Ordenação

Definição: É a operação de reorganizar os dados em uma determinada ordem de acordo com um valor chave.

.

Problema da ordenação -

Entrada: Uma sequência de n valores (a_1, a_2, . . . , a_n).

Saída: Uma permutação (reordenada) (a'_1 , a'2~ , . . . , a'_n) da sequência de entrada tal que $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Do ponto da memória do computador, os algoritmos de ordenação podem ser classificados em:

Ordenação Interna: quando os dados a serem ordenados estão na memória principal.

Ordenação Externa: quando os dados a serem ordenados necessitam de armazenamento em memória auxiliar como por exemplo o disco HD.

Ordenação Interna

Na escolha de um algoritmo de ordenação interna deve ser considerado principalmente:

- O tempo gasto pela ordenação;
- O uso econômico da memória disponível;

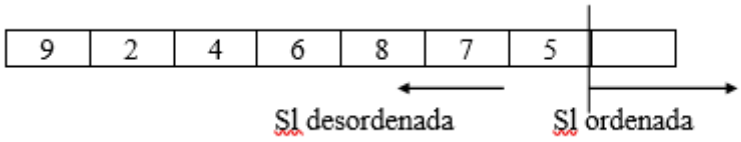
Um algoritmo de ordenação é estável (= *stable*) se não altera a posição relativa dos elementos que têm o mesmo valor.

Estratégia

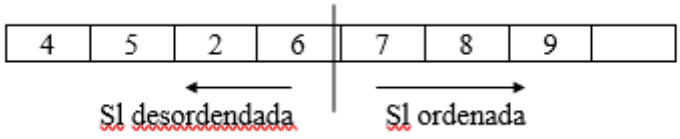
Pode-se sempre visualizar uma lista como a concatenação de sub-listas:

a sub-lista desordenada e a sub-lista ordenada.

Inicialmente, antes de aplicar-se um método de ordenação em uma lista desordenada, a sub-lista desordenada contém todos os elementos e a sub_lista ordenada está vazia, ou, com um elemento pois, uma lista com apenas um elemento está ordenada.



A cada iteração, a sub-lista ordenada ganha elementos e a sub-lista desordenada perde elementos, isto é, elementos da sub-lista desordenada são transferidos para a lista ordenada.



Exemplo: Após 3 iterações do método de ordenação por seleção do maior

Para implementar tal ideia, é necessário que os limites das sub-listas sejam marcadas:

- inicio, fim da sub-lista desordenada e

- inicio e fim da sub-lista ordenada.

Principais Categorias de Métodos

- *Classificação por Seleção*:
  - consiste em uma seleção sucessiva do menor/ maior valor contido na parte ainda desordenada dos elementos.
  - Exemplo: - Método da Seleção Direta (do maior ou do menor)
- *Classificação por Inserção*:
  - ordenação pela inserção de cada um dos elementos em sua posição correta, levando em consideração os elementos já ordenados
  - Exemplos: - Método da Inserção Direta,
    - Método dos Incrementos Decrescentes (Shellsort)
- *Classificação por Trocas*:
  - caracteriza-se pela comparação de **pares de chaves**, trocando-as de posição caso estejam fora de ordem.
  - Exemplos: - Método da Bolha (Bubblesort),
    - QuickSort

- *Classificação por Intercalação*:
  - intercalar partições classificadas por determinado critério, criando partições cada vez maiores, até unir todas as partições do conjunto original
  - Exemplos: - –Método da Intercalação Simples (MergeSort)

Método Ordenação por Inserção:

A ideia deste método é sucessivamente retirar um nó da sub_lista desordenada incluindo-o (na sua posição correta) na sub_lista ordenada. Portanto, devemos continuar o processo de retira da sub_lista desordenada e inclui na ordenada até que a sub_lista desordenada fique vazia. A escolha natural do nó da sub_lista desordenada a retirar, é o que ocupa a primeira posição, pois o mesmo é adjacente ao último nó da sub-lista ordenada e, a partir desta iteração, pertencerá à sub-lista ordenada. Inicialmente a sub-lista desordenada inicia na segunda posição, pois uma lista com um elemento é uma lista ordenada.

```
Enquanto há nós na sub_lista desordenada
        Se o 1º nó da sl desordenada está desordenado em relação ao último da
slordenada
            Encontra o lugar deste nó da sla ordenada, incluindo-o em tal lugar
            Desloca o início da sl  desordenada
fim_enquanto
```

Intercultural Computer Science Education

https://www.youtube.com/watch?v=ROalU379l3U

```
void InsertSort(TLISTA * pL)
{
    int inicdesord;
    TNO aux;
```

```
    for(inicdesord=1;inicdesord<pL->qtnos;inicdesord++){  /* a sl ordenada
possui o primeiro nó*/

        if (pL->elementos[inicdesord].chave>pL->elementos[inicdesord-1].chave){
            aux=pL->vnos[inicdesord];
            for(i=inicdesord-1;i>=0 && pL->vnos[o].chave > aux.chave; i--)
                pL->vnos[i+1]=pL->vnos[i];
                pL->vnos[i+1]=aux;
            }
}
```

Método Ordenação por Seleção:

A ideia deste método é escolher sucessivamente o maior (ou menor) valor e colocá-lo na última (primeira) posição da sub-lista ordenada. A cada iteração, um valor é colocado na posição correta. Portanto, devemos continuar o processo de escolher o maior entre os que sobraram até que não haja mais valores na parte desordenada da lista

```
Enquanto há elementos na sl desordenada
        Escolhe o maior valor da sl desordenada, marcando sua atual posição.
        Troca o conteúdo do últ.elemento da sl desordenada c/o da atual posição
do maior
        Diminui um elemento da parte desordenada
fim_enquanto
```

Intercultural Computer Science Education:
https://www.youtube.com/watch?v=Ns4TPTC8whw

```
// Ordenação por seleção do menor em lista sequencial
/*  Função para trocar conteúdo de dois nós  */
void troca (t_no *pa, t_no *pb){
    t_no    aux;
    aux = *pa;
    *pa = *pb;
    *pb = aux;
}

/*  Seleção do menor elemento de um determinado intervalo de uma lista:
a função seleciona_menor( ) recebe como parâmetros o vetor de elementos da lista,
a posição inicial e a posição final do intervalo a ser considerado, retornando a
posição do menor elemento nesse intervalo.  */

int seleciona_menor( t_no elem[ ], int ini, int fim){
    int indm;   /* guarda a posição do menor  */
    int j;
    if (ini> fim) /*  lista vazia  */
        return (-1);
    indm=ini;
    for (j=ini +1; j<= fim; j++)
```

```
        if (elem [j].chave < elem [indm].chave)
            indm = j;
    return (indm);
}

/*  Ordenação por seleção:  */
int ordena_seleção (t_lista *pl){
    int     ini, j, indm;
    for ( ini=0; ini < pl->num_elem -1; ini++)  {
indm=seleciona_menor(pl->elementos, ini, pl->num_elem -1);
        if (indm != ini)
            troca( &(pl->elementos[ini]), &(pl->elementos[indm]) );
    }
}
```

```
//Ordenação por seleção do menor em lista simplemente encadeada com percurso
recursivo na lista
void ColocaComoSuc (TNO **p, TNO **q){
    (*p)->prox = (*q)->prox;
    (*q)->prox = (*p);
}

TNO * ins_sort_R (TNO *p){
    TNO *q,*p;
    TNO *aux;
    if (p && p->prox) { // sl  tem  mais de um elemento
        q = ins_sort_R(p->prox);
        if (p->inf > q->inf) {  /* último da sl desordenada é maior que o
primeiro da sl ordenada
                                    encontra local do nó desclassificado */
         for (aux = q;
              (aux->prox != NULL) &&(p->inf > aux->prox->inf);
              aux = aux->prox);
        ColocaComoSuc(&p, &aux);
        p = q;
    }
    else
        p->prox = q;
    }
    return p;
}
void InsertSort (TLISTA *L){
    L->prim = ins_sort_R(L->prim);
}
```

```
//Ordenação por seleção do menor em lista duplamente encadeada circular com nó
cabeça
TNO* escolhe_menor(TNO*prim, TNO*ult){
    TNO* menor=prim;
    for (prim=prim->suc;prim!= ult; prim = prim->suc)
        if (prim->info.chv < menor->info.chv)
            menor = prim;
    return menor;
}
```

```c
void posic_apos(TNO *no1, TNO *no2){
    TNO *ant_no1, *suc_no2;
    /* desliga o no1 */
    no1->ant->suc = no1->suc;
    no1->suc->ant = no1->ant;
    /* liga o no1 como suc do no2 */
    no1->ant = no2->ant;
    no1->suc = no2;
    /* liga o no2 e seu ant ao no1 */
    no1->suc->ant = no1;
    no1->ant->suc = no1;
}

void Selection_Sort (TLISTAe *L){
    TNO *prim_desord;
    TNO *ult_ord=L->prim, *menor;
    for (ult_ord=L->prim;ult_ord->suc != L->prim;ult_ord=ult_ord->suc) {
        prim_desord= ult_ord->suc;
        menor=Escolhe_menor(prim_desord,L->prim);
        if (menor != prim_desord)
            posic_após(menor, ult_ord);
    }
}
```

Método Ordenação Bolha:

A ideia deste método é comparar sucessivamente o conteúdo de duas posições vizinhas da parte desordenada da lista. Se as mesmas não estiverem ordenadas, troca-se o conteúdo das posições vizinhas. Quando uma troca é realizada, entre o elemento i e o elemento i+1, o antigo vizinho do elemento i pode ter ficado desordenado em relação a este novo valor que ocupa a posição i. A cada iteração, diversos valores são colocados na posição correta. Em todo pedaço da sublista no qual não houve trocas, pode-se considerar em ordem.Portanto, devemos continuar o processo de comparação entre os vizinhos apenas na parte antecessora à última troca (que pode estar desordenada) até que não haja mais que um valor armazenado na parte desordenada:

```
Enquanto há mais que um elemento na parte desordenada
        Compara conteúdo de posições vizinhas, trocando-os se não estão em
ordem e marca onde ocorreu a última troca de conteúdo de posições
vizinhas
        A nova parte desordenada termina onde ocorreu a última troca
    fim_enquanto
```

Intercultural Computer Science Education
https://www.youtube.com/watch?v=lyZQPjUT5B4

```
void ordena_bolha(TLISTA *pl){
   int i, fim, ultroca;
   for(fim = pl->qtnos - 1; fim > 0; fim = ultroca)   {
      ultroca = 0;
      for(i = 0; i < fim; i++)      {
         if(pl->vnos[i].chave > pl->vnos[i+1].chave)  {
             troca( &(pl->elementos[i]), &(pl->elementos[i+1]) )
              ultroca = i;  // guarda onde realizou a última troca
         }
      }
    }
}
```

Observe que, se o vetor já estivesse ordenado, nenhuma troca seria realizada na primeira comparação entre vizinhos e o processo de ordenação terminaria, pois a parte desordenada ficaria com apenas uma posição (valor de ultroca).

O método de seleção só é mais eficaz que o da bolha quando o vetor está ordenado na ordem inversa à desejada.

Método Ordenação QuickSort:

Baseia-se em um padrão de projeto fundamental para solução de problemas conhecida como **Divisão e Conquista** (D*ivide-and-Conquer*).

O padrão pode ser descrito, de maneira geral, como sendo composto de 3 fases:

Divisão: divide-se os dados de entrada em dois ou mais conjuntos disjuntos (separados);

Recursão: soluciona-se os problemas associados aos subconjuntos recursivamente;

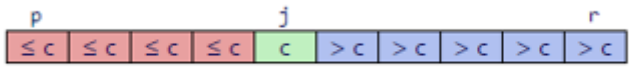
Conquista: obtém-se as soluções dos subproblemas e junta-se as mesmas em uma única solução.

Em ordenação:

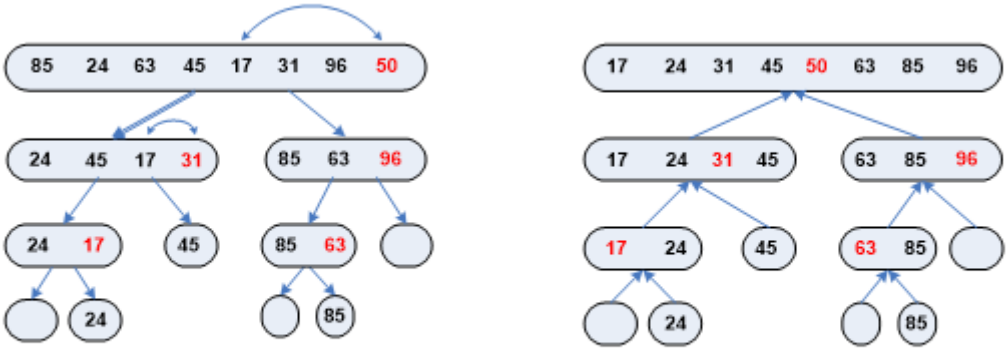
1. Dividir a entrada em conjuntos menores
2. Ordena cada instância menor de maneira recursiva
3. Reunir as soluções parciais para compor a solução do problema original

Características Gerais do Quick Sort

- Inicialmente, a lista de chaves *C* é particionado em três segmentos S1, S2 e S3.
- S2 deverá conter **apenas UMA** chave denominada **pivô**.
- S1 deverá conter todas as chaves cujos valores são **MENORES ou IGUAIS ao pivô**. Esse segmento está posicionado **à esquerda** de S2.
- S3 deverá conter todas as chaves cujos valores são **MAIORES do que o pivô**. Esse segmento está posicionado **à direita** de S2.



Exemplo:



(a) Fase de Divisão (b) Fase de Conquista

```
int  particiona ( tipo  *v, int inic, int fim )
{
    int pivotloc, ind;
    //coloca o pivot na primeira posicão do vetor
    troca ( &v[ inic], &v[(inic+ fim) / 2] )
    // considera  as posições restantes para a subdivisão
    pivotloc = inic ;
    // Coloca os menores que o pivot no início do vetor, marcando onde inicia o
vetor com os maiores que o pivot
    for ( ind = pivotloc + 1; ind <= fim ; ind ++ )
    {
            if   (v[ind] < v[inic])
                // troca o elem com a nova pos do pivot
                    troca  (&v[ ++ pivotloc],&v[ ind]);
    }
    // Coloca o pivot  no  meio das duas sublistas
    troca ( &v[inic],&v[pivotloc] );
    return pivotloc;
}
void quicksortR (tipo * v, int inic, int fim )
{
        int pivotloc;
        if  (fim > inic )       {
            pivotloc = particiona (v, inic, fim );
            quicksortR ( v, inic, pivotloc - 1 );
            quicksortR ( v, pivotloc +1, fim );
        }
}
```

Intercultural Computer Science Education

https://youtu.be/ywWBy6J5gz8

Método Ordenação Merge Sort:

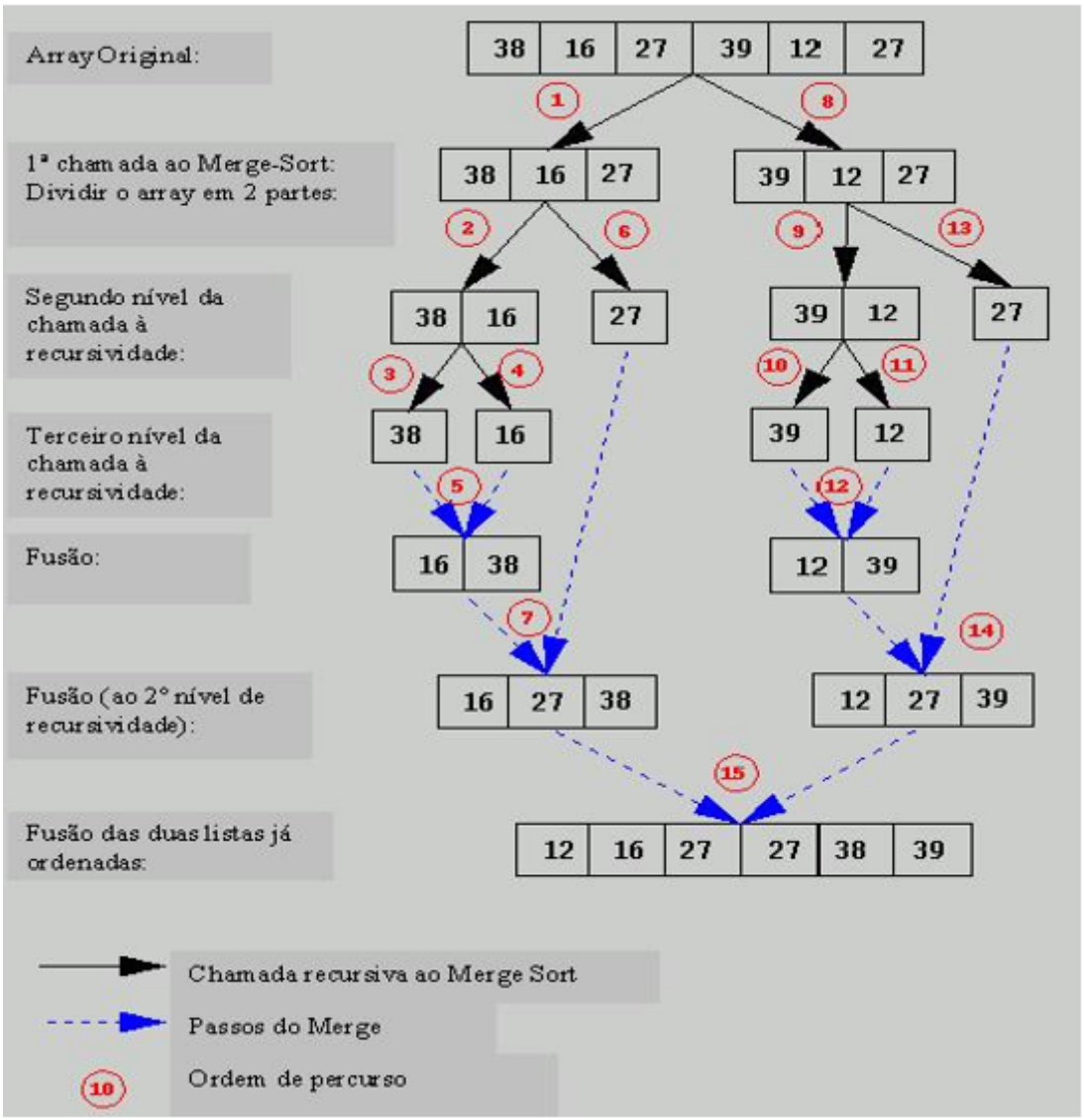
Classificação por intercalação. Baseia-se em um padrão de projeto fundamental para solução de problemas conhecida como **Divisão e Conquista** (D*ivide-and-Conquer*).

Merging: Junção/Intercalação de duas listas ordenadas em uma, mantendo a ordenação, tal que se há duas listas X (x_1<=x_2<=...<=x_m) e Y(y_1<=y_2<=...<=y_n) a lista resultante é: Z(z_1<=z_2<=...<=z_{m+n})

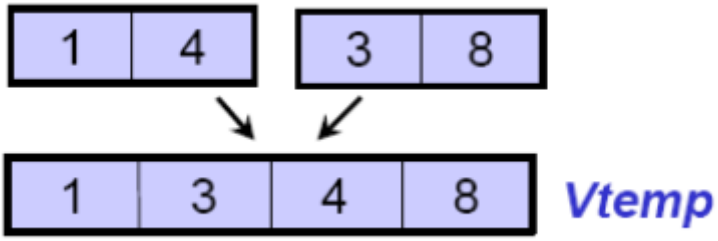
Ex: L1 = { 3 8 9 } L2 = { 1 5 7 }, merge(L1, L2) --> { 1 3 5 7 8 9 }

Ideia do método

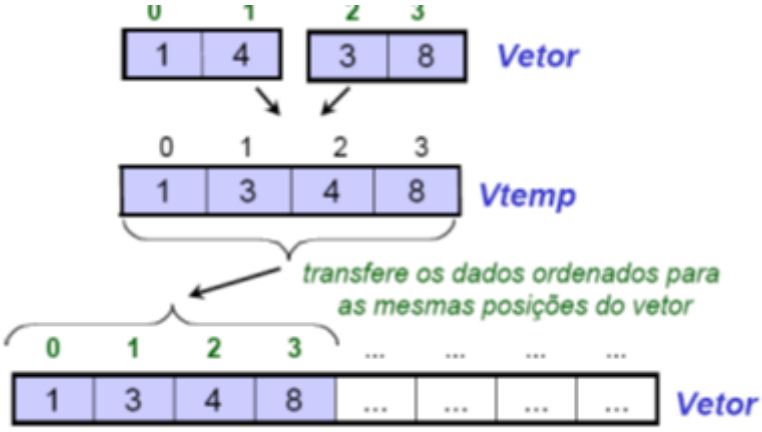
```
Dado uma lista L com tamanho k:

se k == 1 --> a lista está ordenada

senão:

    Divide a lista em duas partes ( esquerda e direita)
    Ordena o lado esquerdo (1 até k/2) pelo Merge Sort
    Ordena o lado direito (k/2+1 até k) pelo Merge Sort
    Intercala o lado direito com o lado esquerdo
```



Para a junção ou merge utiliza pum vetor temporário (**vTemp**) para manter o resultado da ordenação dos 2 sub-vetores.



Após a ordenação o conteúdo de **vTemp** é transferido para o vetor




```
void descarrega( TNO vorig[], int i_or, f_or, TNO vdest[], i_dest)
{
    while (i_or <= f_or){
        vdest[i_dest] = vorig[i_or];
        i_or++;
        i_dest++;
    }
}
void  Merge_SL (TNO vnos[], i_sl1, f_sl1, i_sl2, f_sl2) {
    /* sl1 e sl2 são sub listas ordenadas sobre o buffer vnos. São vizinhas no
buffer. A sub_lista resultante da combinação ordenada das duas sublistas deve
ficar na mesma área do buffer,  ié, sobre as duas sls  */

    TNO  bufferaux[MAX];    //buffer auxiliar para união
    int  i_slr = i_sl1;
    int  f_slr = f_sl2;
    int  i_aux = 0, f_aux=f_sl2 - i_sl1;

    while ( (i_sl1 <= f_sl1) && (i_sl2 <= f_sl2) )
    {
        if (vnos[i_sl1].id < vnos[i_sl2].id)    {
            bufferaux[i_aux] =vnos[i_sl1];
            i_sl1++;
        }
        else{
            bufferaux[i_aux] = vnos[i_sl2];
            i_sl2++;
        }
        i_aux++;
```

```
    }
    if (i_sl1 < f_sl1)  descarrega ( vnos, i_sl1, f_sl1, bufferaux, i_aux );
    else                descarrega ( vnos, i_sl2, f_sl2, bufferaux, i_aux );
    descarrega ( bufferaux, 0, f_aux, vnos, i_sl1, f_sl2 );
}


void MergeSort(TLISTA l)
{
    MergeSortR( l.vnos, 0, l.qtnos-1)
}
void MergeSortR (TNO vnos[], int inic, int fim)
{
    int meio;
    if (inic < fim){/* Tem mais de um elemento */
        meio = (inic + fim) / 2;
        MergeSortR (vnos, inic, meio);
        MergeSortR (vnos, meio+1, fim);
        Merge_SL (vnos, inic, meio, meio+1, fim);
    }
}
```

Exemplo de MergeSort

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	5	8	10	9	12	1	15	7	3	13	4	11	16	14
2	5	6	8	10	9	12	1	15	7	3	13	4	11	16	14
2	5	6	8	9	10	12	1	15	7	3	13	4	11	16	14
2	5	6	8	9	10	1	12	15	7	3	13	4	11	16	14
2	5	6	8	1	9	10	12	15	7	3	13	4	11	16	14
1	2	5	6	8	9	10	12	15	7	3	13	4	11	16	14
1	2	5	6	8	9	10	12	7	15	3	13	4	11	16	14
1	2	5	6	8	9	10	12	7	15	3	13	4	11	16	14
1	2	5	6	8	9	10	12	3	7	13	15	4	11	16	14
1	2	5	6	8	9	10	12	3	7	13	15	4	11	16	14
1	2	5	6	8	9	10	12	3	7	13	15	4	11	14	16
1	2	5	6	8	9	10	12	3	7	13	15	4	11	14	16
1	2	5	6	8	9	10	12	3	4	7	11	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

MergeSort:

Intercultural Computer Science Education

https://www.youtube.com/watch?v=XaqR3G_NVoo

Ferramentas de Simulação*

http://math.hws.edu/TMCM/java/xSortLab

http://www.sorting-algorithms.com/selection-sort

https://pt.wikipedia.org/wiki/Selection_sort#/media/File:Selection-Sort-Animation.gif