Searching Algorithms

Emmanuel Ameh

Evaluating Searching Algorithms

The following project assignment was completed using the Java Programming Language and Visual Studio Code. It includes Java code for Linear Sort, both integer and string inputs as well as Binary Sort, both integer and string inputs. Screenshots of the Java debugging console depicting the executed code are included in this assignment. Attached in separate files are the java codes for implementation. The Big-O Time Complexity Analysis of Linear Sort and Binary Sort were also explained.

**Java Code for Linear and Binary Search Algorithms**

class LinBinSearch {

    public static int linearSearch(int[] arr, int valueToFind){

int index=-1;

int n=arr.length;

for (int i=0;i<n-1;i++) {

if (arr[i]==valueToFind) {

index=i;

break;

}

}

return index;

}

public static int linearSearch(String[] arr,String valueToFind){

int index=-1;

int n=arr.length;

for (int i=0;i<n-1;i++) {

if (arr[i].compareTo(valueToFind) ==0 ) {

index=i;

break;

}

}

return index;

}

public static int binarySearch(int[] arr, int valueToFind){

int n = arr.length;

int hi=n-1, low=0;

while(hi>low){

int mid=(hi+low)/2;

if (arr[mid] > valueToFind) {

hi=mid;

}

else if(arr[mid] < valueToFind){

low=mid+1;

}

else{

return mid;

}

}

return -1;

}

public static int binarySearch(String[] arr, String valueToFind){

int n = arr.length;

int hi=n-1, low=0;

while(hi > low){

// System.out.println(low+" "mid+" "+high);

int mid=(hi+low)/2;

System.out.println(low+" "+mid+" "+hi);

if (arr[mid].compareTo(valueToFind) > 0) {

hi=mid;

}

else if(arr[mid].compareTo(valueToFind) < 0){

low=mid+1;

}

else{

return mid;

}

}

return -1;

}

public static void main(String[] args) {

int[] arr={1,2,5,4,3};

String[] sarr={"my","name", "is", "Ann"};

System.out.println("Linear search of 1 returned "+linearSearch(arr,1));

System.out.println("Linear search of 0 returned "+linearSearch(arr,0));

System.out.println("Linear search of \"is\" returned "+linearSearch(sarr,"is"));

System.out.println("Linear search of \" king\" returned "+linearSearch(sarr,"king"));

System.out.println("Binary search of 1 returned "+binarySearch(arr,1));

System.out.println("Binary search of 0 returned "+binarySearch(arr,0));

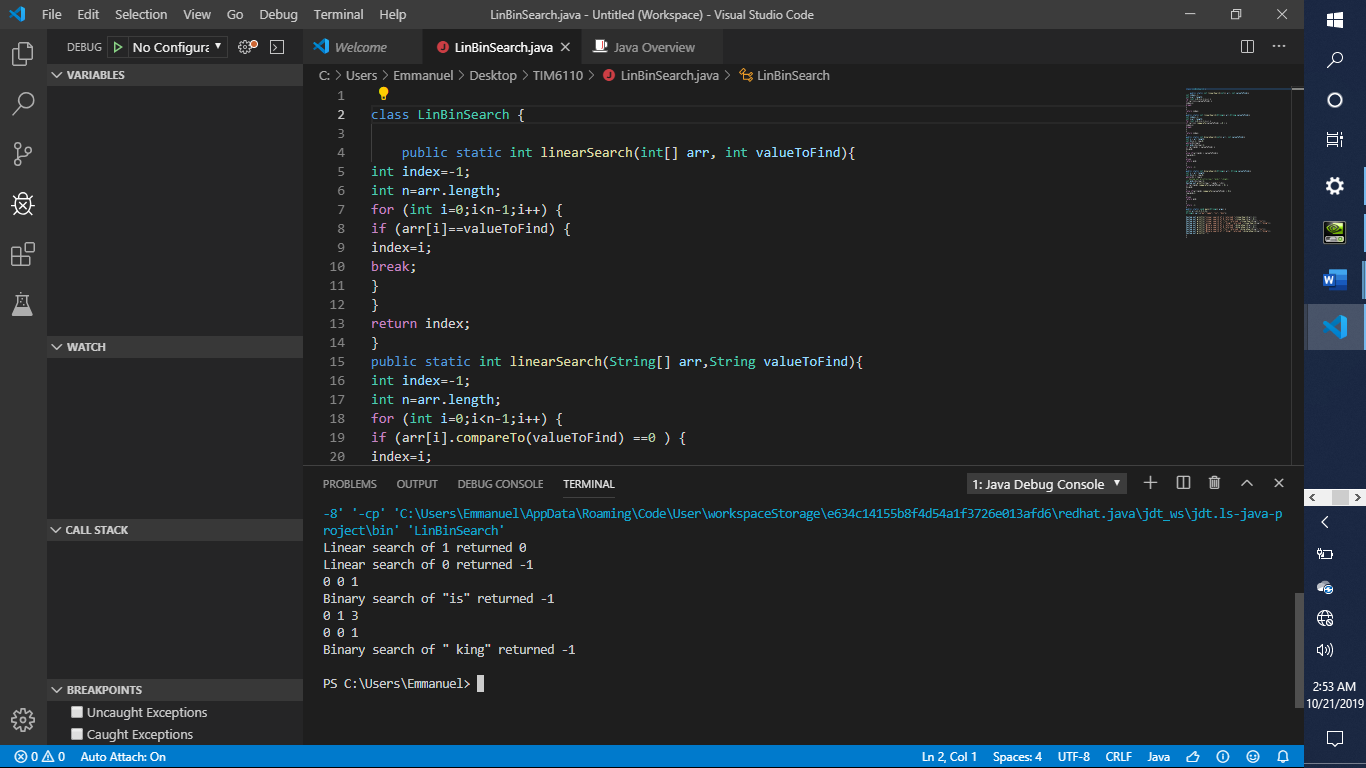
System.out.println("Binary search of \"is\" returned "+binarySearch(sarr,"is"));

System.out.println("Binary search of \" king\" returned "+binarySearch(sarr,"king"));

System.out.println("");

}

}



*Figure 1.* Java Debug Console and editor depicting execution of linear and binary searches.

**Big-O Complexity Notation:**

**Linear Search**

Linear Search algorithm has the Big-O Complexity Notation: O(N). As the name implies

a linear algorithm is applied when the algorithm’s execution-time grows directly proportional to

the size of the data set. With its single loop of iteration through each dataset value, the linear

notation O(N) is likelier.

**Binary Search**

Binary searches are classical examples of logarithmic algorithms. The logarithmic

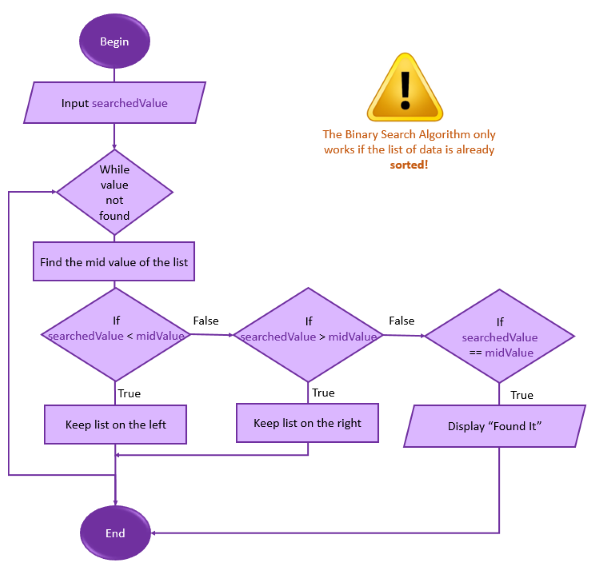
algorithm O(log(N)) has its growth decreasing with increasing dataset, to follow a logarithmic

curve. In processing large datasets, logarithmic algorithms are more efficient than their

exponential or linear counterparts. During a binary search, one-half of the dataset is rejected

after every iteration.

**Binary Search Algorithm Flowchart**



*Figure 2.* Binary Search Algorithm Flowchart. Retrieved from https://www.101computing.net/big-o-notation/