\*multithreading is mostly used for producer and consumer.

\*what is circular queue and priority queue.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Algorithms\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Algorithms:Set of instructions

searching Algorithms:

There are two types of searching Algorithm

1)Sequential serach:Time consuming is more.

\*it can be implemented for any unsorted item.

Ex:Linear search

2)Intervalsearch :time consuming is less.

\*it is specifically for searching in sorted data-structures

\*Repeatedly target the centre of the search structure and divide the search space in half.

Ex:Binary serach

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Binary Search:

\*In this approach the element is always searched in the middle of a portion of an array.

\*Binary serach can be implemented in two ways

1)Iterative Method:

EX:do untill the pointers low nad high meet each other

mid=(low+high)/2

if(x=xarr[mid])

return mid

elif(x>arr[mid]) //x is on the right side

low=mid+1

else //x is on the left side

high=mid-1

EX:#include <stdio.h>

int binary\_search(int arr[], int n, int x ) {

int low = 0;

int high = n - 1;

while (low <= high) {

int mid = (low + high) / 2;

if (arr[mid] == x) {

return mid;

}

else if (x>arr[mid]) {

low = mid + 1;

}

else {

high = mid - 1;

}

}

return -1;

}

int main() {

int arr[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

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

int x = 7;

int result = binary\_search(arr, n, x);

if (result != -1)

{

printf("Element %d is found at index %d.\n", x, result);

} else

{

printf("Element %d is not found in the array.\n", x);

}

return 0;

}

2)Recursive Method:it follows divide and conqure approach

EX:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

T(n)=k1+k2+k3+k4+k5

=n^0

=O(1)

s=d/T

T=d/s 250kb/20kb

T(n)=d/n=n'\*d=O(n')

Time complexity:The time taken by the program to run.

Space complexity:The space complexity of a program is the amount ofbmemory needs

to run the program.

Sorting Techinques

\*Bubble sort

\*selection

\*insert

\*merge

\*Heap

\*Quick

Types of sorting:

1)space used

\*In place:sorting algorithms which does not require any extra space for sorting

EX:Bubble sort

\*out of place:sorting algorithms which require any extra space for sorting

Ex:Merge sort

2)Stability

\*stable:If a sorting algorithm after sorting the contents does not change the sequence

of similar content in which they appear,then this sorting is calledwe

\*unstable:If a sorting algorithm after sorting the contents change the sequence

of similar content in which they appear,then this sorting is called

1)Bubble sort:

\*it is also called as sinking sort

\*we repeatedly compare each pair of adjacent items and swap them if they

are in the wrong order

def bubblesort(arr):

n=len(arr)

swapped=False

for i in range(n-1):

for index in range(0,n-i-1):

if arr[index]>arr[index+1]:

swapped=True

arr[index],arr[index+1]=arr[index+1],arr[index]

if not swapped:

return

arr=[64,34,25,12,22,11,90]

bubblesort(arr)

print("sorted array is:")

EX:#include<stdio.h>

void bubble(int [],int);

int swap(int,int,int);

int main()

{

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

int n=7;

bubble(arr,n);

//printf("%d",value);

}

void bubble(int arr[],int n)

{

int i,j,temp;

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

{

for(j=0;j<n-i-1;j++)

{

if(arr[j]>arr[j+1])

{

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

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

{

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

}

}

2)Selection sort:

\*in case of selection sort we repeatedly find the maximum element and move it to the sorted part of array

to make unsorted part sorted

#include<stdio.h>

void selection(int arr[], int n);

void swap(int \*a, int \*b);

int main() {

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

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

selection(arr, n);

printf("Sorted array: \n");

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

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

}

printf("\n");

return 0;

}

void selection(int arr[], int n)

{

int i, j, min\_index;

for(i = 0; i < n-1; i++)

{

min\_index = i;

for(j = i+1; j < n-1; j++)

{

if(arr[min\_index] >arr[j])

{

min\_index = j;

}

if(min\_index != i) {

swap(&arr[i], &arr[min\_index]);

}

}

}

}

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

3)Insertion sort:

\*Divide the given array into two part

#include<stdio.h>

void insertion(int arr[], int n);

int main() {

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

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

insertion(arr, n);

printf("insetion array: \n");

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

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

}

printf("\n");

return 0;

}

void insertion(int arr[], int n)

{

int i, j,key;

for(i = 1; i < n; i++)

{

key=arr[i];

j=i-1;

while(j >=0 && key<arr[j] )

{

arr[j+1]=arr[j];

j-=1;

}

arr[j+1]=key;

}

}