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

Homework 1
Due:: July 19, 2019

Aluno: Franklin Alves de Oliveira

Homework 1

1 Induction [3pts]

Answers should be written in this document.

1. Prove by Induction that: $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \qquad \forall n \geq 0$

   Resposta:

   (i) Caso base: considere $n = 1$. Então, $\sum_{i=1}^{n} i^2 = 1^2 = \frac{1(1+1)(2\cdot 1+1)}{6} = \frac{2\cdot 3}{6} = \frac{6}{6}$ o que é verdadeiro. Então, a proposição é válida para $n = 1$. Tomemos agora o passo de indução.

   (ii) Passo indutivo: Suponha que, para algum $n \in \mathbb{N}$, a seguinte proposição é verdadeira:

   $$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \tag{1}$$

   Queremos mostrar que, dado que a proposição é válida para $n \in \mathbb{N}$, ela também será válida para $n + 1 \in \mathbb{N}$, isto é, que

   $$\sum_{i=1}^{n+1} i^2 = \frac{(n+1)\cdot[(n+1)+1]\cdot[2(n+1)+1])}{6} = \frac{(n+1)(n+2)(2n+3)}{6}$$

   .

   Partindo da hipótese de indução, de 1, temos:

   $$\sum_{i=1}^{n} i^2 + (n+1)^2 = \sum_{i=1}^{n+1} i^2 = \frac{n(n+1)(2n+1)}{6} + (n+1)^2 = \frac{n(n+1)(2n+1) + 6(n+1)^2}{6} =$$

   $$= \frac{(n+1)}{6}\left[n(2n+1) + 6(n+1)\right] = \frac{(n+1)}{6}\left[2n^2 + n + 6n + 6\right] =$$

   $$= \frac{(n+1)}{6}\left[2n^2 + 7n + 6\right] = \frac{(n+1)}{6}\left[(n+2)(2n+3)\right]$$

   Como queríamos demonstrar. ■

2. Prove by Induction that: $\forall n \geq 7$ it is true $3^n < n!$

   Resposta:

(i) Caso base: Tome $n = 7$. Assim, $3^7 < 7! \iff 2187 < 5040$, o que é verdade.

(ii) Passo indutivo: Suponha, para um certo $n \in \mathbb{N}$, que $3^n < n!$. Então, multiplicando ambos os lados por $(n+1)$, temos:

$$3^n(n+1) < n! \cdot (n+1) \Rightarrow n3^n + 3^n < (n+1)!$$

Como estamos tomando $n \geq 7$, sabemos que $n + 1 > 3$. Portanto,

$$3^{n+1} < (n+1) \cdot 3^n < (n+1)!$$

■

3. Prove by Induction that $\forall n \geq 0$

$$\left\lceil \frac{n}{2} \right\rceil = \begin{cases} \frac{n}{2} & \text{si } n \text{ es par} \\ \frac{n+1}{2} & \text{si } n \text{ es impar} \end{cases}$$

Resposta:

(i) Caso base: Considere $n = 0$ (par) ou $n = 1$ (ímpar). Então,

$$n = 0 \Rightarrow \left\lceil \frac{n}{2} \right\rceil = \frac{0}{2} = 0$$

$$n = 1 \Rightarrow \left\lceil \frac{n}{2} \right\rceil = \frac{1+1}{2} = 1$$

Logo, a proposição é verdadeira para o caso base.

(ii) Passo indutivo: Suponha que, para algum $n \in \mathbb{N}$, é válido que:

$$\left\lceil \frac{n}{2} \right\rceil = \begin{cases} \frac{n}{2} & \text{se } n \text{ é par} \\ \frac{n+1}{2} & \text{se } n \text{ é ímpar} \end{cases}$$

Logo, se n é par, $n + 1$ é ímpar. Nesse caso, teremos:

$$n = 2m, m \in \mathbb{Z}_+ \Rightarrow \left\lceil \frac{n+1}{2} \right\rceil = \left\lceil \frac{2m+1}{2} \right\rceil = \left\lceil m + \frac{1}{2} \right\rceil = m + 1 = \frac{(2m+1)+1}{2} = \frac{(n+1)+1}{2} \tag{2}$$

Agora, se n é ímpar, então $n + 1$ é par e temos:

$$n = 2m + 1, m \in \mathbb{Z}_+ \Rightarrow \left\lceil \frac{n+1}{2} \right\rceil = \left\lceil \frac{2m+2}{2} \right\rceil = \lceil m + 1 \rceil = m + 1 = \frac{(2m+1)}{2} = \frac{(n+1)}{2} \tag{3}$$

Que é exatamente a proposição que queríamos demonstrar. ■

4. Prove by induction that a number is divisible by 3 if and only if the sum of its digits is divisible by 3.

Resposta:

(i) Caso base:

(ii) Passo indutivo:

5. Prove that any integer greater than 59 can be formed using only 7 and 11 cent coins.

Resposta:

(i) Caso base: Considere $n = 60$. Nesse caso, podemos escrever:

$$60 = 7 \cdot 7 + 1 \cdot 11$$

Assim, conseguimos 60 centavos se adicionarmos 7 moedas de 7 centavos a uma moeda de 11 centavos. Logo, a proposição é válida para $n = 60$.

(ii) Passo indutivo: Suponha que, para $n \in \mathbb{N}$, a proposição é verdadeira. Isto é, $n = 7a + 11b$, com $a, b \in \mathbb{Z}_+$.
Assim, para $n + 1$, temos:

$$n + 1 = 7w + 11z.$$

Tome $w = a - 3$ e $z = b + 2$. Assim,

$$7w + 11z = 7(a - 3) + 11(b + 2) = 7a - 21 + 11b + 22 = 7a + 11b + 1 = n + 1$$

■

6. Prove by induction that $F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$

Resposta:

OBS: Vamos considerar $F_1 = F_2 = 1$, $F_3 = 2$ e $F_n = F_{n-1} + F_{n-2}$, onde F_n é o n-ésimo termo da sequência de Fibonacci.

(i) Caso base: Consideremos o caso em que $n = 1$. Então,

$$F_{k+1} = F_k F_2 + F_{k-1} F_1 = F_k + F_{k-1}$$

Que é exatamente a expressão que define a sequência de Fibonacci. Logo, a proposição é verdadeira para o caso base.

(ii) Passo indutivo: Considere que, para um certo $n \in \mathbb{N}$, é válido que $F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$ e também que $F_{n+k-1} = F_k F_n + F_{k-1} F_{n-1}$. Queremos mostrar que essa igualdade também é verdadeira para $n + 1 \in \mathbb{N}$, isto é, $F_{n+1+k} = F_k F_{n+2} + F_{k-1} F_{n+1}$.
Ora, sabemos que

$$F_{n+k+1} = F_{n+k} + F_{n+k-1}$$

Pela hipótese de indução, podemos escrever:

$$F_{n+k+1} = F_k F_{n+1} + F_{k-1} F_n + F_k F_n + F_{k-1} F_{n-1} = F_k(F_{n+1} + F_n) + F_{k-1}(F_n + F_{n-1}) \Rightarrow$$

$$\Rightarrow F_{n+k+1} = F_k F_{n+2} + F_{k-1} F_n$$

■

7. Prove by induction in n that $\sum_{m=0}^{n} \binom{n}{m} = 2^n$

Resposta:

OBS: Esse resultado é um caso particular do **teorema binomial**. Vamos prová-lo por indução e, em seguida, apondar o caso particular que foi pedido para provar nesse item.

Teorema Binomial: Para qualquer $n \in \mathbb{N}$,

$$(x+y)^n = \sum_{m=0}^{n} \binom{n}{m} x^m y^{n-m}$$

Provemos esse teorema por indução.

(i) Caso base: para $n = 1$, temos:

$$(x+y)^1 = x + y = \binom{1}{0} x^0 y^{1-0} + \binom{1}{1} x^1 y^{1-1} = \sum_{m=0}^{1} \binom{1}{m} x^m y^{1-m}$$

Tomando $x = y = 1$, temos o seguinte resultado:

$$(1+1)^1 = 2 = \binom{1}{0} 1^0 1^{1-0} + \binom{1}{1} 1^1 1^{1-1} = \sum_{m=0}^{1} \binom{1}{m}$$

Portanto, a proposição é válida para $n = 1$.

(ii) Passo indutivo: Suponha que é verdade que

$$(x+y)^n = \sum_{m=0}^{n} \binom{n}{m} x^m y^{n-m} \tag{4}$$

Queremos mostrar que

$$(x+y)^{n+1} = \sum_{m=0}^{n+1} \binom{n+1}{m} x^m y^{n+1-m}$$

Para isso, considere o termo $(x+y)^{n+1}$. Ora,

$$(x+y)^{n+1} = (x+y)(x+y)^n = (x+y)\left[\sum_{m=0}^{n} \binom{n}{m} x^m y^{n-m}\right] =$$

$$= \sum_{m=0}^{n} \binom{n}{m} x^{m+1} y^{n-m} + \sum_{m=0}^{n} \binom{n}{m} x^m y^{n-m+1} =$$

$$= x^{n+1} + \sum_{m=0}^{n-1} \binom{n}{m} x^{n+1} y^{n-m} + \sum_{m=0}^{n} \binom{n}{m} x^m y^{n-m+1} =$$

$$= x^{n+1} + \sum_{m=1}^{n} \binom{n}{m-1} x^n y^{n+1-m} + \sum_{m=0}^{n} \binom{n}{m} x^m y^{n-m+1} =$$

$$= x^{n+1} + y^{n+1} + \sum_{m=1}^{n} \binom{n}{m-1} x^n y^{n+1-m} + \sum_{m=1}^{n} \binom{n}{m} x^m y^{n-m+1} =$$

$$= x^{n+1} + y^{n+1} + \sum_{m=1}^{n} \left[\binom{n}{m-1} + \binom{n}{m}\right] (x^m y^{n-m+1}) =$$

$$= x^{n+1} + y^{n+1} + \sum_{m=1}^{n} \binom{n+1}{m} x^m y^{n-m+1} = \sum_{m=0}^{n+1} \binom{n+1}{m} x^m y^{n-m+1} \tag{5}$$

Com isso, provamos, por indução, o teorema binomial. Agora, fazendo $x = y = 1$ na equação 5, temos o seguinte resultado:

$$2^{n+1} = \sum_{m=0}^{n+1} \binom{n+1}{m} 1^m 1^{n-m+1} = \sum_{m=0}^{n+1} \binom{n+1}{m}$$

Logo, do teorema binomial, segue que $2^n = \sum_{m=0}^{n} \binom{n}{m}$, como queríamos demonstrar.

■

8. Prove by induction that a graph with n vertices can have at most $\frac{n(n-1)}{2}$ edges.

   Resposta:

   (i) Caso base: $n = 1$. Um grafo com apenas 1 vértice não tem nenhuma aresta. Além disso,

   $$\frac{1(1-1)}{2} = 0$$

   Portanto, a sentença é válida para $n = 1$.

   (ii) Passo indutivo: Suponha que um grafo com n vértices possui $\frac{n(n-1)}{2}$ vértices. Se adicionarmos um vértice, o número máximo de conexões que ele terá é n. Assim, o $n+1$-ésimo vértice terá, no máximo, n conexões. Logo,

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

   Que é, exatamente, o número máximo de arestas que um grafo com $n+1$ vértices pode ter.

   Com isso, mostramos que a sentença é válida $\forall\ n \in \mathbb{N}$. ■

9. Prove by induction that a complete binary tree[1] with n levels has $2^n - 1$ vertices.

   Resposta:

   (i) Caso base: Considere uma árvore binária com apenas 1 nível. Nesse caso, ela tem apenas 1 vértice. Além disso, $2^1 - 1 = 1$, que é exatamente o número de vértices de uma árvore binária com um só nível. Assim, provamos que a proposição é válida para $n = 1$.

   (ii) Passo indutivo: Suponha que uma árvore binária de decisão com n níveis possui $2^n - 1$ vértices. Se adicionarmos um nível à essa árvore, dado que cada vértice dá origem a dois novos vértices (uma vez que a árvore é binária), ao adicionarmos o $n+1$-ésimo nível à árvore, aumentamos o número de vértices em 2^n. Ainda, $2^n - 1 + 2^n = 2 \cdot 2^n - 1 = 2^{n+1} - 1$, que é exatamente o que queríamos provar.

   ■

10. A polygon is convex if each pair of points in the polygon can be joined by a straight line that does not leave the polygon. Prove by induction in $n > 3$ that the sum of the angles of a polygon of n vertices is $180(n-2)$.

    Resposta:

    (i) Caso base: Considere $n = 4$, ou seja, um polígono com 4 lados. Sabemos que a soma dos seus ângulos é 360 (que é exatamente a soma dos ângulos de um quadrado). Ainda, $180(4-2) = 180 \cdot 2 = 360$. Portanto, a afirmativa é verdadeira para $n = 4$.

    (ii) Passo indutivo: Agora, suponha que a soma dos ângulos internos de um polígono com n lados, $n \in \mathbb{N}$, é exatamente $180(n-2)$. Queremos mostrar que a soma dos ângulos internos de um polígono com $n+1$ lados é $180(n+1-2) = 180(n-1)$

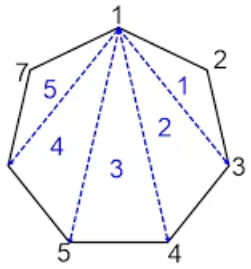

Figura 1: Polígono de 7 lados subdividido em 5 triângulos

Note que um polígono com n lados pode ser seccionado em $n-2$ triângulos, como mostra a figura a seguir:

Sabemos que a soma dos ângulos internos de um triângulo é 180 graus. Note que, se adicionamos mais um lado a um polígono de n lados, conseguimos formar mais um triângulo em seu interior. Assim, adicionamos 180 graus à soma dos seus ângulos internos. Em outras palavras, a soma dos ângulos internos de um polígono com $n+1$ lados será:

$$180(n-2) + 180 = 180\,[1 + (n-2)] = 180(n-1)$$

Que é exatamente o que queríamos demonstrar. ■

2 Correctness of bubblesort [2pts]

Bubblesort is a popular, but inefficient, sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order.

Algorithm 1: BUBBLESORT(A)

```
1 for i = 1 to A.length − 1 do
2 |   for j = A.length downto i + 1 do
3 |   |   if A[j] < A[j − 1] then
4 |   |   |   exchange A[j] with A[j − 1]
5 |   |   end
6 |   end
7 end
```

A Let A' denote the output of BUBBLESORT(A). To prove that BUBBLESORT is correct, we need to prove that it terminates and that

$$A'[1] \leq A'[2] \leq ... \leq A'[n] \tag{6}$$

where $n = A.length$. In order to show that BUBBLESORT actually sorts, what else do we need to prove?

[1] http://web.cecs.pdx.edu/ sheard/course/Cs163/Doc/FullvsComplete.html

Resposta:

Além do que foi mencionado, precisamos mostrar que os elementos da lista ordenada A' e da lista inicial A são os mesmos, mas não necessariamente na mesma ordem.

The next two parts will prove inequality (6).

B State precisely a loop invariant for the **for** loop in lines 2–6, and prove that this loop invariant holds. Your proof should use the structure of the loop invariant proof presented in this chapter.

Resposta:

Para provar que o *loop* de um algoritmo cumpre seu objetivo, devemos mostrar que a condição invariante analisada no *loop* é verdadeira em cada uma das seguintes etapas:

- **Inicialização:** a condição é verdadeira antes da primeira iteração do *loop*.
- **Manutenção:** a condição é verdadeira antes da n-ésima iteração do *loop* e também é verdadeira antes da $(n+1)$-ésima iteração (ao fim da n-ésima).
- **Finalização:** ao fim da última iteração do *loop*, o invariante permanece verdadeiro.

O invariante é uma propriedade que permanece verdadeira em cada uma das etapas e ajuda a analisar se o algoritmo cumpre apropriadamente o seu objetivo (está correto).

No caso do **loop das linhas 2-6**, podemos destacar a seguinte propriedade:

P1: A cada iteração, o elemento $A[j]$ é o menor elemento da lista $A[j, \ldots, A.length]$

Vamos provar que essa propriedade se cumpre em cada uma das etapas citadas acima.

- Inicialização: Como $j = A.length$, temos que a lista $A[j, \ldots, A.length] = A[j]$. Logo, trivialmente, $A[j]$ é seu maior elemento.
- Manutenção: A cada iteração, se $A[j-1] < A[j]$, os termos são trocados. Assim, na lista alterada, $A[j-1, j]$, $A[j-1]$ é o menor dos dois elementos. Assim, a lista $A[j-1, \ldots, A.length]$ sempre terá $A[j-1]$ como menor elemento.
- Finalização: Quando $j = i+1$, temos que o elemento $A[i] = A[j-1]$ será o menor elemento da lista $A[i, \ldots, A.length]$

Com isso, provamos que a propriedade **P1** permanece verdadeira em cada uma das etapas. Logo, $P1$ é o invariante do *loop* das linhas 2-6.

C Using the termination condition of the loop invariant proved in part (B), state a loop invariant for the for loop in lines 1–7 that will allow you to prove inequality (6). Your proof should use the structure of the loop invariant proof presented in this chapter.

Vamos mostrar que a propriedade **P2** abaixo é o invariante do ***loop* externo das linhas 1-7**.

Resposta:

P2: A lista $A[1, \ldots, j]$ está ordenada, i.e., $A[1] \leq A[k] \leq A[j],\ \forall\ k = 1, \ldots, j$.

- Inicialização: Antes da primeira iteração, quando $i = 1$, temos $A[1, \ldots, i] = A[1]$. Portanto, a lista está trivialmente em ordem crescente.
- Manutenção: Como resultado do *loop* interno (das linhas 2-6), a lista $A[i, \ldots, A.length]$ tem $A[i]$ como menor elemento. Assim, $A[i] \leq A[j],\ \forall\ j = i+1, \ldots, A.length$.
- Finalização: Quando $i = A.length$, temos como resultado uma lista $A[1, \ldots, i, \ldots, A.length]$ ordenada.

D What is the worst-case running time of BUBBLESORT? How does it compare to the running time of insertion sort?

Resposta:

A tabela abaixo exibe o número de comparações feitas pelo Bubble Sort para o j-ésimo elemento, com $j = 1, \ldots, n$, onde n é o tamanho do vetor.

Elemento	**# de Comparações**
1	$(n-1)$
2	$(n-2)$
$\vdots$	$\vdots$
j	$(n-j)$
$\vdots$	$\vdots$
$(n-1)$	1

Assim, a complexidade do Bubble Sort é dada por:

$$(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2} = \frac{n^2 - n)}{2}, \text{ i.e., } O(n^2)$$

Que é exatamente a complexidade do algoritmo Insertion Sort no pior cenário (quando o vetor está ordenado em ordem decrescente).
No melhor caso, i.e., quando os elementos do vetor já estão em ordem crescente, a complexidade de tempo do Insertion Sort é $O(n)$, enquanto que o Bubble Sort continua tendo complexidade $O(n)$.

OBS: Dizemos que a complexidade do Bubble Sort é $\Theta(n)$.

3 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.

Definições:

(a) $f(n) = O(g(n))$ se $\exists\ N \in \mathbb{N}$, e $\exists\ c \in \mathbb{R}_+$ tal que, $\forall\ n \in \mathrm{N}$, se $n \geq N$, então $f(n) \leq c \cdot g(n)$.
(b) $f(n) = \Omega(g(n))$ se $\exists\ N \in \mathbb{N}$ e $\exists\ c \in \mathbb{R}_+$ tal que, $\forall\ n \in \mathbb{N}$, se $n \geq N$, então $f(n) \geq c \cdot g(n)$.
(c) $f(n) = \Theta(g(n))$ se (a) e (b) são satisfeitas, isto é, $f(n) = O(n)$ e $f(n) = \Omega(g(n))$

- $f(n) = \log n^2$; $g(n) = \log n + 5$
  Resposta:
  Vamos verificar quais definições, (a), (b) ou (c) são satisfeitas.
  (a) $f(n) = O(g(n))$. Para ver isso, tome $N = 1$ e $c = 3$. Assim, $f(n) = 2\log n \leq 3\log n + 15 = cg(n)$.
  (b) Faça $c = 1$ e $N = 32$, temos $f(n) = 2\log n \geq \log n + 5 = cg(n)$. Assim, é fácil ver que $f(n) = \Omega(g(n))$.
  (c) Note que, de (a) e (b), temos que f(n) = O(n) e f(n) = $\Omega(g(n))$. Logo, $f(n) = \Theta(g(n))$.
- $f(n) = \log^2 n$; $g(n) = \log n$
  Resposta:
  (a) $\nexists\ c \in \mathbb{R}_+,\ N \in \mathbb{N}$ tais que $f(n) \leq c \cdot g(n)$, se $n \geq N$. Logo, $f(n) \neq O(g(n))$.
  (b) Faça $c = 1$ e $N = 4$. Assim, temos $f(n) = \log n \cdot \log n = 4 \geq 2 = \log n = c \cdot g(n)$. Logo, $f(n) = \Omega(g(n))$.

(c) Como a definição (a) não é satisfeita, temos apenas que $f(n) = \Omega(g(n))$.

- $f(n) = n \log n + n$; $g(n) = \log n$

  Resposta:

  (a) $\nexists\ c \in \mathbb{R}_+,\ N \in \mathbb{N}$ tais que $f(n) \leq c \cdot g(n)$, se $n \geq N$. Logo, $f(n) \neq O(g(n))$.
  (b) Escolha $c = 1$ e $N = 2$. Assim, $f(n) = n \cdot \log n \cdot \log n = 4 \geq 1 = 1 \cdot \log n = c \cdot g(n)$. Portanto, $f(n) = \Omega(g(n))$.
  (c) Como apenas a definição (a) é satisfeita, podemos afirmar apenas que $f(n) = \Omega(g(n))$.

- $f(n) = 2^n$; $g(n) = 10n^2$

  Resposta:

  (a) $\nexists\ c \in \mathbb{R}_+,\ N \in \mathbb{N}$ tais que $f(n) \leq c \cdot g(n)$, se $n \geq N$. Logo, $f(n) \neq O(g(n))$.
  (b) Escolhendo convenientemente $c = \frac{1}{10}$ e $N = 5$, temos: $f(n) = 2^n = 32 \geq 25 = n^2 = cg(n)$. Assim, $f(n) = \Omega(g(n))$.
  (c) Somente a definição (a) é satisfeita. Logo, temos que $f(n) = \Omega(g(n))$.

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

Resposta:

(a) Tome $c = 6$ e $N = 1$. Assim, $n^3 - 3n^2 - n + 1 \leq n^3 + 3n^3 + n^3 + n^3 = 6n^3$. Logo, $f(n) = O(g(n))$.
(b) Escolha $c = \frac{1}{10}$ e $N = 5$. Assim, $5^3 - 3 \cdot 5^2 - 5 + 1 = 46 \geq \frac{5^3}{10} = 12,5$. Portanto, $f(n) = \Omega(g(n))$.
(c) Como as definições (a) e (b) são satisfeitas, afirmamos que $f(n) = \Theta(g(n))$.

■

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

Resposta:

Ora, $n^2 < 2^n,\ \forall\ n > 4$. Além disso, não existem $c \in \mathbb{R}_+$ e $N \in \mathbb{N}$ tais que, se $n \geq N$, $n^2 > c \cdot 2^n$, isto é, $n^2 \neq \Omega(2^n)$

Portanto, $n^2 = O(2^n)$. ■

4 Insertion Sort - Mergesort - Quicksort [3pts]

Implement the insertion sort, merge sort and quicksort using the template `test.py` (use Python 3.X). Create a `test.cpp` file and write the equivalent code from `test.py` in C++, ie., the functions: main, `insertion_sort`, `merge_sort`, `quicksort` and `is_sorted`. For the random number generations you can use the `rand` function from `cstdlib`[2]. Your code should print the tuple (number of objects, time insertion_sort, time merge_sort, time quicksort)
You must submit both `test.py` and `test.cpp`. Graphs and descriptions must be included in this document.

4.1 Random Order

1. Create 10 sets of numbers in random order. The sets must have {10k, 20k, 30k, ..., 100k} numbers.
2. Sort these numbers using the 3 algorithms and calculate the time each algorithm takes for each set of numbers.

[2] `http://www.cplusplus.com/reference/cstdlib/rand/`

3. Generate a plot (using excel or another tool) showing a *linechart*, where the x-axis is the "number of elements", and the y-axis is the time that the algorithms took in C++ and Python. This plot must have 6 lines of different colors with a legend.

4. Write a small paragraph (3 to 4 lines) describing the results.

   Resposta:

   A figura abaixo exibe os tempos de execução de cada algoritmo (Insertion Sort, Merge Sort e Quicksort), no caso em que os vetores têm elementos aleatórios.

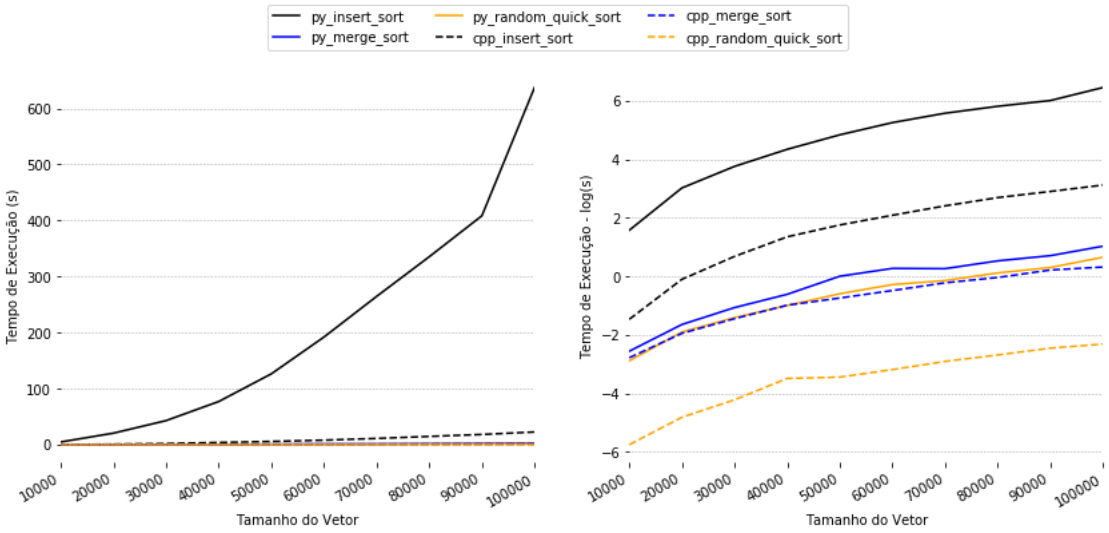

Figura 2: Tempo de execução com vetores em ordem aleatória

Nesse gráfico, podemos destacar dois pontos:

1. O Insertion Sort foi o algoritmo mais ineficiente dos três testados (o que apresentou o maior tempo de execução). Isso já era esperado dado que a complexidade desse algoritmo é dada por $O(n^2)$, no pior caso, enquanto que o Merge Sort $O(n \cdot logn)$ $O(n^2)$. Já o Quicksort, devido à sua natureza aleatória, tem complexidade $C(n)$ tal que $O(n \cdot logn) \leq C(n) \leq O(n^2)$. Assim, à medida que aumentamos o tamanho do vetor, é esperado que o tempo gasto pelo Insertion Sort aumente na ordem de n^2, isto é, mais rápido que para os demais algoritmos.

2. Para todos os três algoritmos testados, o tempo de execução apresentado pelo C++ foi inferior ao tempo de execução do Python, o que reflete o fato de C++ ser uma linguagem compilada, enquanto o Python é uma linguagem interpretada (executada linha a linha).

4.2 Ascending Order

Do the same experiment when the numbers are ordered in ascending order.

Resposta:
A figura abaixo exibe os tempos de execução de cada algoritmo, no caso em que os vetores estão ordenados de maneira crescente (o que seria o melhor dos casos).
Nesse gráfico vemos que o algoritmo Merge Sort foi o pior dos três (gastou mais tempo em execução), uma vez que sua complexidade é dada por $O(n \cdot logn)$. Assim, no caso em que os vetores já estão ordenados, o Insetion Sort e o Quick Sort são mais rápidos, dado que suas complexidades são $O(n)$ e $C(n)$, com $O(n) \leq C(n) \leq O(n \cdot logn)$.

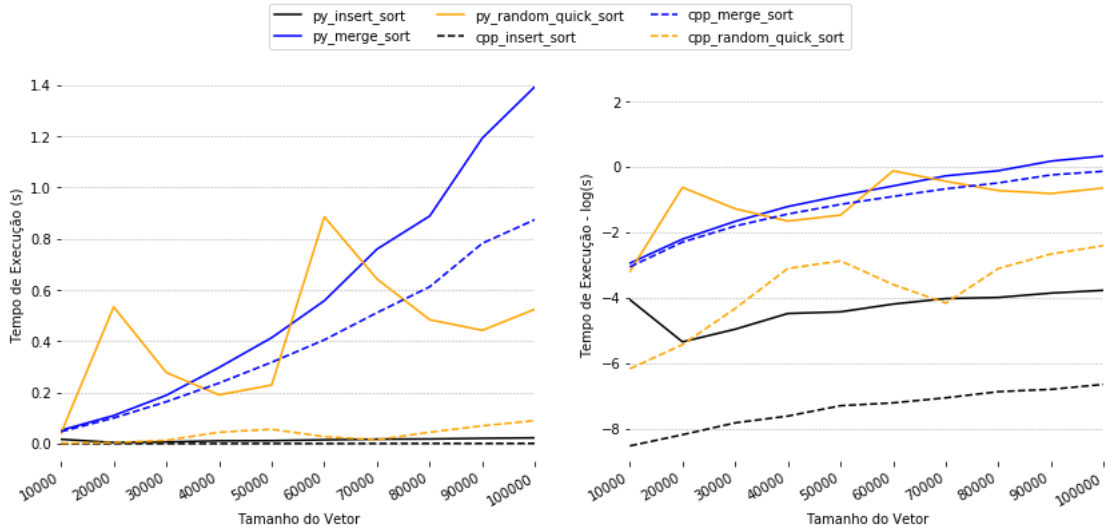

Figura 3: Tempo de execução com vetores em ordem crescente (melhor caso)

4.3 Descending Order

Do the same experiment when the numbers are ordered in descending order.

Resposta:
A figura abaixo exibe os tempos de execução de cada algoritmo, no caso em que os vetores estão ordenados de maneira decrescente (o que configura o pior dos casos).

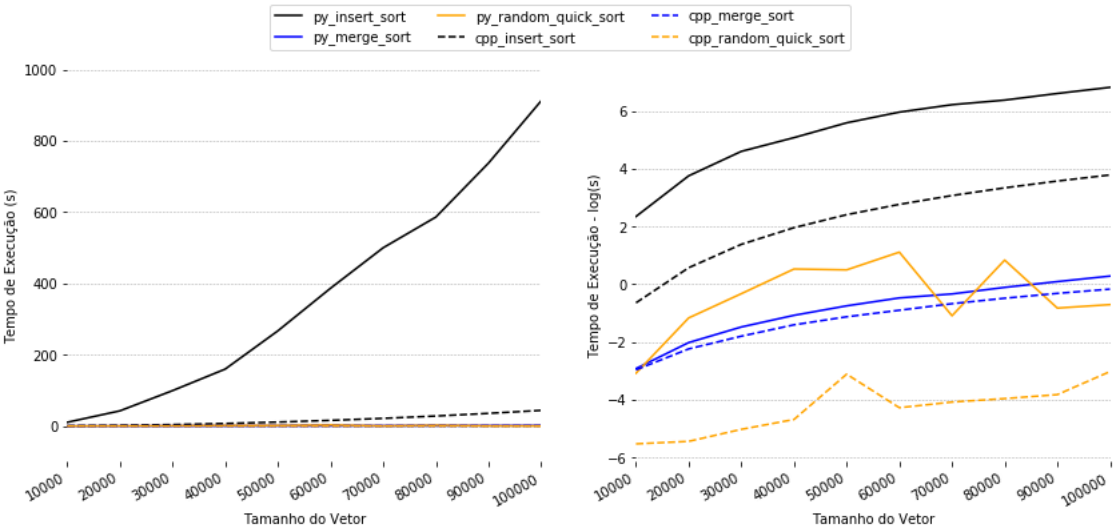

Figura 4: Tempo de execução com vetores em ordem decrescente (pior caso)

No gráfico acima, vemos que, no pior caso, o tempo de execução do Insertion Sort aumenta na ordem $O(n^2)$, o que é bem superior ao tempo gasto pelos outros dois algoritmos, que crescem (no máximo), na ordem $O(n \cdot logn)$