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

Homework 1
Due:: March 13, 2020

Homework 1

1 Induction [3pts]

Answers should be written in this document.

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

   - Primeiramente, vamos fazer para o caso onde n = 1. Note que $\sum_{i=1}^{1} i^2 = 1^2 = \frac{1(1+1)(2*1+1)}{6} = \frac{1(2)(3)}{6} = 1$. Vamos supor que essa propriedade é verificada para um certo k, assim: $\sum_{i=1}^{k} i^2 = \frac{k(k+1)(2k+1)}{6}$. Somaremos $(k+1)^2$ em ambos os lados da igualdade, ficando com: $\sum_{i=1}^{k+1} i^2 = \frac{k(k+1)(2k+1)}{6} + (k+1)^2 = \frac{k(k+1)(2k+1)+6(k+1)^2}{6}$, colocando $k+1$ em evidencia ficamos com: $\sum_{i=1}^{k+1} i^2 = \frac{(k+1)(k(2k+1)+6(k+1))}{6}$. Note tambem que $k(2k+1)+6(k+1) = 2k^2+7k+6 = (2k^2+3k)+(4k+6) = k(2k+3)+2(2k+3) = (k+2)(2k+3)$. Logo: $\sum_{i=1}^{k+1} i^2 = \frac{k(k+1)(2k+1)}{6} + (k+1)^2 = \frac{(k+1)(k+2)(2k+3)}{6} = \frac{(k+1)[((k+1)+1)(2(k+1)+1))}{6}$.

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

   - Vamos verificar nosso caso base: $3^7 < 7!$, vejamos que $3^7 = 2187$ e $7! = 5040$, logo de fato a desigualdade segue verdadeira. Suponha que para um dado $k > 7$, $3^k < k!$. Queremos provar que isso implica a validez para k+1. Para isso, multiplique por 3 em ambos os lados ficando com $3^{k+1} < 3k!$ o que implica que $3^{k+1} < (k+1)k!$, pois $3k! < (k+1)k!$, logo $3^{k+1} < (k+1)!$.

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

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

   - Vamos usar a definição de teto. Para n = 0, $\lceil \frac{0}{2} \rceil = \lceil 0 \rceil = 0$. Para um certo n par, suponhamos que $\lceil \frac{n}{2} \rceil = \frac{n}{2}$. Para o próximo par $n+2$ teremos que $\lceil \frac{n+2}{2} \rceil = \lceil \frac{n}{2} + 1 \rceil = \lceil \frac{n}{2} \rceil + 1 = \frac{n}{2} + 1$ por hipótese. Se n é ímpar, digamos 1, $\lceil \frac{1}{2} \rceil = \lceil 0.5 \rceil = 1$. Suponhamos que para um dado n ímpar maior que 1, $\lceil \frac{n}{2} \rceil = \frac{n+1}{2}$. Para o próximo ímpar n+2, teremos que: $\lceil \frac{n+2}{2} \rceil = \lceil \frac{n}{2} + 1 \rceil = \lceil \frac{n}{2} \rceil + 1 = \frac{n+1}{2} + 1 = \frac{(n+2)+1}{2}$ por definição.

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

   - Note que para x = 1,2,3,...,10, temos que $x \equiv s(x)(mod 3)$, onde s(n) retorna a soma dos digitos de n. Suponha que $\forall k < n$, $n \equiv s(n)(mod 3)$. Ao somar 1 a k, temos duas possibilidades, ou o ultimo digito dele nao é 9, nesse caso s(k+1) = s(k)+1, que prova o problema, ou seu ultimo digito é 9, dai segue q ele pode ser escrito como $k+1 = 10*t$e assim $s(k+1) = s(t)$, e t¡n por hipotese, temos que $s(t) \equiv t(mod 3)$

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

   - Vamos usar indução forte em $11x + 7y = n$. Veja que para os casos onde $59 < n < 67$ teremos:

   $$11x + 7y = 60, \text{ tem-se que } x = 1; y = 7$$
   $$11x + 7y = 61, \text{ tem-se que } x = 3; y = 4$$
   $$11x + 7y = 62, \text{ tem-se que } x = 3; y = 4$$
   $$11x + 7y = 63, \text{ tem-se que } x = 0; y = 9$$
   $$11x + 7y = 64, \text{ tem-se que } x = 2; y = 6$$
   $$11x + 7y = 65, \text{ tem-se que } x = 4; y = 3$$
   $$11x + 7y = 66, \text{ tem-se que } x = 6; y = 0$$

   Veja que por exemplo, para $n = 67$, basta somarmos 7 a equação 1 e aumentar em uma unidade o termo y ficando com $11x + 7y = 67$, $(x = 1; y = 8)$, de modo análogo, até n , podemos somar k 7's a alguma das 7 equações acima e obter nossa solução com y aumentado em k unidades, por exemplo $11a + 7(b + k) = n$ onde $11a + 7b = t$ é alguma das equações listadas acima. Para $11a' + 7b' = n + 1$ basta somar k vezes o 7 a equação de forma $11a'' + 7b'' = t + 1$ que tem solução por hipótese, pois $59 < t + 1 < n$.

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

   - Vamos primeiramente fazer indução em k. Para k = 1, teremos:
   $F_{n+1} = F_1 F_{n+1} + F_0 F_n$, onde $F_1 = 1$ e $F_0 = 0$
   portanto de fato a propriedade é válida nesse caso. Suponha que ela é válida para todos os números menores que k , inclusive ele, logo por hipótese $F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$. Sabemos que $F_{n+(k+1)} = F_{n+k} + F_{n+k-1}$, e por hipótese $F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$ e $F_{n+k-1} = F_{k-1} F_{n+1} + F_{k-2} F_n$. Substituindo na equação teremos $F_{n+(k+1)} = F_k F_{n+1} + F_{k-1} F_n + F_{k-1} F_{n+1} + F_{k-2} F_n = F_n(+F_{k-2} + F_{k-1}) + F_{n+1}(F_k + F_{k-1}) = F_n F_k + F_{n+1} F_{k+1}$. Agora basta provar que a validez da propriedade para n implica a validez para $n + 1$. Suponhamos que até um dado n , segue que $F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$ e tambem temos que $F_{n+1+k} = F_{n+k} + F_{n+k-1}$ agora vamos usar a hipótese de que a propriedade é valida $\forall n_0 < n + 1$. Disso, segue que:

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

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

   - Vejamos que para $n = 1$, $\binom{1}{0} = 2^0 = 1$. Agora, suponhamos que $\forall t \leq n$, $\sum_{t=0}^{n} \binom{n}{t} = 2^t$. Sabemos tambem pela relação de stifel que $\binom{n}{m} + \binom{n}{m+1} = \binom{n+1}{m+1}$, e por hipótese $1 + \sum_{t=0}^{n} \sum_{m=0}^{t} = 1 + \sum_{t=0}^{n} 2^a = 2^{n+1}$, ou seja, $1 + \sum_{t=0}^{n} \sum_{m=0}^{t} = \binom{n+1}{0} + \sum_{m=1}^{n+1} \binom{n+1}{m}$, o que conclui a prova.

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

   - Para o caso base n = 1, temos no máximo 0 arestas, uma vez que temos apenas um nó. Suponha que dado n, temos $\frac{n(n-1)}{2}$ arestas. Ao adicionar 1 aresta, ou seja, com n+1 arestas, teremos agora $\frac{n(n-1)}{2} + n$ arestas, pois a novo nó terá n possibilidades para se conectar e formar uma aresta, assim:

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

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

   - Para n = 1, primeiro nivel , a arvore de fato tem $2^1 - 1$ nós.Veja que a cada nivel, a quatidade de nós é uma potencia de dois , uma vez que sucessivamente cada vertice da a luz a outros dois.Assim no nivel k , temos 2^{k-1} vertices em baixo. Suponha que dado nivel n, temos $2^n - 1$ vertices e ja sabemos tambem que nesse caso temos 2^{n-1} vertices na base. No próximo nivel, esses vertices da base geraram 2^n novos vertices uma vez que eles se multiplicaram. Assim no total teremos: $2(2^n) + 2^n - 1 = 2^{n+1} - 1$ vertices no nivel n+1.

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 $180(n - 2)$.

    - Para um poligono com tres vértices é trivial que a soma de seus angulos internos seja 180(3-2) = 180. Suponha que em dado poligno convexo de n vértices, teremos uma soma de angulos internos equivalente a 180(n-2). Agora imagine o poligono fechado, desenhamos um ponto fora dele no ponto médio de alguma aresta. Se ligarmos esse novo ponto as duas pontas dessa aresta formaremos um triangulo, cuja soma dos angulos é 180 graus.Agora note que basta tirarmos a aresta usada como referencia anterioremente de modo que nao faça diferença nos angulos. Agora temos um poligono convexo com n+1 lados cuja soma dos seus angulos internos é 180(n-2)+180=180((n+1)-2).

2 Correctness of bubblesort [3pts]

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 **(0.5pt)** Let A' denote the output of BUBBLESORT(A). To prove that BUBBLESORT is correct, we need to prove that it terminates and that

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

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

- Precisamos encontrar uma invariante e provar que ela sempre é verdade, somente nesse caso teremos exatamente certeza que o algoritimo funciona e faz o que devia estar fazendo.

The next two parts will prove inequality (1).

B **(1.0pt)** 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.

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

- Lema: Sempre levamos o maior elemento para a ultima posição.
  Prova : Para n = 1 é trivial que o for rodado uma vez leva o 1, o maior elemento , para sua posição correta. Suponha que para n = k, esse for leve k para ultima posição. Se n = k+1, quando chegarmos em $A[j] = k+1$, o if sempre será verdadeiro se tiver um número ao lado direito de $k + 1$, assim no final ele estará na ultima posição.

C **(1.0pt)** 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 (1). Your proof should use the structure of the loop invariant proof presented in this chapter.

- Lema: Quando esse for terminar de rodar pela k-ésima vez ,$A[len(A) - k : len(A) - 1]$ estará na posição correta e em ordem.
  Prova: Vejamos que para n = 1, após a primeira iteração desse for o elemento estará em sua posição correta. Suponhamos que para um certo n , existe $k < n$ tal que na k-ésima iteração do for, $A[len(A) - k : len(A) - 1]$ está ordenado e nas posições corretas. Assim, esses números não saem do lugar nas proximas iterações pois ja estão em ordem crescente. Sabemos tambem que o elemento k-1 está em algum lugar atrás desse novo bloco ordenado que obtemos. Então na iteração k+1, quando chegarmos em A[j] = k-1, ele será levado até exatamente o lado esquerdo de k, pois ele é o maior elemento da lista tirando aqueles ja ordenado por hipótese e o if $A[j] < A[j - 1]$ sempre resultará em verdadeiro. Logo , na k+1 iteração, teremos k+1 elementos ordenados, mais especificamente $A[len(A) - k : len(A) - 1]$. (nota: veja que k-1 nao passa de k , pois o if resulta em falso em qualquer iteração seja qual ela for)

D **(0.5pt)** What is the worst-case running time of BUBBLESORT? How does it compare to the running time of insertion sort?

- Quando temos a lista em ordem inversa, ou seja, decrescente. Assim teremos uma complexidade de n^2 de execução. Ambos tem as mesmas complexidades nos piores e melhores casos.

3 Insertion Sort - Mergesort [4pts]

Implement the insertion sort and merge sort 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`, 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)
You must submit both `test.py` and `test.cpp`. Graphs and descriptions must be included in this document.

3.1 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 2 algorithms and calculate the time each algorithm takes for each set of numbers.
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.

3.2 Ascending Order

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

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

3.3 Descending Order

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

- Podemos notar pelos gráficos que o crescimento de tempo do insert sort para uma array aleátoria cresce quase de forma linear, veja:

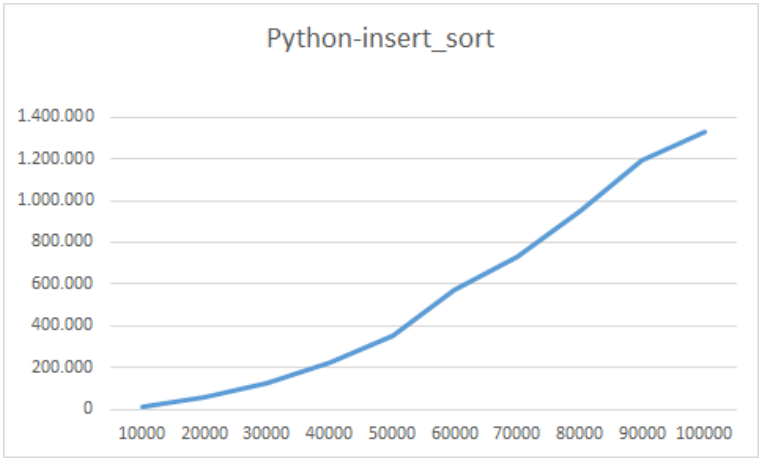

Figure 1: array aleatoria

enquanto para o merge sort, teremos um grafico do tipo: Com arrayz crescentes teremos os seguintes gráficos:

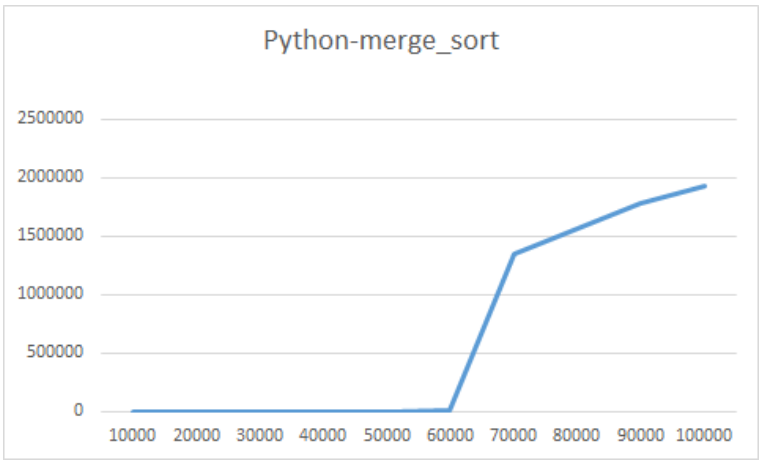

Figure 2: array aleatoria

E os dois ultimos testes com listas decrecentes:
Podemos concluir que:

- insert sort se sai melhor com arrays randomicas a longo prazo.
- Com arrays crescentes, o inser sort tambem leva a mellhor uma vez que nao precisa ordenar nada, ja o merge faz todo o processo de divisão e tudo até o fim mesmo estando ordenada a array.

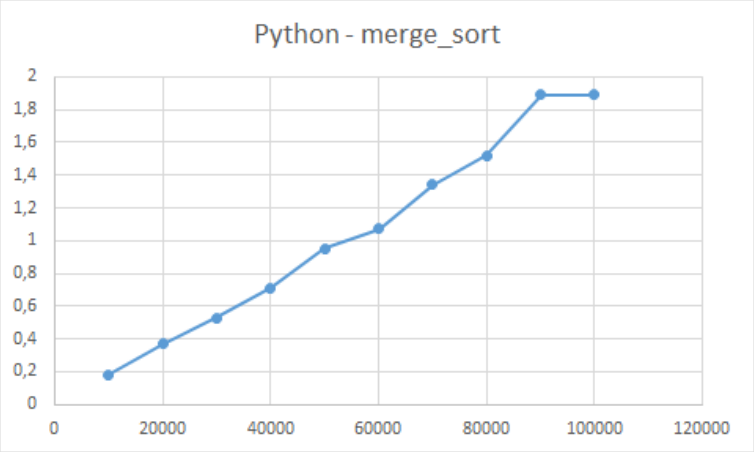

Figure 3: array crescente

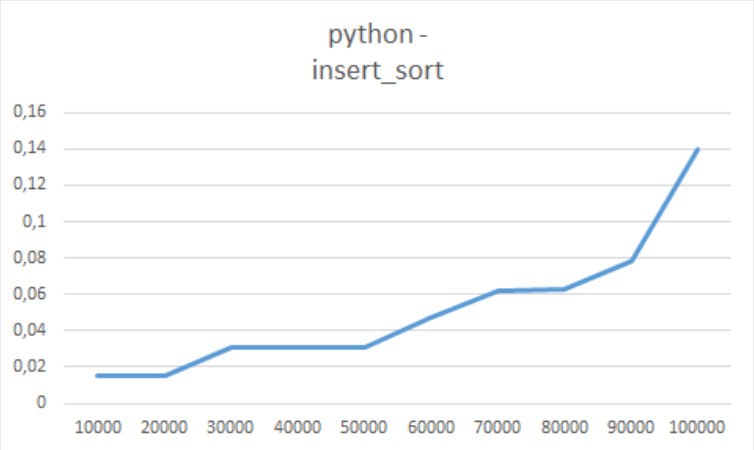

Figure 4: array crescente

- Com arrays decrescentes, o merge sort se sai melhor uma vez que estamos considerando o pior caso do insert sort.

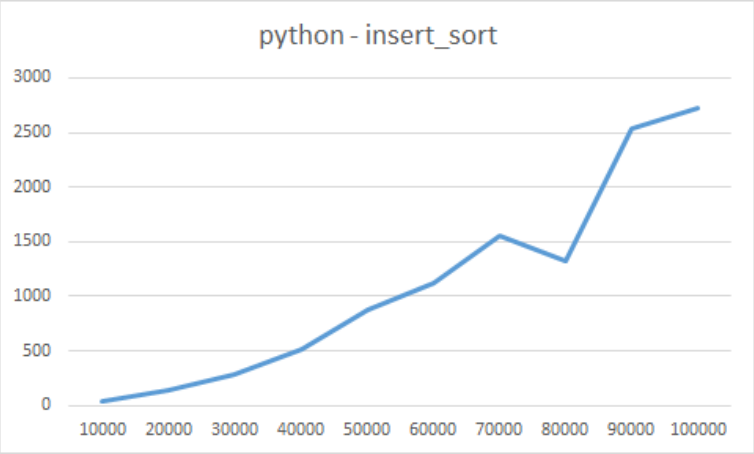

Figure 5: array decrescente

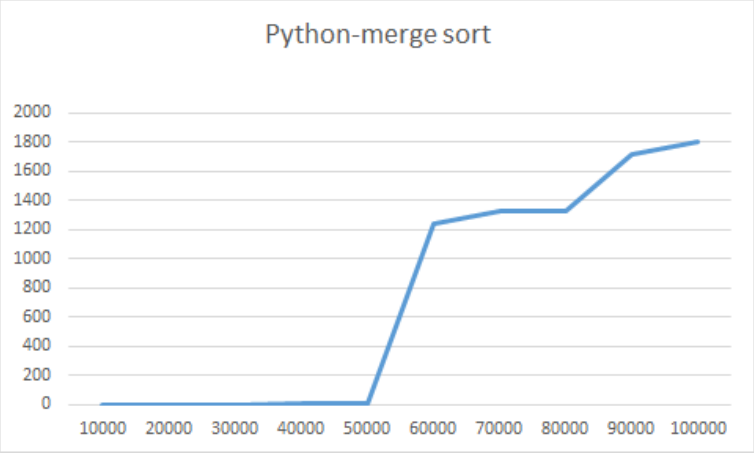

Figure 6: array decrescente