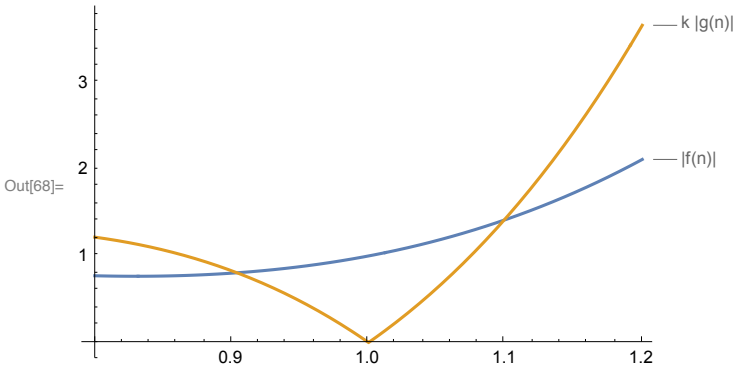
Aula 4
Análise da Complexidade

Algoritmos de ordenação

Def. Uma função f(n)∈O(g(n)) se $\exists_{k>0}\ \exists_{n_0>0}\ \forall_{n>n_0}\ |f(n)| < k\,|g(n)|$



Nota: Recorde que me Java todas operações aritméticas e booleanas são de tempo O(1), pois os tipos são limitados. O mesmo não acontece em Mathematica onde os inteiros são arbitrariamente longos.

Exercício

a) Mostre que se f(n)∈O(g(n)) sse $\limsup_{n\to\infty} \frac{|f(n)|}{|g(n)|} < \infty$.

R:

⇒) $\limsup_{n\to\infty} \frac{|f(n)|}{|g(n)|} = \lim_{n\to\infty} \left(\sup_{m>n} \left\{ \frac{|f(m)|}{|g(m)|} \right\}\right) \leq$

$\sup_{m>n_0} \left\{ \frac{|f(m)|}{|g(m)|} \right\} \leq \sup_{m>n_0} \left\{ \frac{k\,|g(m)|}{|g(m)|} \right\} = k < \infty$

⇐) $\lim_{n\to\infty} \left(\sup_{m>n} \left\{ \frac{|f(m)|}{|g(m)|} \right\}\right) = k'$ logo qq $\varepsilon>0$, existe n_0 tal que qq $n > n_0$ temos $\frac{|f(m)|}{|g(m)|} < k'+\varepsilon$

escolha-se um ε ao nosso gosto, existe $k=k'+\varepsilon$ e

existe n_0 tal que qq $n > n_0$ $|f(m)| < (k'+\varepsilon)\,|g(m)|$!

b) Mostre que se p(n) é um polinómio de grau k então p(n) ∈ $O(n^k)$.

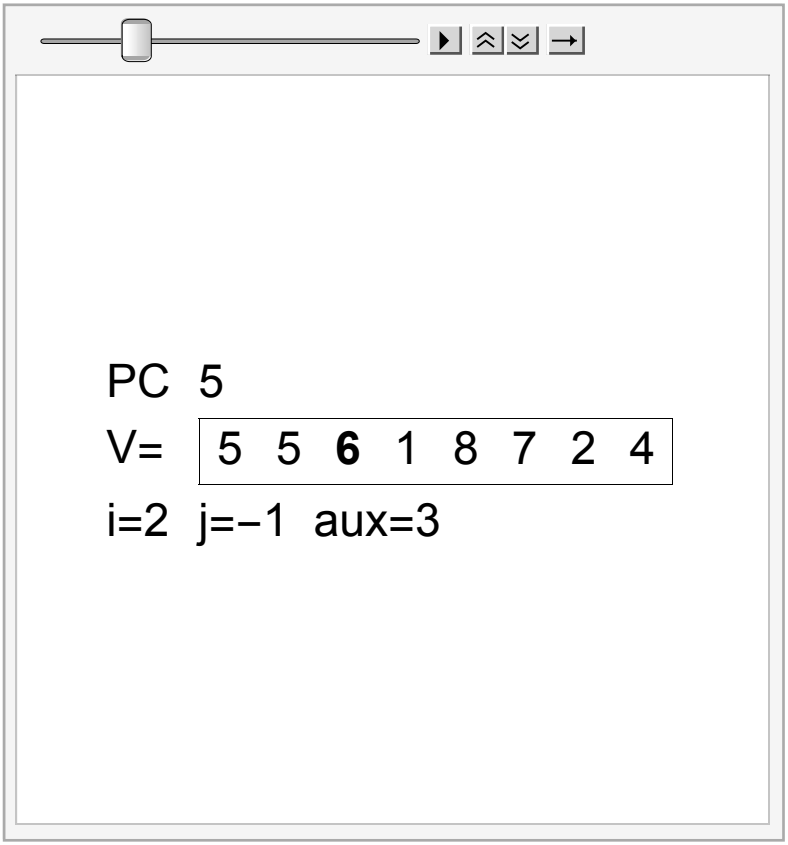
R:

$p(n)=a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$

Usando o exercício anterior basta verificar
$\lim_{n\to\infty} \frac{p(n)}{n^k} = a_k < \infty$!

Insertion Sort

O InsertionSort é um dos algoritmos de ordenação mais simples.



```
public static void insertionSort (int V[]) {
1 int i = 1, j, aux, n = V.length;
2 while (i < n) {
3   aux = V[i];
4   j = i - 1;
5   while (j >= 0 && V[j] > aux) {
6       V[j + 1] = V[j];
7       j--;}
8   V[j + 1] = aux;
9   i++;}
10 }
```

Análise da Complexidade do InsertionSort

Pior caso:

Melhor caso:

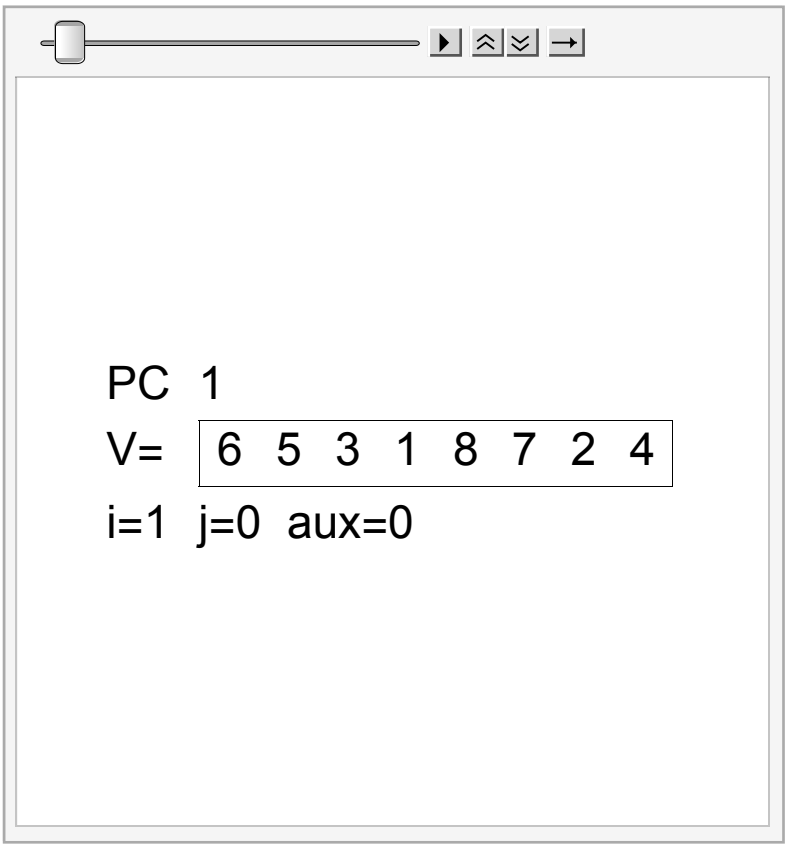
Caso médio:
Assuma que no segundo ciclo poderá terminar com j=-1,0,….,i-1 equiprovavelmente.

O objectivo é calcular a complexidade média

Exercício: Analise a complexidade do Insertionsort no caso em que apenas o último elemento está desordenado, e que pode ser colocado equiprovalvemente em qualquer posição.

Bubble Sort

Outro algoritmo muito simples de ordenação é o BubbleSort




```
public static void BubbleSort (int V[])){
1  int i=1, j, aux, n = V.length;
2  while (i < n) {
3  j = 0;
4  while (j < n-i) {
5   if (V[j+1] < V[j]) {
6       aux = V[j];
7       V[j] = V[j+1];
8       V[j+1] = aux;}
9     j++;}
10 i++;}}
```

Exercício: Encontre os invariantes relevantes para o Bubblesort.

Análise da Complexidade do BubbleSort

Pior caso:

Melhor caso:

Caso médio:

Exercício: Modifique o BubbleSort para que seja linear no melhor caso.

```
public static void BubbleSortM (int V[])){
  int i=1, j, aux, n = V.length;
  while (i < n) {
  j = 0;
  while (j < n-i) {
    if (V[j+1] < V[j]) {
        aux = V[j];
        V[j] = V[j+1];
        V[j+1] = aux;}
     j++;}
  i++;}}
```

Quick Sort

O QuickSort é um algoritmo muito eficiente que se deve a Hoare. O algoritmo é recursivo e baseia-se num método de partição.

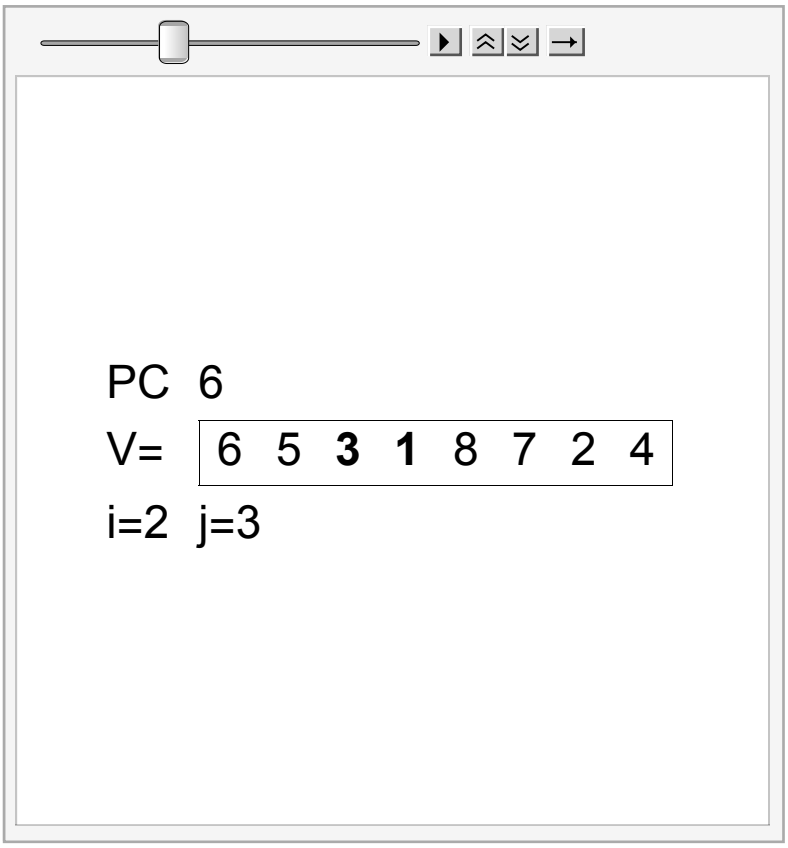
```
public static void swap(int vec[], int a, int b) {//troca elem da posição a com
posição b - O(1)
  int tmp = vec[a];
  vec[a] = vec[b];
  vec[b] = tmp;
}

public static int partition(int vec[], int left, int right) {
1  int i=left, j=left + 1;      //vec[left] é chamado de pivot
2  while(j<=right) {            //j varre os elementos de vec[left+1,...,right]
3    if (vec[j]<=vec[left]){    //i é posição do último elemento menor ou igual a
4      i++;                     //vec[left] que foi encontrado, tendo o cuidado de
5      swap(vec,i,j);}          //deixar todos os elementos em posições < i
6    j++;}                      //menores ou iguais a vec[left]
7  swap(vec,left, i);           //o valor retornado é o valor onde fica o pivot
8  return i;
}

public static void quickSort(int vec[], int left, int right) { //sobreposição em Java
  int r;
  if (right > left) {
    r = partition(vec,left,right);
    quickSort(vec, left, r - 1);
    quickSort(vec, r + 1, right);
  }
}

public static void quickSort(int vec[]) {
    quickSort(vec, 0, vec.length-1);
}
```

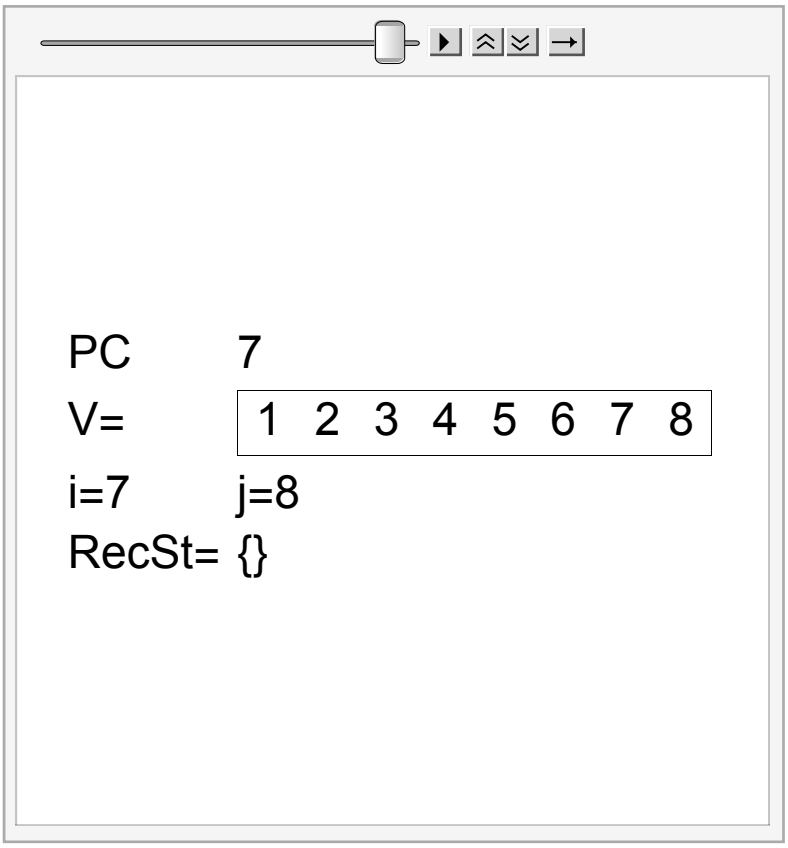
Comecemos por examinar o método de partição.



O método particiona o vetor em duas partes, a parte com elementos menores ou iguais ao pivot (vec[left]) e a parte como elementos superiores. A nova posição do pivot é o valor retornado pela função. Esta posição é depois usada para continuar a ordenar o vector.

Assim facilmente se obtém o Quicksort:

Out[67]=

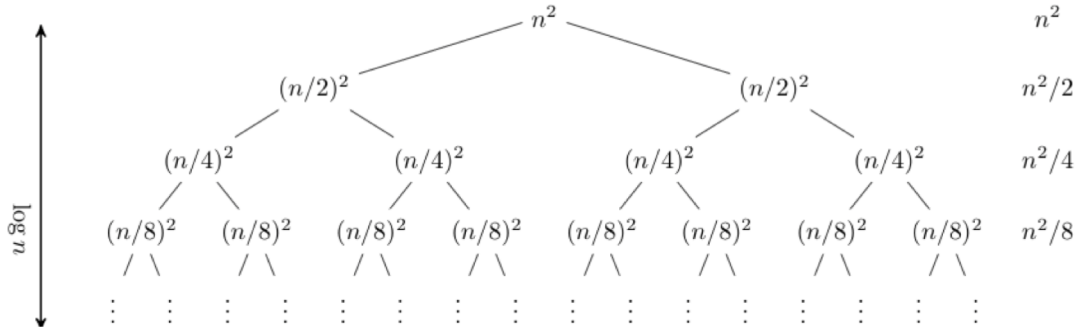


A demonstração da correção do QuickSort é feita por indução, assumindo a correção do partition.

Teo: O Algoritmo QuickSort ordena o vector por ordem crescente.

Vamos agora estudar as ferramentas para analisar a complexidade de algoritmos recursivos.

Considere a por exemplo a recursão $T(n) = 2\,T(n/2) + n^2$



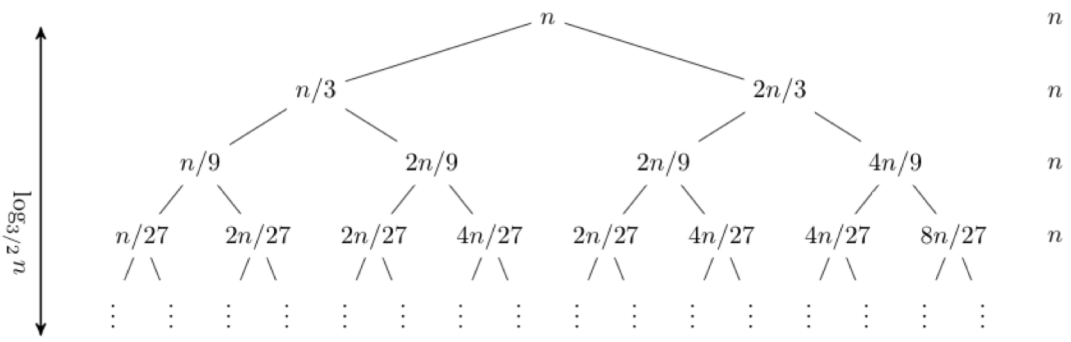
Da análise da árvore concluímos que

$T(n)=\sum_{i=1}^{\log(n)} n^2/2^i$

In[73]:=
```
Sum[n^2/2^i, {i, 0, Log[n]}]
```

In[74]:=
```
Expand[%]
```

Considere agora $T(n) = T(n/3) + T(2\,n/3) + n$



T(n)=$n \sum_{i=1}^{\log_{3/2}(n)} O(1)$

Teorema Mestre Sejam a≥1 e b>1 constantes, f(n) uma função e T(n) definida sobre os números naturais tal que

T(n) =a T(n/b)+f(n)

onde n/b pode ser ⌊n/b⌋ ou ⌈n/b⌉. Então T(n) ∈ $O\left(n^{\log_b a}\right)$ desde que f(n) ∈ $O\left(n^{\log_b a-\varepsilon}\right)$ para algum $\varepsilon > 0$.

Prova...