**DAY -18(21-11-2024)**

* Multithreading mostly used producer and consumer, here consumer is read it and producer is write it
* **Question**

**1) circular queue**

**2)Priority queue for objective**

* **SEARCHING ALGORITHM:**
* Algorithms is a set of instructions
* They are 2 types of searchs:

1. Linear or sequence search
2. Binary search

* Linear search:
* Linear search is a time consuming
* advantage it is used to search the unsorted elements
* Disadvantage is time is more (time complexity) i.e O(N)
* It is linear in manner(it starts at starting and ends at last)
* Linearsearch(arr[],key)

{

For(int i=0;i<n;i++)

{

If(arr[i]==key)

{

Printf(“key found”);

}

} return index;

}

* Binary search:
* It reduces the number of iterations
* It can be do it in the sorted list
* They repeat that it targeted on the center of the search structure and divide the search space in half
* Binary search is an efficient algorithm for finding an element in a sorted array. It works by repeatedly dividing the search interval in half. If the value of the search key is less than the item in the middle of the interval, the algorithm narrows the interval to the lower half. Otherwise, it narrows it to the upper half. This process continues until the value is found or the interval is empty.
* It can be implemented in 2 ways :

1. Iterative method
2. Recursive method

* Recursive method follows the divide and conquer rule
* Take low(starting) and high(end)
* Mid=Arr[(low+high)/2]
* X==arr[mid] then return mid
* X>arr[mid] compare x with the right side of the mid this is done by setting low to low=mid+1
* X<arr[mid] compare x with the left side of the mid .this is done by setting high=mid-1
* If low and high is meet then it means key is not found
* Do until low is equal to high

mid = (low +high)/2

if ( x== arr[mid])

  return mid

elif( x > arr[mid])

  low = mid + 1

else

  high = mid -1

program for iterative :

#include <stdio.h>

int binary(int arr[],int key,int low,int high){  
    while(low <= high){  
        int mid = (low + high)/2;  
        if(key == arr[mid])  
        return mid;  
        else if(key > arr[mid])  
        low = mid + 1;  
        else   
        high = mid - 1;  
    }  
    return 0;  
}

int main() {  
    int arr[5] = {1,2,3,4,5};  
    int n = 5;  
    int flag = binary(arr,10,0,n);  
    if(flag == 0)  
    printf("key not found: %d\n",flag);  
    else  
    printf("key found: %d\n",flag);  
    return 0;  
}

**program for binary search using recursive:**

#include <stdio.h>

int binary(int arr[],int key,int low,int high){

if(low>high)

{

return 0;

}

else{

int mid = (low + high)/2;

if(key == arr[mid])

return mid;

else if(key > arr[mid])

return binary(arr,key,mid + 1,high);

else

return binary(arr,key,low,mid - 1);

}

return 0;

}

int main() {

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

int n = 5;

int flag = binary(arr,5,0,n);

if(flag == 0)

printf("key not found: %d\n",flag);

else

printf("key found: %d\n",flag);

return 0;

}

SORTING ALGORITHM:

* A data structure is of two types:

1. Linear🡺arrays,stack,queue
2. Non-linear 🡺tree, graphs

* 1)Primitive data structures==> int,char,float,pointers

2)Non-primitive data structure🡺arrays, lists

Lists🡺1) linear list->Stack, queue

2)Nonlinear list ->tree, graphs

* Practical algorithm design issues:
* Time complexity
* Space complexity
* Amount of memory its need to run to completion
* Instruction space, data space, environment space are the components for the space complexity
* BIG O notations:
* T(n)=k1\*(n^0) +k2+k3+k4

= n^0

=O (1)

* Speed=distance/time

T(n)=d/s=d/n=n’\*d

=O(n’)

O(n) is used in looping statements

* Half the value🡺log(n)
* O(n^2) 🡺two looping statements
* Types of sorting

1. Space 🡺in place, out of place

* In place sorting🡪 which does not require the extra space

Eg: bubble sort

* Out of place🡪 requires the extra space

Eg: merge sort (divide and conquer rule, unsorted list)

1. Stability🡺 stable, unstable

* Stable🡪 after sort the position does not changes or after sort the contents does not change the sequence of the similar contents in which they appear.

Eg: insertion sort

* Unstable🡪 after sorting the position is changes

Eg: quick sort

* + BUBBLE SORT:
* Bubble sort is referred to the sinking sort
* We repeatedly compare each pair of adjacent items and swap them if they are in wrong order
* Eg: 5,9,3,1,2,8,4,7,6
* In 1st iteration it compare 5 with 9 ,9 with 3 it swap(5,3,9,1,2,8,4,7,6 )and 9 with 1 (9>1) swap it and after 1 st iteration 9 value will be at the last position in the array
* From next iteration itself we don’t consider last element i.e 9
* In the first iteration of increasing order, the last element in the correct position
* We used two looping statements

Program :

#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]);

}

}

Program for bubble sort using swap function:

* Refer the selection for swap function
  + SELECTION SORT:
* In case of selection sort we repeatedly find the minimum element and move it to the sorted array to make unsorted part sorted
* Sorted array| unsorted array
* Eg : 5,7,4,3,8,1,6,5,9,2
* Here check the minimum value which is 1 then exchange it with first element in the array
* In 1st iteration the first element got sorted (left side)
* Program :

#include<stdio.h>

void selection(int [],int);

int main()

{

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

int n=7;

selection(arr,n);

}

void swap(int arr[],int i,int j)

{

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

void selection(int arr[],int n)

{

int i,j,min\_index,temp;

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;

}

}

swap(arr,i,min\_index);

}

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

{

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

}

}

Program of selection sort using flags (because without flag the swap is being called multiple times):

#include<stdio.h>

void selection(int [],int);

int main()

{

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

int n=7;

selection(arr,n);

}

void swap(int arr[],int i,int j)

{

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

void selection(int arr[],int n)

{

int flag=0;

int i,j,min\_index,temp;

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

{

min\_index=i;

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

{

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

{

min\_index=j;

flag=1;

}

}

if(flag==1)

{

swap(arr,i,min\_index);

}

}

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

{

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

}

}

* + INSERTION SORT:
* Insertion sort is a straightforward sorting algorithm that builds a sorted array one element at a time by comparing each new element to the already sorted elements and inserting it in the correct position.
* Eg : 5,3,4,7,2,8,6,9,1
* Here the 5 is placed in sorted array and compare the single element i.e 3 (which is present in the unsorted array) with 5 if the 3 less than 5 then insert in that position
* When we have insufficient memory we use insertion sort
* When we have the continuous inflow of numbers and we want to sort
* The time complexity is O(N^2)
* Program :

#include<stdio.h>

void insertion(int [],int);

int main()

{

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

int n=7;

insertion(arr,n);

}

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=j-1;

}

arr[j+1]=key;

}

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

{

Printf(“sorted array”);

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

}

}

* + MERGE SORT:
* It is a divide and conquer rule
* Divide the input array in 2 halves and we keep them recursively until the array cannot be divided
* Question : write a program in c to merge two arrays or list to form a sorted list