ALGORITMOS E ESTRUTURA DE DADOS

LINEAR SEARCH

É usado em situações onde o array não está ordenado, sendo a única possibilidade. Dessa maneira, olha-se valor por valor até encontrar o que se busca.

Upper Bound Running time: O(n) - Se estiver procurando um número na posição 100, então será necessário iterar as 100 posições.

Lower Bound Running time: O(1) - Pode acontecer um momento de sorte e ser o primeiro número, levando apenas 1 passo.

BINARY SEARCH

Demanda que o array esteja ordenado, dessa maneira pode-se olhar a posição central, conferir se o número é menor ou maior do que o desejado, descartando assim a metade do array que o número não se encontra, o que resulta em cortar o problema pela metade até que se encontre o valor.

Upper Bound Running Time - O (log n) - Como cortamos o problema pela metade a cada iteração, se torna logaritmo, assim conforme “n” aumenta, o tempo continua sendo cortado pela metade.

Lower Bound Running time: O(1) - Pode ocorrer um momento de sorte e ser o primeiro número, levando apenas 1 passo.

SELECTION SORT (ORDENAÇÃO)

Consiste em percorrer o array em um loop, conferindo qual é o menor valor entre todos, guardando em uma variável, e ao fim o trocando de posição com a primeira posição, depois com a segunda, assim por diante.

Upper Bound Running Time - O(n²) - É feito um loop para cada valor, e em cada um deles é feito um novo loop a partir de n-1 para saber se existe um número menor do que ele, e então as posições são trocadas, o que resulta em n iterações do primeiro loop vezes n-1 iterações do segundo loop, portanto **O(n²)**.

Lower Bound Running Time - O(n²) - É feito um loop para cada valor, e em cada um deles é feito um novo loop a partir de n-1 para saber se existe um número menor do que ele, e mesmo se estiver ordenado, irá fazer o mesmo processo.

BUBBLE SORT (ORDENAÇÃO)

Upper Bound Running Time - O(n²) - Compara pares de números, trocando suas posições se estiverem fora de ordem, levando o maior número do array sempre para o final. Dessa forma seria sempre n-1 * n-1, resultando em **O(n²)**.

Lower Bound Running Time - O(n) - Já em lower bound, ao comparar duplas de números, se torna **O(n)**, já que passa pelos valores pelo menos e apenas uma vez se já estiver ordenado.

RECURSION

Habilidade de uma função chamar ela mesma.

MERGE SORT (ORDENAÇÃO)

Upper Bound e Lower Bound são O(n log n) - Pois sempre se vai para o centro do array, ordena a metade da esquerda, a da direita, e junta-se ambas. Dessa forma, mesmo se o array já estiver ordenado, esse vai ser o tempo de execução.

ARRAYS EM C

Para adicionar um novo elemento em um array já preenchido, é necessário alocar memória, copiar os elementos e adicionar o novo elemento, com o tempo de execução de **O(n)**.

Upper Bound to Insert an Item Running Time - O(n) - Pois passa por todos os elementos “n” vezes até preencher o novo array.

Lower Bound to Insert an Item Running Time - O(1) - No melhor caso existe apenas um elemento para copiar.

Upper Bound of Searching - O(1) - Caso saiba a exata posição do que procura, pode acessá-la diretamente em um passo.

LINKED LISTS

Em C, pode se criar uma tipagem que guarda o valor desejado e o ponteiro para o próximo valor,

sendo mais eficiente em inserções do que o Array com o número de posições fixadas.

Upper Bound of Searching - O(n) - Se estiver procurando pelo número 100, irá ter 100 passos até o encontrar.

LINKED LISTS INSERTIONS

Running Time O(1) - Se não for necessário manter a sequência ordenada, pode-se apenas adicionar um elemento ao começo da lista, sem ter que seguir todos os ponteiros até o final.

TREES

Insertion Time O(log n) - Pois para encontrar a posição do novo elemento, é realizado uma busca utilizando-se de **Binary Search,** que também é **O(log n)**.

Porém é necessário criar uma verificação para que a estrutura de árvores não se torne uma linked list como “sintoma”.

BINARY SEARCH TREES

Elementos armazenados com um elemento “root”, que contém sempre um número menor a esquerda e um maior a direita, o que nos permite percorrer suas “sub-árvores” através de binary search, assim como em arrays.

HASH TABLES

Se trata de um array de linked lists, onde o tempo de execução para procurar algo academicamente ainda é **O(n)**, porém em quanto mais “hashs” o array é divido, menor será o tempo de execução, porém custando mais memória.

O tempo para inserir um item é o mesmo da linked list, já que sabemos exatamente em que posição do array tem que ser armazenado, e nesse index a colocamos como a primeira da linked list, portanto **O(1)**.

HASH FUNCTIONS

Funções que recebem inputs, e através de alguma fórmula nos dizem onde o valor que queremos está armazenado ou em que index do array deve ser armazenada a nova informação.

TRIES

Uma árvore feita de arrays de ponteiros para outros nodes. Tempo de execução de **O(1)**, pois em um caso onde é necessário encontrar o nome de alguém, cada letra do nome que possui um índice correspondente nos arrays, irá apontar para a posição da próxima letra (ou não, caso o nome não exista na estrutura), levando como tempo o exato número de caracteres do nome a ser encontrado, portanto constante.

QUEUES

Estrutura de dados onde o primeiro a entrar é o primeiro a sair.

First in, First Out - Enqueue, Dequeue

STACKS

Estrutura de dados onde o último a entrar é o primeiro a sair.

Last in, First out - Push, Pop.

DICTIONARY

Associar chaves com valores.