| Searching | Pior caso | Caso médio | Melhor caso |
|---|---|---|---|
| Busca Binária* | $O(\log n)$ | $O(\log n)$ | $O(1)$ |
| Busca Sequencial | $O(n)$ | $O(n)$ | $O(1)$ |

Lista em alocação sequencial; * Lista ordenada

Complexidade de algoritmos – notação Big O

| Sorting | Pior caso | Caso médio | Melhor caso |
|---|---|---|---|
| BubbleSort | $\Theta(n^2)$ | $\Theta(n^2)$ | $\Theta(n^2)$ |
| InsertionSort | $\Theta(n^2)$ | $\Theta(n^2)$ | $\Theta(n)$ |
| SelectionSort | $\Theta(n^2)$ | $\Theta(n^2)$ | $\Theta(n^2)$ |
| QuickSort* | $\Theta(n^2)$ | $O(n \log n)$ | $O(n \log n)$ |
| HeapSort** | $\Theta(n \log n)$ | $\Theta(n \log n)$ | $\Theta(n \log n)$ |
| MergeSort * | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ |

* recursivo ** organização hierárquica (árvore)

**BubbleSort(A)**

```
n ← nA;  fim ← n;
para j de 1 até (n-1) repita
    para k de 1 até (fim-1) repita
        se (A[k] > A[k+1])  então  aux ← A[k];
                                   A[k] ← A[k+1];      TROCA
                                   A[k+1] ← aux ;
    fim ← fim – 1;
```

**InsertionSort(A)**

```
n ← nA;
para j de 2 até n repita
    A[0] ← A[j];  k ← j-1;
    enquanto (A[0] < A[k]) faça
        A[k+1] ← A[k];    DESLOCAR PARA
        k ← k-1  ;        A DIREITA
    A[k+1] ← A[0];
```

**SelectionSort(A)**

```
n ← nA;  fim ← n;
para j de 1 até (n-1) repita
    pos ← 1;
    para k de 2 até fim repita
        se (A[k] > A[pos]) então pos ← k;
    se (pos ≠ fim) então  aux ← A[fim];
                          A[fim] ← A[pos];    TROCA
                          A[pos] ← aux  ;
    fim ← fim – 1;
```

**quicksort(p,n,A)**

```
se (p < n)  então | j← separarLista(p, n, A);
                  | quicksort(p, j-1, A);
                  | quicksort(j+1, n, A)
```

**separarLista(p,n,A)**

```
i ← p;  j ← n+1;
enquanto  (i < j)  faça
      | repita  i ← i+1 até que  ((A[i] ≥ A[p] ou (i=n));
      | repita  j ← j – 1  até que  A[j] ≤ A[p];
      | se (i < j) então | aux ← A[i];
      |                  | A[i] ← A[j];        TROCA
      |                  | A[j] ← aux;  ;
aux ← A[p];
A[p] ← A[j];     TROCA
A[j] ← aux;
devolver j
```

**mergeSort(p,n,A)**

```
se (p < n)  então | p1 ← p;   u1 ← (p + u) DIV 2;
                  | p2 ← u1 + 1;   u2 ← n;
                  | mergeSort(p1, u1, A);
                  | mergeSort(p2, u2, A);
                  | intercalar(p1, u1, p2, u2, A);
```

**intercalar(p1,u1,p2,u2,A)**

```
k ← p1;  j ← p2;  i ← 0; nB ← n;
enquanto  (k ≤ u1)  e  (j ≤ u2)  faça
      | se (A[k] ≤ A[j])  então  i ← i + 1;  B[i] ← A[k];  k ← k + 1;
      |                   senão  i ← i + 1];  B[i] ← A[j];  j ← j + 1;

se (k ≤ u1)   então  i ← i + 1;  B[i] ← A[k];  k ← k + 1;
              senão  i ← i + 1];  B[i] ← A[j];  j ← j + 1;

para k de p até n repita   A[k] ← B[k];
```