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)** $f(n) = \log n^2$; $g(n) = \log n + 5$
  - Neste caso temos que $f(n) = \Theta(g(n))$, pois usando a definição, $\exists c1, c2$ tal que $c2 * g(n) \leq f(n) \leq c1 * g(n)$. Basta tomarmos c2 = 1 e c1 = 2, dai teremos: $1 * (\log n + 5) \leq \log n^2 = 2 * \log n \leq 2 * (\log n + 5)$, para n grande.
- **(0.25pt)** $f(n) = \log^2 n$; $g(n) = \log n$
  - Nesse caso teremos que $f(n) = \Omega(g(n))$, pois da definição segue que, $\forall$ c real , sempre existirá n tal que:
    $\log^2 n \geq c * \log n \rightarrow \log n \geq c$
    pois $f(n)$ é crescente e $t(n) = c$ é uma constante.
- **(0.25pt)** $f(n) = n \log n + n$; $g(n) = \log n$
  - Esse caso é bem análogo ao de cima, pois $f(n) = \Omega(g(n))$, tambem. O argumento é o mesmo, $\forall c$ real escolhido, sempre teremos que:
    $n \log n + n \geq c * \log n \rightarrow n \geq c$
    ou seja, basta tomar um n maior que esse c que a desigualdade ja passa a ser válida.
- **(0.25pt)** $f(n) = 2^n$; $g(n) = 10n^2$
  - Usando a mesma definição , consideremos as funçoes continuas em R por $f(x) = 2^x$ e $h(x) = c * x^2$, se provarmos que $f(x) \geq h(x)$, concluiremos que $f(n) = \Omega(g(n))$. Vamos analisar a taxa de variação da taxa de variação de ambas as funçoes. Note que a taxa de variação da taxa de variação de 2^x é $(\log 2)^2 * 2^x$ e a de $10c * x^2$ é $20c$, ou seja, constante, assim $\forall$ c real , existe um n tal que a taxa de variação da taxa de variação é maior para $f(x)$, assim em algum momento a taxa de variação de $f(x)$ irá ser maior que a de $h(x)$ para todo x maior que esse dado primeiro e dai em algum momento $f(x) \geq h(x)$, logo , $f(n) = \Omega(g(n))$.

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

- Neste item, temos que provar que $\exists c, d$ reais, tal que:
  $c * n^3 \leq n^3 - 3n^2 - n + 1 \leq d * n^3$. Tome d = 4, dai de fato $n^3 - 3n^2 - n + 1 \leq 4 * n^3$, pois nesse caso cada cubo de n é maior ou igual a cada parcela do termo da esquerda. Para o lado esquerdo da desigualdade, queremos provar que $c * n^3 \leq n^3 - 3n^2 - n + 1 = n(n^2 - 3n - 1) + 1$. Podemos dividir tudo por n, considerando que n é diferente de zero e ignorar o ultimo termo que tenderá a zero. Assim ficamos com:
  $c * n^2 \leq n^2 - 3n - 1 \rightarrow 0 \leq n^2(1 - c) - 3n - 1$. Resolvendo , usando a formula resolutiva do

segundo grau, concluimos que DELTA = 0 para termos somente valores não negativos da função. Com isso, concluimos que $c = 13/4$ e assim segue a desigualdade do lado esquerdo.

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

- Temos que provar que para existe um c real tal que $n^2 \leq c * 2^n$. Vamos usar indução em n. Para o caso base, veja que de fato, $1^2 \leq c * 2^1$ (tome um c grande, por exemplo 1000). Suponhamos que exista um certo n tal que $n^2 \leq c * 2^n$ seja verdadeiro. Multiplicando por 2 em ambos os lados ficamos com $2n^2 \leq c * 2^{n+1}$ e note que $(n+1)^2 = n^2 + 2n + 1 \leq 2n^2 \rightarrow 2n + 1 \leq n^2$, logo $n \geq 3$. Logo para $n \geq 3$, segue a desigualdade. Para n = 1 ja provamos, e se n = 2 , $2^2 \leq c * 2^2$ de fato.

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 $n > 1; T(0) = 0; T(1) = 1$

- Veja que podemos manipular a equação para chegar em:
  $T(n) - 3T(n-1) - 4T(n-2) = 0$
  que pode ser resumida em uma equação linear homogenea da forma:
  $x^2 - 3x - 4 = 0$, cujas raizes sao -1 e 4. Assim o termo geral de $t(n)$ é dado por $T(n) = c1 * (-1)^n + c2 * 4^n$. Usando as condições iniciais conseguimos chegar a um sistema de duas incognitas:

$$\begin{aligned} c1 + c2 &= 0 \\ -c1 + 4c2 &= 1 \end{aligned}$$

  cuja solução é dada por $c1 = -1/5$ e $c2 = 1/5$. Assim, $T(n) = \frac{-1}{5} * (-1)^2 + \frac{1}{5} * 4^n$

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

- Para resolver essa recorrencia utilizaremos o método descrito na segunda parte do pdf. Note que podemos escreve-la da forma:

$$T(n) - 2T(n-1) = -(n+5)3^n$$

  cuja equação homogenea é:

$$(x-2)(x-3)^2 = 0$$

  ou seja, $T(n) = c1 * 2^n + c2 * 3^n + n * c3 * 3^n$, e usando que T(0) = 0, concluimos que c1 = c2, logo finalmente teremos $T(n) = c1 * 2^n + c1 * 3^n + n * c3 * 3^n$.

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

- De acordo com o metodo mestre, identificamos essa recorrencia como uma que se encaixa em suas hipóteses. Assim, temos que a = 4, d = 2, b = 2 e podemos concluir que $a = b^d$ ou seja , $4 = 2^2$, assim $T(n) = O(n^2 log n)$.

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

- Podemos usar novamente o método maestro. Veja que nessa versão, teremos:

$$\text{Se } f(n) = O(n^{log_b a - e}), \text{ então } T(n) = \theta(n^{log_b a})$$

$$\text{Se } f(n) = \theta(n^{log_b a}), \text{ então } T(n) = \theta(n^{log_b a} * lgn)$$

$$\text{Se } f(n) = \omega(n^{log_b a + e}), \text{ então } T(n) = \theta(f(n))$$

Veja que nesse caso, $f(n)$ se encaixa no caso do meio, pois no primero $nlogn$ sempre será maior que $1/n$ para n grande. Na segunda, para n grande, $nlogn$ é maior que $c * n$ para qualquer c. Logo, $T(n) = \theta(n^{log_b a} * lgn)$.

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

- Novamente podemos utilizar o método visto na primeira questão para resolver essa recorrencia. A equação caracteristica dessa recorrencia é dada por:

$$x^3 - x^2 - 2x + 2 = 0$$

Veja que 1 é raiz desse polinomio. Logo, usando o teorema da algebra, podemos dividir esse polinomio por $x - 1$ para diminuir seu grau em 1 e encontrar as duas ultimas soluções. Veja que, efetuando esse procedimento concluimos que:

$$x^3 - x^2 - 2x + 2 = (x-1)(x^2 - 2) = 0$$

cujas raizes facilmente sao verificadas por $2^{1/2}$,$-2^{1/2}$ e 1. Logo ,$T(n) = c1 * 1^n + c2 * 2^{n/2}$. Usando as condições iniciais concluimos que:

$$T(n) = 100 + 6 * 2^{n/2}$$

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

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

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

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
}
```