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

Homework 2
Due: April 03, 2020

Homework 2

Cristiana Aparecida Nogueira Couto

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)** $f(n) = \log n^2$; $g(n) = \log n + 5$
  Resposta: $f(n) = \Theta(g(n))$.
  Como $f(n) = log(n^2) = 2log(n)$, logo podemos encontrar constantes c_1 e $c_2 \in R$, e $n_0 \in N$ tais que $\forall n \in N, (n \geq n_0 \Rightarrow c_1 g(n) \leq f(n) \leq c_2 g(n))$.
- **(0.25pt)** $f(n) = \log^2 n$; $g(n) = \log n$
  Resposta: $f(n) = \Omega(g(n))$.
  $f(n) = log(n)^2 = log(n) * log(n)$.
  Não existe uma constante c_2 e um $n_0 \in N$, tal que $\forall n \in N, (n \geq n_0 \Rightarrow f(n) \leq c_2 g(n))$, já que $log(n)$ tende a infinito quando n vai pra infinito. Em algum momento o valor de $log(n)$ ultrapassará a constante.
  Assim, temos somente que $\exists c_1 \in R$, e $n_0 \in N$ tais que $\forall n \in N, (n \geq n_0 \Rightarrow c_1 g(n) \leq f(n)$
- **(0.25pt)** $f(n) = n \log n + n$; $g(n) = \log n$
  Resposta: $f(n) = \Omega(g(n))$.
  Para esse item, o raciocínio é semelhante ao do item anterior, pois temos um n (variável) multiplicando $log(n)$ em f(n). Logo, $\exists c_1 \in R$, e $n_0 \in N$ tais que $\forall n \in N, (n \geq n_0 \Rightarrow c_1 g(n) \leq f(n))$.
- **(0.25pt)** $f(n) = 2^n$; $g(n) = 10n^2$
  Resposta: $f(n) = \Omega(g(n))$.
  Podemos provar por indução que a partir de um certo $n \in N$, $2^n > n^2$. A seguir:
  Caso base : $n = 5$.
  Hipotése indutiva: Para um certo $k \geq 5 \in N$, vale que $2^k \geq k^2$.
  Passo de indução: $2^{k+1} > 2k^2$
  Como $(n-1)^2 > 2, n \geq 5$.
  Desenvolvendo $n^2 + n^2 - 2n + 1 > 2 + n^2$
  obtemos: $2n^2 > (n+1)^2$.
  Logo, $\forall n \in N, 2^n \geq k^2$.
  Assim, temos que: $\exists c \in R$, e $n_0 \in N$ tais que $\forall n \in N, (n \geq n_0 \Rightarrow c.g(n) \leq f(n))$. Por maior que seja a constante multiplicando $g(n)$, em algum momento $f(n)$ irá ultrapassar o valor de $g(n)$.

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

Para provar começamos mostrando que $n^3 - 3n^2 - n + 1 = O(n^3)$.

Seja $c_1 = 6$ e $n_1 = 1$.

$n^3 - 3n^2 - n + 1 \leq n^3 + 3n^2 + n + 1 \leq n^3 + 3n^3 + n^3 + n^3 = 6n^3$.

$\forall n \in N, (n > 1 \Rightarrow n^3 - 3n^2 - n + 1 \leq 6n^3)$.

Agora, provaremos que $n^3 - 3n^2 - n + 1 = \Omega(n^3)$.

Seja $c_2 = \frac{1}{3}$ e $n_2 = 5$

$n^3 - 3n^2 - n + 1 \geq n^3 - 3n^2 - n = n^3 - (3n^2 + n) \geq \frac{1}{3}n^3$

A última desigualdade vem de $3n^2 + n \leq \frac{2}{3}n^3$, para todo $n \geq 5$.

Logo, concluimos que $\forall n \in N, (n \geq n_0 \Rightarrow \frac{1}{2}n^3 \leq n^3 - 3n^2 - n + 1 \leq 6n^3$, onde $n_0 = max(n_1, n_2) = 5$.

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

Seja $c = 1$ e $n_0 = 5$.

Temos que $n^2 < 2^n$ para todo $n \geq 5$, como mostrado na resposta 4 do item A. Desse modo, $\forall n \in N, (n > 5 \Rightarrow n^2 \leq 2^n)$, como queríamos provar.

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

A equação caracteristíca é dada por: $x^2 - 3x - 4x = 0$

Resolvendo a equação obtemos que:

$T(n) = c_1 * 4^n + c_2(-1)^n$

Dadas as condições iniciais, têm-se:

$c_1 + c_2 = 0$ e $4 * c_1 - c_2 = 1$

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

$t(n) = O(4^n)$.

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

Equação caracteristíca: $(x-2)(x-3)^2 = 0$

$T(n) = c_1 2^n + c_2 3^n + c_3 n 3^n$

Dadas as condições iniciais temos que: $c_1 + c_2 = 0$.

Para encontrar os valores das constantes, calculamos a seguir: $t(1) = 2t(0) - (1+5).3^1 = 0 - 18 = -18$. Assim, a resolução da equação caracteristíca mostra que esse caso não pode corresponder a um caso real de cálculo de tempo de execução, pois há valores negativos para $t(n)$.

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

Aplicando uma mudança de variáveis, temos: $n = 2^k$

$t(2^k) = 4t(2^{k-1}) + (2^k)^2$, para $k \geq 2$

$s(k) = t(2^k)$

$s(k) = 4s(k-1) + 4^k$

A equação caracteristíca para $s(k)$ é dada por: $(x-4)^2 = 0$

Assim, $s(k) = c_1 4^k + c_2 k 4^k$ e $t(n) = c_1 n^2 + c_2 n^2 logn$

Calculando as constantes, obtemos: $t(n) = n^2 + n^2 logn$.

$t(n) = O(n^2 logn)$

D **(0.25pt)** $T(n) = 2T(n/2) + n \log n$ if $n > 1$; n power of $2, t(1) = 0$.

Aplicando uma mudança de variáveis, temos: $n = 2^k$

$t(2^k) = 2t(2^{k-1}) + 2^k k$, para $k \geq 0$

$s(k) = t(2^k)$

$s(k) = 2s(k-1) + 2^k k$

A equação caracteristíca para $s(k)$ é dada por: $(x-2)^3 = 0$

Assim, $s(k) = c_1 2^k + c_2 k 2^k + c_3 k^2 2^k$ e $t(n) = c_1 n + c_2 nlogn + c_3 nlog^2 n$.

Calculando as constantes, obtemos: $c_1 = 0, c_2 = \frac{1}{2}, c_3 = \frac{1}{2}$

$t(n) = \frac{1}{2} nlogn + \frac{1}{2} nlog^2 n$

$t(n) = O(nlog^2 n)$

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

A equação característica é dada por: $x^3 - x^2 - 2x + 2 = 0$. Cujas soluções são $\sqrt{2}$ e 1.

Assim, $t(n) = c_1 1^n + c_2 (\sqrt{2})^n + c_2 n (\sqrt{2})^n$.

$t(n) = O(n\sqrt{2}^n)$

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

Fazendo a substituição $n = 2^k$, obtemos: $t(2^k) = \frac{3}{2} t(2^{k-1}) - \frac{1}{2} t(2^{k-2}) - \frac{1}{2}$.

$s(k) = t(2^k)$

$s(k) = \frac{3}{2} s(k-1) - \frac{1}{2} s(k-2) - \frac{1}{2}$

A equação característica é: $x^2 - \frac{3}{2} x + \frac{1}{2}$.

Portanto, $s(k) = c_1 1^k + c_2 (\frac{1}{2})^k$

$s(k) = c_1 + \frac{c_2}{2^k}$.

Calculando as constantes a partir das condições iniciais, temos $c_1 = 2$ e $c_2 = -1$.

Sendo assim, $t(n) = 2 - \frac{1}{n}$.

$t(n) = O(1)$

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

No nível i o tempo gasto, sem contar as chamadas recursivas, é de $(\frac{3}{81})^i n^2$. Cada nó pode ser reescrito com contribuição de um tempo computacional de $t(\frac{n}{9^i})$. No nível zero o tempo gasto é n^2, no primeiro nível $\frac{3^1}{81^1} . n^2$ e assim sucessivamente. O que pode ser verificado construindo uma representação visual da árvore é que cada subproblema no último nível contribui $t(1)$ para o total.

Precisamos encontrar a altura da árvore. Os subproblemas chegam a $t(1)$ quando $1 = \frac{n}{9^i}$. Sendo assim, $i = log_9 n$ e a árvore tem $log_9 n + 1$ níveis. Nesse último nível cada nó contribui com 1^2, de onde podemos calcular que há $3^{log_9 n} = n^{log_9 3} = n^2$ deles.

O custo total da árvore é: $t(n) = n^2 + \frac{3}{81}n^2 + (\frac{3}{81})^2 n^2 + ... + (\frac{3}{81})^{log_9(n-1)} + n^2 \leq$

$$n^2 \sum_{i=0}^{\infty} (\frac{3}{81})^i + n^2 = n^2 . \frac{1}{1 - \frac{3}{81}} + n^2$$

Assim, $t(n) = O(n^2)$.

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

Seguindo raciocínio semelhante ao item anterior: no nível i o tempo gasto, não contando chamadas recursivas, é de $(\frac{4}{2})^i cn = 2^i cn$. No último nível, onde cada subproblema contribui $t(1)$, temos $i = log_2(n)$.

Portanto, a altura da árvore é $log_2 n + 1$ e o total da árvore completa é: $t(n) = cn + 2cn + 2^2 cn + ... + 2^{log_2(n-1)} cn + cn^2 =$

$$\sum_{i=0}^{log_2(n-1)} (2)^i + cn^2 = cn^2 - cn + cn^2 = 2cn^2 - cn$$

.

Dada a condição inicial, $t(1) = 1$, temos $t(n) = 2n^2 - n$ e portanto, $t(n) = O(n^2)$.

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
}
```

4 References

https://web.stanford.edu/class/archive/cs/cs161/cs161.1168/lecture3.pdf