Exercise-1 Module-7

Write a Pseudocode for these problems

1. S = (A + B + C) / Y

Step 1 Start

Step 2 Read values for X, Y and Z as input

Step 3 calculate W = X + Y + Z

Step 4 calculate S = W / 3

Step 5 Write the value of S

Step 6 Stop

2. Convert from Celsius to Fahrenheit (Multiply by 9, then divide by 5, then add 32) F = (9/5)C + 32

step 1 Start

Step 2 Read the value of c as input

step 3 Calculate X = 9 * c

step 4 Calculate Y = X / 5

step 5 Calculate F = Y + 32

Step 6 Write the values of F

Step 7 Stop

3. Area of Circle ($A = \pi r2$)

Step 1 Start

step 2 Get the value of radius r as input

step 3 Calculate R= (r * r)

step 4 Calculate A = 3.14 * R

step 5 Write the value of A

step 6 Stop

4. Volume of Sphere (4/3πr3)

step 1 Start

step 2 Get the value of r as input

step 3 Calculate R=(r * r * r)

step 4 Calculate A = 3*3.14* R

step 5 Calculate S= 4 / A

step 6 Write the value of S

step 7 Stop

5. Average speed = $Distance\ Travele$ s=d/t

step 1 Start

step 2 Get the value of d and t as input

Step 3 calculate S = d/t

step 4 Write the value of S

step 5 Stop

7.ALGORITHM

Exercise-2

Case 1

*If we are taking the first condition, the person will call his friend before he reaches the bus stop and his friend will wait before he reaches the bus stop. Also, he doesn't want to spend time traveling. So, it is a best condition.

Case 2

*If we take the second condition, the person will take the auto and reach the destination on time but the cost will be little high. So, it is average condition.

Case 3

* If we take the third condition, the person has to take a new bus. Even after he reaches the bus stop he also has to wait for another bus so it will take some time but he has to spend less money only. So, it is the worst condition.

EXERCISE-3 MODULE-7

1.Quick Sort Algorithm

Quick sort is one of the most famous sorting algorithms based on divide and conquers strategy which results in an O(n log n) complexity.

So, the algorithm starts by picking a single item which is called pivot and moving all smaller items before it, while all greater elements in the later portion of the list.

This is the main quicksort operation named as a partition, recursively repeated on lesser and greater sublists until their size is one or zero - in which case the list is wholly sorted.

Choosing an appropriate pivot, as an example, the central element is essential for avoiding the severely reduced performance of O(n2).

Choosing the pivot

Picking a good pivot is necessary for the fast implementation of quicksort. However, it is typical to determine a good pivot. Some of the ways of choosing a pivot are as follows -

Pivot can be random, i.e. select the random pivot from the given array.

Pivot can either be the rightmost element of the leftmost element of the given array.

Select median as the pivot element.

Algorithm For Quick_Sort(list)

Pre: list 6= fi

Post: the list has been sorted in ascending order

if list.Count = 1 // list already sorted

return list

end if

pivot <- Median_Value(list)

for i <- 0 to list.Count - 1

if list[i] = pivot

equal.Insert(list[i])

end if

```
if list[i] < pivot

less.Insert(list[i])

end if

if list[i] > pivot

greater.Insert(list[i])

end if

end for

return Concatenate(Quick_Sort(less), equal, Quick_Sort(greater))

end Quick_sort
```

Working of Quick Sort Algorithm

Pick an element, called a pivot or refer it as the partitioning element, from the array. For instance, let us consider the last element as a pivot.

Partition: rearrange the array such that all elements with values less than the pivot come before the pivot (i.e., on the left of the pivot), while all elements with values greater than the pivot come after it (i.e., on the right of pivot). The pivot element will be in its final position after it.

Recursively apply the above steps to the sub-array of elements on the left side of the pivot and on the right side of the pivot to get a sorted array.

Now, using pictorial representation, let's try to sort an array which has initial values as X= {75, 26, 15, 67, 54, 31, 49}.

In the above representation, we select the element at the last index of the array as a pivot, which is 49 and then call partition() to thereby re-arrange the elements of the array in a way that elements less than 49 are before it in and elements greater than 49 are after it.

Then, after the first pass, we consider the subarrays at left and right and select a partition element for them. Here, in the above representation, we select 31 as a pivot from the left array and 67 as a pivot from the right array. And once again call the partition() to split the array.

Thus, we recursively keep partitioning the array with help of pivot unless we obtain a sorted array.

2.Selection Sort Algorithm

Selection Sort algorithm is used to arrange a list of elements in a particular order (Ascending or Descending).

In selection sort, the first element in the list is selected and it is compared repeatedly with all the remaining elements in the list.

If any element is smaller than the selected element (for Ascending order), then both are swapped so that first position is filled with the smallest element in the sorted order.

Next, we select the element at a second position in the list and it is compared with all the remaining elements in the list.

If any element is smaller than the selected element, then both are swapped. This procedure is repeated until the entire list is sorted.

Step by Step Process

The selection sort algorithm is performed using the following steps...

Step 1 - Select the first element of the list (i.e., Element at first position in the list).

Step 2: Compare the selected element with all the other elements in the list.

Step 3: In every comparision, if any element is found smaller than the selected element (for Ascending order), then both are swapped.

Step 4: Repeat the same procedure with element in the next position in the list till the entire list is sorted.

FOR EXAMPLE

Lets consider the following array as an example: arr[] = {64, 25, 12, 22, 11}

First pass:

For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where 64 is stored presently, after traversing whole array it is clear that 11 is the lowest value.

64 25 12 22 11

Thus, replace 64 with 11. After one iteration 11, which happens to be the least value in the array, tends to appear in the first position of the sorted list.

11 25 12 22 64

Second Pass:

For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.

11 25 12 22 64

After traversing, we found that 12 is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.

11 12 25 22 64

Third Pass:

Now, for third place, where 25 is present again traverse the rest of the array and find the third least value present in the array.

11 12 25 22 64

While traversing, 22 came out to be the third least value and it should appear at the third place in the array, thus swap 22 with element present at third position.

11 12 22 25 64

Fourth pass:

Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array

As 25 is the 4th lowest value hence, it will place at the fourth position.

11 12 22 25 64

Fifth Pass:

At last the largest value present in the array automatically get placed at the last position in the array

The resulted array is the sorted array.

11 12 22 25 64

Exercise-4

1.LINEAR SEARCH ALGORITHM

Linear search, often known as sequential search, is the most basic search technique.

Take the entire list and search for the particular element from the list by checking each element from the beginning

In linear search the element index is noted as a result.

If the element is not found, then it returns a NULL value.

Here is an example for a linear search algorithm.

The procedures for implementing linear search are as follows:

Step 1: First, read the search element (Target element) in the array.

Step 2: In the second step compare the search element with the first element in the array.

Step 3: If both are matched, display "Target element is found" and terminate the Linear Search function.

Step 4: If both are not matched, compare the search element with the next element in the array.

Step 5: In this step, repeat steps 3 and 4 until the search (Target) element is compared with the last element of the array.

Step 6 - If the last element in the list does not match, the Linear Search Function will be terminated, and the message "Element is not found" will be displayed.

FOR EXAMPLE:

The given list consists of 7 elements that are 6 arrays. Now, find out 39 from the list of indexes

Algorithm:

Step 1: First, read the search element 39 in the array.

Step 2: Now, compare each element in the array and find 39.

Step3: if we take the first element a[0] it is 13 . So, Element is not found.

Step4: Now move to the next element a[1] is 9. Element is not found.

Step5: Like this move to the next next element a[2] is 21, a[3] is 15.

Step6: Now the next element a[4] is 39 it is matching the target. So, the target element is found.

2.Binary Search Algorithm

Binary search is the most frequently used technique as it is much faster than a linear search.

In the binary search method, the collection is repeatedly divided into half and the key element is searched in the left or right half of the collection depending on whether the key is less than or greater than the mid element of the collection.

A simple Binary Search Algorithm is as follows:

Calculate the mid element of the collection.

Compare the key items with the mid element.

If key = middle element, then we return the mid index position for the key found.

Else If key > mid element, then the key lies in the right half of the collection. Thus repeat steps 1 to 3 on the lower (right) half of the collection.

Else key < mid element, then the key is in the upper half of the collection. Hence you need to repeat the binary search in the upper half.

For example, take the following sorted array of 10 elements

Let's calculate the middle location of the array.

Mid = 0+9/2 = 4

#1) Key = 21

First, we will compare the key value with the [mid] element and we find that the element value at mid = 21.

Thus we find that key = [mid]. Hence the key is found at position 4 in the array.

#2) Key = 25

We first compare the key value to mid. As (21 < 25), we will directly search for the key in the upper half of the array.

Now again we will find the mid for the upper half of the array.

Mid = 4+9/2 = 6

The value at location [mid] = 25

Now we compare the key element with the mid element. So (25 == 25), hence we have found the key at location [mid] = 6.

Thus we repeatedly divide the array and by comparing the key element with the mid, we decide in which half to search for the key. Exercise-4

1.LINEAR SEARCH ALGORITHM

Linear search, often known as sequential search, is the most basic search technique.

Take the entire list and search for the particular element from the list by checking each element from the beginning

In linear search the element index is noted as a result.

If the element is not found, then it returns a NULL value.

Here is an example for a linear search algorithm.

The procedures for implementing linear search are as follows:

Step 1: First, read the search element (Target element) in the array.

Step 2: In the second step compare the search element with the first element in the array.

Step 3: If both are matched, display "Target element is found" and terminate the Linear Search function.

Step 4: If both are not matched, compare the search element with the next element in the array.

Step 5: In this step, repeat steps 3 and 4 until the search (Target) element is compared with the last element of the array.

Step 6 - If the last element in the list does not match, the Linear Search Function will be terminated, and the message "Element is not found" will be displayed.

FOR EXAMPLE:

The given list consists of 7 elements that are 6 arrays. Now, find out 39 from the list of indexes

Algorithm:

Step 1: First, read the search element 39 in the array.

Step 2: Now, compare each element in the array and find 39.

Step3: if we take the first element a[0] it is 13 . So, Element is not found.

Step4: Now move to the next element a[1] is 9. Element is not found.

Step5: Like this move to the next next element a[2] is 21, a[3] is 15.

Step6: Now the next element a[4] is 39 it is matching the target. So, the target element is found.

2.Binary Search Algorithm

Binary search is the most frequently used technique as it is much faster than a linear search.

In the binary search method, the collection is repeatedly divided into half and the key element is searched in the left or right half of the collection depending on whether the key is less than or greater than the mid element of the collection.

A simple Binary Search Algorithm is as follows:

Calculate the mid element of the collection.

Compare the key items with the mid element.

If key = middle element, then we return the mid index position for the key found.

Else If key > mid element, then the key lies in the right half of the collection. Thus repeat steps 1 to 3 on the lower (right) half of the collection.

Else key < mid element, then the key is in the upper half of the collection. Hence you need to repeat the binary search in the upper half.

For example, take the following sorted array of 10 elements

Let's calculate the middle location of the array.

Mid = 0+9/2 = 4

#1) Key = 21

First, we will compare the key value with the [mid] element and we find that the element value at mid = 21.

Thus we find that key = [mid]. Hence the key is found at position 4 in the array.

#2) Key = 25

We first compare the key value to mid. As (21 < 25), we will directly search for the key in the upper half of the array.

Now again we will find the mid for the upper half of the array.

Mid = 4+9/2 = 6

The value at location [mid] = 25

Now we compare the key element with the mid element. So (25 == 25), hence we have found the key at location [mid] = 6.

Thus we repeatedly divide the array and by comparing the key element with the mid, we decide in which half to search for the key. Exercise-4

1.LINEAR SEARCH ALGORITHM

Linear search, often known as sequential search, is the most basic search technique.

Take the entire list and search for the particular element from the list by checking each element from the beginning

In linear search the element index is noted as a result.

If the element is not found, then it returns a NULL value.

Here is an example for a linear search algorithm.

The procedures for implementing linear search are as follows:

Step 1: First, read the search element (Target element) in the array.

Step 2: In the second step compare the search element with the first element in the array.

Step 3: If both are matched, display "Target element is found" and terminate the Linear Search function.

Step 4: If both are not matched, compare the search element with the next element in the array.

Step 5: In this step, repeat steps 3 and 4 until the search (Target) element is compared with the last element of the array.

Step 6 - If the last element in the list does not match, the Linear Search Function will be terminated, and the message "Element is not found" will be displayed.

FOR EXAMPLE:

The given list consists of 7 elements that are 6 arrays. Now, find out 39 from the list of indexes

Algorithm:

Step 1: First, read the search element 39 in the array.

Step 2: Now, compare each element in the array and find 39.

Step3: if we take the first element a[0] it is 13 . So, Element is not found.

Step4: Now move to the next element a[1] is 9. Element is not found.

Step5: Like this move to the next next element a[2] is 21, a[3] is 15.

Step6: Now the next element a[4] is 39 it is matching the target. So, the target element is found.

2.Binary Search Algorithm

Binary search is the most frequently used technique as it is much faster than a linear search.

In the binary search method, the collection is repeatedly divided into half and the key element is searched in the left or right half of the collection depending on whether the key is less than or greater than the mid element of the collection.

A simple Binary Search Algorithm is as follows:

Calculate the mid element of the collection.

Compare the key items with the mid element.

If key = middle element, then we return the mid index position for the key found.

Else If key > mid element, then the key lies in the right half of the collection. Thus repeat steps 1 to 3 on the lower (right) half of the collection.

Else key < mid element, then the key is in the upper half of the collection. Hence you need to repeat the binary search in the upper half.

For example, take the following sorted array of 10 elements

Let’s calculate the middle location of the array.

Mid = 0+9/2 = 4

#1) Key = 21

First, we will compare the key value with the [mid] element and we find that the element value at mid = 21.

Thus we find that key = [mid]. Hence the key is found at position 4 in the array.

#2) Key = 25

We first compare the key value to mid. As (21 < 25), we will directly search for the key in the upper half of the array.

Now again we will find the mid for the upper half of the array.

Mid = 4+9/2 = 6

The value at location [mid] = 25

Now we compare the key element with the mid element. So (25 == 25), hence we have found the key at location [mid] = 6.

Thus we repeatedly divide the array and by comparing the key element with the mid, we decide in which half to search for the key. Exercise-4

1.LINEAR SEARCH ALGORITHM

Linear search, often known as sequential search, is the most basic search technique.

Take the entire list and search for the particular element from the list by checking each element from the beginning

In linear search the element index is noted as a result.

If the element is not found, then it returns a NULL value.

Here is an example for a linear search algorithm.

The procedures for implementing linear search are as follows:

Step 1: First, read the search element (Target element) in the array.

Step 2: In the second step compare the search element with the first element in the array.

Step 3: If both are matched, display "Target element is found" and terminate the Linear Search function.

Step 4: If both are not matched, compare the search element with the next element in the array.

Step 5: In this step, repeat steps 3 and 4 until the search (Target) element is compared with the last element of the array.

Step 6 - If the last element in the list does not match, the Linear Search Function will be terminated, and the message "Element is not found" will be displayed.

FOR EXAMPLE:

The given list consists of 7 elements that are 6 arrays. Now, find out 39 from the list of indexes

Algorithm:

Step 1: First, read the search element 39 in the array.

Step 2: Now, compare each element in the array and find 39.

Step3: if we take the first element a[0] it is 13 . So, Element is not found.

Step4: Now move to the next element a[1] is 9. Element is not found.

Step5: Like this move to the next next element a[2] is 21, a[3] is 15.

Step6: Now the next element a[4] is 39 it is matching the target. So, the target element is found.

2.Binary Search Algorithm

Binary search is the most frequently used technique as it is much faster than a linear search.

In the binary search method, the collection is repeatedly divided into half and the key element is searched in the left or right half of the collection depending on whether the key is less than or greater than the mid element of the collection.

A simple Binary Search Algorithm is as follows:

Calculate the mid element of the collection.

Compare the key items with the mid element.

If key = middle element, then we return the mid index position for the key found.

Else If key > mid element, then the key lies in the right half of the collection. Thus repeat steps 1 to 3 on the lower (right) half of the collection.

Else key < mid element, then the key is in the upper half of the collection. Hence you need to repeat the binary search in the upper half.

For example, take the following sorted array of 10 elements

Let's calculate the middle location of the array.

Mid = 0+9/2 = 4

#1) Key = 21

First, we will compare the key value with the [mid] element and we find that the element value at mid = 21.

Thus we find that key = [mid]. Hence the key is found at position 4 in the array.

#2) Key = 25

We first compare the key value to mid. As (21 < 25), we will directly search for the key in the upper half of the array.

Now again we will find the mid for the upper half of the array.

Mid = 4+9/2 = 6

The value at location [mid] = 25

Now we compare the key element with the mid element. So (25 == 25), hence we have found the key at location [mid] = 6.

Thus we repeatedly divide the array and by comparing the key element with the mid, we decide in which half to search for the key.