
        StackType();
        bool IsFull();
        bool IsEmpty();
        void MakeEmpty();
        void Push(ItemType);
        ItemType Pop();
        ItemType Peek();
        void ShowItems();
    private:
        int top;
        ItemType  items[MAX_ITEMS];
};

#endif // STACKTYPE_H
```

::::::::::::::StackType.cpp::::::::::::::

```
#include <iostream>
#include "StackType.h"

using namespace std;

template <class ItemType>
StackType<ItemType>::StackType()
{
    top = -1;
}
template <class ItemType>
bool StackType<ItemType>::IsEmpty()
{
    return (top == -1);
}
template <class ItemType>
void StackType<ItemType>::MakeEmpty()
{
    top = -1;
}
template <class ItemType>
bool StackType<ItemType>::IsFull()
{
    return (top ==  MAX_ITEMS-1);
}
template <class ItemType>
void StackType<ItemType>::Push(ItemType newItem)
{
    if( IsFull() )
        throw FullStack();
    top++;
    items[top] = newItem;
}
template <class ItemType>
ItemType StackType<ItemType>::Pop()
{
    if( IsEmpty() )
        throw EmptyStack();
    top--;
    return items[top+1];
}

template <class ItemType>
ItemType StackType<ItemType>::Peek()
{
    if (IsEmpty())
        throw EmptyStack();
    return items[top];
}

template <class ItemType>
void StackType<ItemType>::ShowItems()
{
    for (int i=0; i<=top; i++) {
        cout << items[i] << "    ";
    }
    cout << endl;
    return;
}
```

:::::::::::::: main.cpp::::::::::::::

```
#include <iostream>
#include <string>
#include "StackType.h"
using namespace std;

bool isPalindrome(string str)
{
    stack<char> charStack;
    stack<char> charStack2;
    int length = str.length();

    for (int i = 0; i < length / 2; ++i)
    {
        charStack.Push(str[i]);
    }

    for (int i = length; i > (length + 1) / 2; --i)
    {
        charStack2.Push(str[i]);
    }

    while (!charStack.empty() && !charStack2.empty())
    {
        if (charStack.top() != charStack2.top())
        {
            return false;
        }
        else
        {
            return true;
        }
        charStack.Pop();
        charStack2.Pop();
    }
}
int main()
{
    string input;
    cout << "Enter a string: ";
    getline(cin, input, '.');

    string str;
    for (int i = 0; i < input.length(); ++i)
    {
        str += tolower(input[i]);
    }

    if (isPalindrome(str))
    {
        cout << "The string is a palindrome." << endl;
    }
    else
    {
        cout << "The string is not a palindrome." << endl;
    }

    return 0;
}
```

**<u>Answer to Question 06:</u>** Here is the body of the ReplaceItem function, which will be added on StackType.cpp. As I write them on previous question, I am just writing the function body.

```
template <class ItemType>
void StackType<ItemType>::ReplaceItem(StackType &stack, ItemType oldItem,
ItemType newItem)
{
    StackType<ItemType> tempStack;

    while (!stack.IsEmpty())
    {
        ItemType i = stack.Pop();

        if (i == oldItem)
        {
            i = newItem;
        }
        stack.Push(i);
    }
}
```

**<u>Answer to Question 07:</u>** Here is the C++ code for postfix expressions. A point to be noted that we used StackType.cpp and StackType.h from our slide and keep them unchanged, the main.cpp file is modified by me.

::::::::::::::StackType.h::::::::::::::

```
#ifndef STACKTYPE_H
#define STACKTYPE_H
const int MAX_ITEMS = 5;

class FullStack
{};
class EmptyStack
{};

template <class ItemType>
class StackType
{
    public:
        StackType();
        bool IsFull();
        bool IsEmpty();
        void MakeEmpty();
        void Push(ItemType);
        ItemType Pop();
        ItemType Peek();
        void ShowItems();
    private:
        int top;
        ItemType  items[MAX_ITEMS];
};

#endif // STACKTYPE_H
```

::::::::::::::StackType.cpp:::::::::::::

```
#include <iostream>
#include "StackType.h"

using namespace std;

template <class ItemType>
StackType<ItemType>::StackType()
{
    top = -1;
}
template <class ItemType>
bool StackType<ItemType>::IsEmpty()
{
    return (top == -1);
}
template <class ItemType>
void StackType<ItemType>::MakeEmpty()
{
    top = -1;
}
template <class ItemType>
bool StackType<ItemType>::IsFull()
{
    return (top ==  MAX_ITEMS-1);
}
template <class ItemType>
void StackType<ItemType>::Push(ItemType newItem)
{
    if( IsFull() )
        throw FullStack();
    top++;
    items[top] = newItem;
}
template <class ItemType>
ItemType StackType<ItemType>::Pop()
{
    if( IsEmpty() )
        throw EmptyStack();
    top--;
    return items[top+1];
}

template <class ItemType>
ItemType StackType<ItemType>::Peek()
{
    if (IsEmpty())
        throw EmptyStack();
    return items[top];
}

template <class ItemType>
void StackType<ItemType>::ShowItems()
{
    for (int i=0; i<=top; i++) {
        cout << items[i] << "    ";
    }
    cout << endl;
    return;
}
```

::::::::::::::main.cpp::::::::::::::

```cpp
#include <iostream>
#include <string>
#include "StackType.cpp"

using namespace std;

bool isOperator(char c)
{
    return (c == '+' || c == '-' || c == '*' || c == '/');
}
bool isOperand(char c)
{
    return (c >= 48 && c <= 57);
}
int performOperation(int num1, int num2, char op)
{
    int ans;
    switch (op)
    {
    case '+':
        ans = num1 + num2;
    case '-':
        ans = num1 - num2;
    case '*':
        ans = num1 * num2;
    case '/':
        ans = num1 / num2;
    }
    return ans;
}
int postfix(string line)
{
    StackType<int> objStack;
    for (int i = 0; i < line.length(); i++)
    {
        if (isdigit(line[i]))
        {
            objStack.Push(int(line[i]));
        }
        else if (isOperator(line[i]))
        {
            int num2 = objStack.Pop();
            int num1 = objStack.Pop();
            char opera = line[i];
            int result = performOperation(num1, num2, opera);
            objStack.Push(result);
        }
    }
    return objStack.Peek();
}

int main()
{
    string input;
    getline(cin, input);
    int answer = postfix(input);
    cout << "Result: " << answer << endl;
    return 0;

}
```

**<u>Answer to Question 08:</u>**

a)

::::::::::::: QueueType.h:::::::::::::

```
#ifndef QUETYPE_H
#define QUETYPE_H

#define DEFAULT_SIZE 50

typedef char ItemType;
class FullQueue
{};
class EmptyQueue
{};

template <class ItemType>
class QueType {
public:
    QueType();
    QueType(int max);
    ~QueType();
    void MakeEmpty();
    bool IsEmpty();
    bool IsFull();
    int Length();
    void Enqueue(ItemType newItem);
    void Dequeue(ItemType& item);
private:
    int front;
    int rear;
    ItemType* items;
    int maxQue;
    int length;


};


#endif // QUETYPE_H
```

b)

::::::::::::: QueueType.cpp:::::::::::::

```
#include<iostream>
#include "QueType.h"

template <class ItemType>
QueType<ItemType>::QueType(int max)
{
  maxQue = max + 1;
  front = maxQue - 1;
  rear = maxQue - 1;
  items = new ItemType[maxQue];
  length = 0;
}
```

```
template <class ItemType>
QueType<ItemType>::QueType()
{
  maxQue = DEFAULT_SIZE + 1;
  front = maxQue - 1;
  rear = maxQue - 1;
  items = new ItemType[maxQue];
  length = 0;
}
template <class ItemType>
QueType<ItemType>::~QueType()
{
  delete[] items;
}
template <class ItemType>
void QueType<ItemType>::MakeEmpty()
{
  front = maxQue - 1;
  rear = maxQue - 1;
  length = 0;
}
template <class ItemType>
bool QueType<ItemType>::IsEmpty()
{
  return (length == 0);
}

template <class ItemType>
bool QueType<ItemType>::IsFull()
{
  return (length == maxQue - 1);
}

template <class ItemType>
int QueType<ItemType>::Length()
{
  return length;
}
template <class ItemType>
void QueType<ItemType>::Enqueue(ItemType newItem)
{
  if (IsFull())
    throw FullQueue();
  else
  {
    rear = (rear + 1) % maxQue;
    items[rear] = newItem;
    length++;
  }
}
template <class ItemType>
void QueType<ItemType>::Dequeue(ItemType &item)
{
  if (IsEmpty())
    throw EmptyQueue();
  else
  {
    front = (front + 1) % maxQue;
    item = items[front];
    length--;
  }
}
```

To implement the code according to new requirement, we need to add a new function and change some existing function. We added the function Length(), and modified functions MakeEmpty(), IsEmpty(), IsFull(), Enqueue() and Dequeue() in order to maintain the new length.

c) If we compare our new code with the previous one from slide, we do not see any change in time complexity of these two codes. Because in our new code, we add or modify some function, but they do not contain any loop. As they both work in constant time, both the new and old implementation’s time complexity is O(1).