In graphs: dfs and bfs (stack and queue): https://www.youtube.com/watch?v=bIA8HEEUxZI

1. **Linear** **Search**:

Description: Linear Search is implemented by looping through a list of elements and compare each one as we pass by with the desired element.

Steps: Iterate from 0 to N-1 and compare the value of every index with x if they match return index

Time Complexity: O(N)

C++ Implementation:

int search(int arr[], int size, int desiredElement) {

int i;

for (i = 0; i < size; i++)

if (arr[i] == desiredElement)

return i;

return -1;

}

int main() {

int arr[] = { 2, 3, 4, 10, 40 };

int desiredElement = 10;

int size = sizeof(arr) / sizeof(arr[0]);

// Function call

int result = search(arr, size, desiredElement);

if (result == -1) {

cout << "Element is not present in array";

} else {

cout << "Element is present at index " << result;

}

}

1. **Binary** **Search:**

Description: Binary search is used in a **sorted** array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

Steps:

1. Compare x with the middle element.
2. If x matches with the middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in the right half subarray after the mid element. So we repeat for the right half.
4. Else (x is smaller) repeat for the left half.

Complexity: O(Log n)

C++ Implementation:

void binarySearch(vector<int> v, int To\_Find)

{

int low = 0, high = v.size() - 1;

int mid;

// This below check covers all cases , so need to check

// for mid = low - (high - low) / 2

while (high - low > 1) {

int mid = (high + low) / 2;

if (To\_Find > v[mid]) {

low = mid + 1;

} else {

high = mid;

}

}

if (v[low] == To\_Find) {

cout << "Found"

<< " At Index " << low << endl;

} else if (v[high] == To\_Find) {

cout << "Found"

<< " At Index " << high << endl;

} else {

cout << "Not Found" << endl;

}

}

int main()

{

vector<int> v = { 1, 3, 4, 5, 6 };

int To\_Find = 1;

binarySearch(v, To\_Find);

return 0;

}

1. **Ternary** **Search**

Description: we divide the array into two parts but in this algorithm, we divide the given array into three parts and determine which has the key (searched element). We can divide the array into three parts by taking mid1 and mid2 which can be calculated as shown below. Initially, l and r will be equal to 0 and n-1 respectively, where n is the length of the array. It is same as the binary search. The only difference is that, it reduces the time complexity a bit more. Its time complexity is O(log n base 3) and that of binary search is O(log n base 2). (I prefer this over binary when the list is sorted)

Steps:

1. First, we compare the desired element with the element at mid1. If found equal, we return mid1.
2. If not, then we compare the desired element with the element at mid2. If found equal, we return mid2.
3. If not, then we check whether the desired element is less than the element at mid1. If yes, then repeat to the first part.

If not, then we check whether the key is greater than the element at mid2. If yes, then repeat to the third part.

If not, then we repeat to the second (middle) part.

Complexity: O(log3n)

C++ Implementation:

int ternarySearch(int l, int r, int key, int ar[])

{

while (r >= l) {

// Find the mid1 and mid2

int mid1 = l + (r - l) / 3;

int mid2 = r - (r - l) / 3;

// Check if key is present at any mid

if (ar[mid1] == key) {

return mid1;

}

if (ar[mid2] == key) {

return mid2;

}

// Since key is not present at mid,

// check in which region it is present

// then repeat the Search operation

// in that region

if (key < ar[mid1]) {

// The key lies in between l and mid1

r = mid1 - 1;

} else if (key > ar[mid2]) {

// The key lies in between mid2 and r

l = mid2 + 1;

} else {

// The key lies in between mid1 and mid2

l = mid1 + 1;

r = mid2 - 1;

}

}

// Key not found

return -1;

}

// Driver code

int main()

{

int l, r, p, key;

// Get the array

// Sort the array if not sorted

int ar[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Starting index

l = 0;

// length of array

r = 9;

// Checking for 5

// Key to be searched in the array

key = 5;

// Search the key using ternarySearch

p = ternarySearch(l, r, key, ar);

// Print the result

cout << "Index of " << key << " is " << p << endl;

// Checking for 50

// Key to be searched in the array

key = 50;

// Search the key using ternarySearch

p = ternarySearch(l, r, key, ar);

// Print the result

cout << "Index of " << key << " is " << p;

}