Fundação Getulio Vargas
Estruturas de Dados e Algoritmos
Prof. Jorge Poco

Homework 2
Due: March 27, 2020

Homework 2

1 Growth of Functions [2pts]

A For each of the following pairs of functions, either $f(n)$ is in $O(g(n))$, $f(n)$ is in $\Omega(g(n))$, or $f(n) = \Theta(g(n))$. Determine which relationship is correct and briefly explain why.

- **(0.25pt)** $f(n) = \log n^2$; $g(n) = \log n + 5$
  The correct relation is $f(n) = \Theta(g(n))$. Since $f(n) = \log n^2 = 2\log n$ has the same order of increasing than $g(n)$, that is, $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ at the same time.
- **(0.25pt)** $f(n) = \log^2 n$; $g(n) = \log n$
  The correct relation is $f(n) = \Omega(g(n))$. Because $\log n > 1$ for n greater than the logarithmic base (a), we have that $\log^2 n > \log n$ $\forall n > a$.
- **(0.25pt)** $f(n) = n\log n + n$; $g(n) = \log n$
  The correct relation is $f(n) = \Omega(g(n))$. When $n > 1$, $n\log n > \log n$. So, $n\log n + n > \log n$ $\forall n > 1$.
- **(0.25pt)** $f(n) = 2^n$; $g(n) = 10n^2$
  The correct relation is $f(n) = \Omega(g(n))$. When $n = 10$, we have that $1024 = 2^{10} > 10 \cdot 10^2 = 1000$. Since the function $h(n) = 2^n - 10n^2$ is always increasing for $n \geq 10$, $\forall n \geq 10$: $2^n > 10n^2$.

B **(0.5pt)** Prove that $n^3 - 3n^2 - n + 1 = \Theta(n^3)$.

So, we need to prove that $n^3 - 3n^2 - n + 1 = O(n^3)$ (1) and $n^3 - 3n^2 - n + 1 = \Omega(n^3)$ (2).

(1): As $\lim_{n\to\infty} \dfrac{n^3 - 3n^2 - n + 1}{n^3} = 1$, we can find a n_0 such that $\forall n \geq n_0$: $\dfrac{n^3 - 3n^2 - n + 1}{n^3} < 2$. In other words, $\forall n \geq n_0$: $n^3 - 3n^2 - n + 1 < 2n^3$.

(2): As $\lim_{n\to\infty} \dfrac{n^3 - 3n^2 - n + 1}{n^3} = 1$, we can find a n_1 such that $\forall n \geq n_1$: $\dfrac{n^3 - 3n^2 - n + 1}{n^3} > 0,5$. In other words, $\forall n \geq n_1$: $n^3 - 3n^2 - n + 1 > 0,5n^3$.

C **(0.5pt)** Prove that $n^2 = O(2^n)$.

For $n = 5$, we have that $25 = n^2 < 2^n = 32$. Since the function $h(n) = 2^n - n^2$ is always increasing for $n \geq 5$, $\forall n \geq 5$: $2^n > n^2$.

2 Recurrence Equations [2pts]

Solve the following equations and give their order of complexity. In class, we saw four methods to solve this type of problem. In this case, the exercises from A to F you will have to use the characteristic equation method. For this, you have to study the attached document "recurrencias.pdf". In that document, you can find an explanation of the technique and examples of how to solve them; it is very didactic. If you have some doubts, post a question through the Classroom.

A **(0.25pt)** $T(n) = 3T(n-1) + 4T(n-2)$ if $n > 1; T(0) = 0; T(1) = 1$

Characteristic equation: $x^2 - 3x - 4 = 0$.

Roots: $x_1 = 4$ and $x_2 = -1$.

So, $T(n) = c_1 4^n + c_2(-1)^n$. Using $T(0) = 0$ and $T(1) = 1$ we have

$\begin{cases} c_1 + c_2 = 0 \\ 4c_1 - c_2 = 1 \end{cases}$

Then, $c_1 = \dfrac{1}{5}$ and $c_2 = -\dfrac{1}{5}$.

$$T(n) = \frac{1}{5}(4^n - (-1)^n)$$

Thus, $T(n) = O(4^n)$.

B **(0.25pt)** $T(n) = 2T(n-1) - (n+5)3^n$ if $n > 0; T(0) = 0$

Characteristic equation: $(x-2)(x-3)^2 = 0$.

Roots: $x_1 = 2$ and $x_2 = 3$.

So, $T(n) = c_1 2^n + c_2 3^n + c_3 n 3^n$. Using $T(0) = 0$, $T(1) = -18$ and $T(2) = -99$ we have

$\begin{cases} c_1 + c_2 = 0 \\ 2c_1 + 3c_2 + 3c_3 = -18 \\ 4c_1 + 9c_2 + 18c_3 = -99 \end{cases}$

Then, $c_1 = 9$, $c_2 = -9$ and $c_3 = -3$.

$$T(n) = 3(3 \cdot 2^n - 3 \cdot 3^n - n3^n)$$

As T assumes negative values for n greater than a N, it doesn't represent an algorithm.

C **(0.25pt)** $T(n) = 4T(n/2) + n^2$, if $n > 4; T(0) = 0;$ n power of 2 ; $T(1) = 1; T(2) = 8$

Changing variables with $n = 2^k$, $T(2^k) = 4T(2^{k-1}) + 4^k$.

Characteristic equation: $(x-4)^2 = 0$.

So, $T_k = c_1 4^k + c_2 k 4^k$, $T(n) = c_1 n^2 + c_2 n^2 \log n$. Using $T(1) = 1$ and $T(2) = 8$ we have

$\begin{cases} c_1 = 1 \\ 4c_1 + 4c_2 = 8 \end{cases}$

Then, $c_1 = 1$ and $c_2 = 1$.

$$T(n) = n^2 + n^2 \log n$$

Thus, $T(n) = O(n^2 \log n)$

D **(0.25pt)** $T(n) = 2T(n/2) + n \log n$ if $n > 1;$ n power of 2

Changing variables with $n = 2^k$, $T(2^k) = 2T(2^{k-1}) + 2^k k$.

Characteristic equation: $(x-2)^3 = 0$.

So, $T_k = c_1 2^k + c_2 k 2^k + c_3 k^2 2^k$, $T(n) = c_1 n + c_2 n \log n + c_3 n \log^2 n$. Using $T(1) = 0$, $T(2) = 2$ and $T(4) = 12$ we have

$$\begin{cases} c_1 = 0 \\ 2c_1 + 2c_2 + 2c_3 = 2 \\ 4c_1 + 8c_2 + 16c_3 = 12 \end{cases}$$

Then, $c_1 = 0$, $c_2 = \frac{1}{2}$ and $c_3 = \frac{1}{2}$.

$$T(n) = \frac{1}{2} n \log n + \frac{1}{2} n \log^2 n$$

Thus, $T(n) = O(n \log^2 n)$

E **(0.25pt)** $T(n) = T(n-1) + 2T(n-2) - 2T(n-3)$ if $n > 1; T(n) = 9n^2 - 15n + 106$ if $n = 0, 1, 2$

Characteristic equation: $x^3 - x^2 - 2x + 2 = (x-1)(x-\sqrt{2})(x+\sqrt{2}) = 0$.

Roots: $x_1 = 1$, $x_2 = \sqrt{2}$ and $x_3 = -\sqrt{2}$.

So, $T(n) = c_1 + c_2\sqrt{2}^n + c_3(-\sqrt{2})^n = c_1 + c_2 2^{\frac{n}{2}} + c_3(-1)^n 2^{\frac{n}{2}}$. Using $T(0) = 106$, $T(1) = 100$ and $T(2) = 112$ we have

$$\begin{cases} c_1 + c_2 + c_3 = 106 \\ c_1 + \sqrt{2}c_2 - \sqrt{2}c_3 = 100 \\ c_1 + 2c_2 + 2c_3 = 112 \end{cases}$$

Then, $c_1 = 100$, $c_2 = 3$ and $c_3 = 3$.

$$T(n) = 100 + 3 \cdot 2^{\frac{n}{2}}(1 + (-1)^n)$$

Thus, $T(n) = O(2^{\frac{n}{2}})$.

F **(0.25pt)** $T(n) = (3/2)T(n/2) - (1/2)T(n/4) - (1/2)$ if $n > 2; T(1) = 1; T(2) = 3/2$

Changing variables with $n = 2^k$, $T(2^k) = \frac{3}{2}T(2^{k-1}) - \frac{1}{2}T(2^{k-2}) - \frac{1}{2}$.

Characteristic equation: $(x^2 - \frac{3}{2}x + \frac{1}{2})(x-1) = (x-1)^2(x - \frac{1}{2}) = 0$.

Roots: $x_1 = 1$ and $x_2 = \frac{1}{2}$.

So, $T_k = c_1(1)^k + c_2 k(1)^k + c_3(\frac{1}{2})^k$, $T(n) = c_1 + c_2 \log n + c_3\frac{1}{n}$. Using $T(1) = 1$, $T(2) = \frac{3}{2}$ and $T(4) = \frac{5}{4}$ we have

$$\begin{cases} c_1 + c_3 = 1 \\ c_1 + c_2 + c_3\frac{1}{2} = \frac{3}{2} \\ c_1 + 2c_2 + c_3\frac{1}{4} = \frac{5}{4} \end{cases}$$

Then, $c_1 = 4$, $c_2 = -1$ and $c_3 = -3$.

$$T(n) = 4 - \log n - 3\frac{1}{n}$$

As T assumes negative values for n greater than a N, it doesn't represent an algorithm.

G **(0.25pt)** $T(n) = 3T(n/9) + n^2$ (using recurrence tree)

Using the recurrence tree method we can see that the number of times that it is going to repeat is $\log_9 n$. And it repeats the equation $\frac{n^2}{27^k}$ each time, where k is the step that algorithm is in.

So, $T(n) = \sum_{k=1}^{\lceil \log_9 n \rceil} (\frac{n^2}{27^k}) + n^2 = n^2 \frac{\frac{1}{27}(1 - (\frac{1}{27})^{\log_9 n})}{1 - \frac{1}{27}} + n^2 = \frac{n^2}{26} - \frac{\sqrt{n}}{26}$

Thus, $T(n) = O(n^2)$.

H **(0.25pt)** $T(n) = 4T(n/2) + cn; T(1) = 1$ (using recurrence tree)

Using the recurrence tree method we can see that the number of times that it is going to repeat is $\log_2 n$. And it repeats the equation $2^k cn$ each time, where k is the step that algorithm is in.

So, $T(n) = \sum_{k=1}^{\lceil \log_2 n \rceil} (2^k cn) + cn = 2cn^2 - cn$

Using $T(1) = 1, T(1) = 2c - c = c = 1$.

Then, $T(n) = 2n^2 - n$.

Thus, $T(n) = O(n^2)$.

3 LinkedList Improvements [6pts]

A **(1.0pt) Insert/remove:** The file "list.cpp" has the base code of the LinkedList class. In this class, we are implementing the "find" function using double pointers, so that we can use it in the "insert" and "remove" functions. Your job is to implement these two functions by following the comments in the base code.

B **(1.0pt) Constructor:** You must improve the constructor. In this version, we must be able to create a list using one of the following two options:

```
int main() {
    // Option 1: Variadic Templates
    LinkedList list1(1, 2, 10, 2, 3);
    list1.print();
    // Option 2: Initialization List
    LinkedList list2({1, 2, 10, 2, 3});
    list1.print();
}
```

For the first option you might use the *initializer_list* from STL and for the second option the *Variadic Templates* feature. An example for both cases can be found at `https://bit.ly/2EFy4fc`. Any of them is a valid answer but I recommend to try both options.

C **(0.5pt) Destructor:** Your job here is to release the dynamic memory reserved by the new operator. In this method, you must call `delete pNode` for each node in the list. To check that your code is working print something in the Node destructor to check that all the nodes are destructed.

D **(1.0pt) Templates:** Modify your class so it must support any data type. Remember we saw templates in classes. The following code should work if succeed this step.

```
int main() {
    // create a linked list for three data types
    LinkedList<int> ilist(1, 10, 2);
    ilist.print();
    // output: 1 10 2
    LinkedList<float> flist(1.2, 1.4, 100000);
    flist.print();
    // output: 1.2 1.4 100000
    LinkedList<std::string> slist("one", "two", "three");
    slist.print();
    // output: one two three
}
```

E **(1.5pt) Iterators:** Here, you must implement all the necessary to make your LinkedList to work using iterators. It should be similar to the STL data structures in C++. The following code should work if you succeed.

```
int main() {
    LinkedList<int> ilist(1, 2, 10, 2, 3);
    LinkedList<int >::Iterator it;
    for (it=ilist.begin(); it!=ilist.end(); ++it) {
        std::cout << *it << " ";
    }
    cout << endl;
}
```

Just because this might be your first-time implementation and Iterator, you must include the following class inside your LinkedList class. Be careful, the code below is only a skeleton, you have to implement the methods.

```
class Iterator {
public:
    Node<T> *pNodo;
public:
    Iterator() { ... }
    Iterator(Node<T> *p) { ... }

    bool operator!=(Iterator it) { ... }
    Iterator operator++() { ... }
    T& operator*() { ... }
};
```

In addition, you have to implement the following methods in the LinkedList class.

```
    Iterator begin()  { ... }
    Iterator end() { ... }
```

F **(1.0pt) Exceptions:** Our current implementation of the remove method does nothing if the element is not found in the list. However, the correct way to handle an exception is to throw an exception if something unusual happens. First, you must create an exception class (check this link to have an idea `https://bit.ly/2HXAwx3`). Then, modify your remove method to throws an exception if the element is not found. I will test using the following code:

```
int main() {
    LinkedList<int> ilist(1, 2, 10, 2, 3);
    // if we remove an item that doesn't exist it should throw an error
    ilist.remove(20);
    // output: libc++abi.dylib: terminating with uncaught
    //         exception of type NotFoundException: Element not found

}
```

Finally, you must use handle correctly the exception using try and catch structure.

```
int main() {
    // Correct way to handle exceptions
    try {
        ilist.remove(20);
    } catch (const NotFoundException& e) {
        cerr << e.what();
    }
    // output: Element not found
}
```