**TITLE: SEARCHING**

Finding the essential information from a group of items stored as elements in the computer memory is referred to as searching in data structures. Among them are:

**Linear Search: a linear search or sequential search is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.**

**Algorithm for Linear search**:

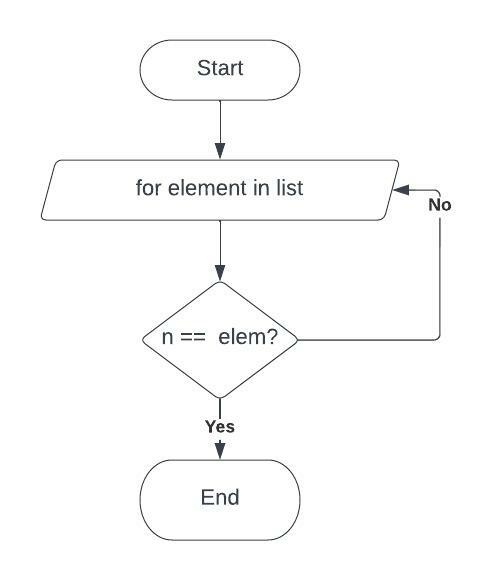
Input: n, list[n]

Func bubbleSort(n,list)

for element in list:

if n == element:

return True



flowchart for linear search

**WAP to implement Linear Search.**

#include <stdio.h>

int linearSearch(int arr[],int size,int searchFor){

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

if (arr[i] == searchFor){

return i;

}

}

return -1;

}

int main(){

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

int size = sizeof(arr) / sizeof(int);

int searchFor = 2;

int indexof = linearSearch(arr,size,searchFor);

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

printf("%d ",arr[i]);

}

printf("\n");

if (indexof >=0){

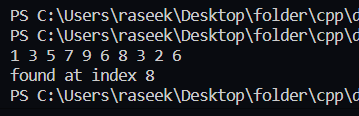
printf("found at index %d",indexof);

}else{

printf("not found");

}

}

**Binary Search:** Binary Search is a searching algorithm 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)

Algorithm for selection sort:

1. Low = 0, high= len(list) - 1
2. Set mid = (low + high) / 2
3. If n equals list[mid] then, return index of n
4. If list[mid] > n, high = mid – 1, else low = mid + 1
5. Repeat step 2 to 4 until (low <= high)

Pseudocode:

Input: n,list[n]

Func BinarySearch(n,list[n])

Low = 0 , high = sizeoflist – 1

While (low <= high):

mid = (low+high)/2

If (lis[mid] == n), return True

If ( list[mid] > n ) then high = mid - 1

else low = mid + 1

end while

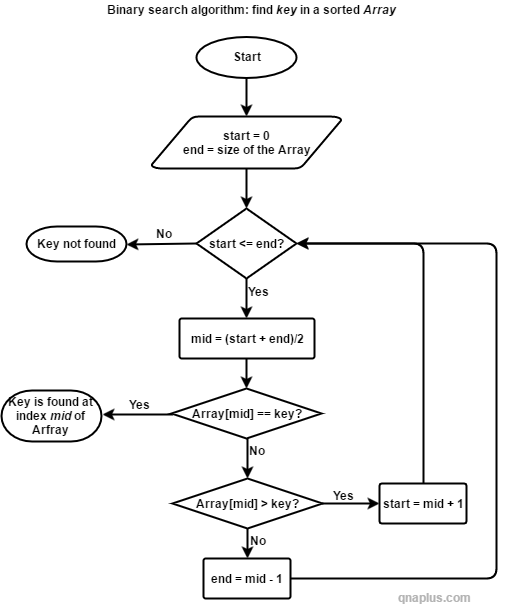


Figure: Binary Search

**WAP to implement Binary Search.**

#include <stdio.h>

int binarySearch(int arr[],int size,int searchFor){

int low = 0;

int high = size - 1;

int mid;

while (low <= high){

mid = (low + high) / 2;

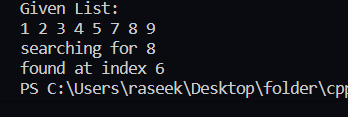
if (arr[mid] == searchFor){

return mid;

}

if (arr[mid] > searchFor){

high = mid - 1;

 }

else{

low = mid + 1;

}

}

}

int main(){

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

int size = sizeof(arr) / sizeof(int);

int searchFor = 8;

int indexof = binarySearch(arr,size,searchFor);

printf("Given List:\n" );

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

printf("%d ",arr[i]);

}

printf("\n");

printf("searching for %d\n",searchFor);

if (indexof >=0){

printf("found at index %d",indexof);

}else{

printf("not found");

}

}

**Hashing:** Hashing is designed to solve the problem of needing to efficiently find or store an item in a collection. Hashing is a technique to make things more efficient by effectively narrowing down the search at the outset.

**Algorithm for insertion sort:**

1. Get hash using n % sizeoflist
2. If hash[key] == n, return key
3. Else calculate hash for n by add 1 until n is discovered
4. If n is not discovered then n is not present in the list.

Pseudocode:

func searchFromHashTable(int hash[],int searchFor,int size)

int key = getKey(searchFor,size)

if (hash[key] == searchFor)

return key

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

key = getKey(key+1,size)

if (hash[key] == searchFor)

return key

return -1

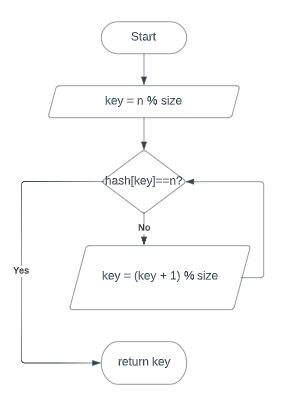


Figure: Hashing search

**WAP to implement insertion sort.**

#include <stdio.h>

#include <stdlib.h>

int getKey(int x,int size){

return x % size;

}

void generateHashTable(int \*hash,int \*arr,int size){

int key;

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

hash[i] = -1;

}

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

key = getKey(arr[i],size);

if (hash[key] == -1){

hash[key] = arr[i];

}

else{ // collision occured

//solving collision through linear probing

while (hash[key] != -1){

key = getKey(key+1,size);

}

hash[key] = arr[i];

}

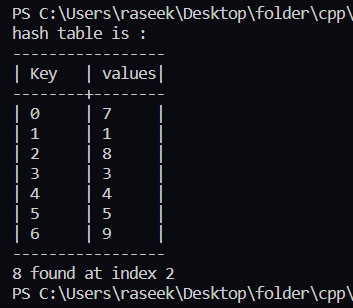
}

}

int searchFromHashTable(int hash[],int searchFor,int size){

int key = getKey(searchFor,size);

if (hash[key] == searchFor){

 return key;

}

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

key = getKey(key+1,size);

if (hash[key] == searchFor){

return key;

}

}

return -1;

}

int main(){

int arr[] = {1,4,5,7,3,8,9};

int size = sizeof(arr) / sizeof(int);

int hash[size]; //key will be the index

generateHashTable(&hash[0],&arr[0],size);

printf("hash table is : \n");

printf("-----------------\n");

printf("| Key \t| values|\n");

printf("--------+--------\n");

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

printf("| %d\t| %d\t|\n",i,hash[i]);

}

printf("-----------------\n");

int searchFor = 8;

int result = searchFromHashTable(hash,searchFor,size);

printf("%d found at index %d",searchFor,result);

}