Important : Producer and consumer in multithreading

Circular queue and priority queue

SEARCHING ALGORITHMS

It is to check or retrieve an element

1. Linear search (Sequential) – It can be implemented for any unsorted search
2. Binary search (Interval) --- The list should be sorted …search in 1st half and 2nd half --- efficient one

Linear search

* It checks one element by element and goes on until the desired element is reached
* liner search(array, key)

for each item in the array

if item

Binary search

We have two types:

1. iterative
2. recursive

It follows divide and conquer approach

* find the middle element mid of the array 🡺 [(low+high]/2]
* If x==mid then return mid else compare the element to be searched with m
* If x>mid, compare x with mid element

Iteration method

Do until the pointers low and high meet each other

Mid = (low+high)/2

If(x==arr[mid])

Return mid

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

Low = mid+1

Else // x is on left side

High = mid-1

Now doing it using dynamic memory allocation

BinarySearch(arr,x,low,high)

SORTING

Three basic design goals:

1. Time complexity

T(n) = k1\*(n^0)+k2+k3+k4+k5

= n(0)

= O(1)

S=d/t

T=d/s 250kb/20kb // ex

**T(n)=d/n=n` \*d = O(n`)**

**Where n` = 1/n**

1. Space complexity

Sorting types:

1. Bubble sort
2. Merge sort
3. Insertion
4. Quick

Sorting

Space used Stability

In place Out place Stable Unstable

Ex : Bubble Ex : Merge Ex : Insertion Ex : Quick

* In place does not require any extra space for sorting
* Out place requires an extra space for sorting
* In stable sorting the position of the container must not change

70 10 80 40 50 40 20

10 20 40 40 50 70 80

* In unstable sorting the position of the container change

1. 10 80 40 50 40 20

10 20 40 40 50 70 80

BUBBLE SORT

* It is also known as Sinking sort
* We repeatedly compare each pair of adjacent items and swap them if they are in the wrong order
* If the first iteration is in the increasing order then last element(MAX) will be in right position

To swap two elements in an array using bubble sort

#include <stdio.h>

// Function to swap two elements

void swap(int \*x, int \*y) {

int temp = \*x;

\*x = \*y;

\*y = temp;

}

// Bubble Sort function

void bubbleSort(int arr[], int n) {

// Outer loop for number of passes

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

// Inner loop for each pass comparison

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

// Swap if the element is greater than the next element

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

swap(&arr[j], &arr[j + 1]);

}

}

}

}

// Function to print the array

gvoid printArray(int arr[], int size) {

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

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

}

printf("\n");

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

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

printf("Original array: \n");

printArray(arr, n);

// Call bubble sort

bubbleSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

Bubble will sort from right to left….Selection will sort from left to right

WAP to swap two numbers in an array using selection sort