LEARN DSA WITH C++

WEEK :: 03

LEARN DSA WITH C++

PDF

COURSE INSTRUCTOR BY
ROHIT NEGI

MADE BY-
PRADUM SINGHA

Check Profile

My profile

LEARN DSA WITH C++

WEEK :: 03 **DAY: 01** **DATE: 01-05-2023**

TIME & SPACE COMPLEXITY

It is defined as the time taken by the algorithm as our input size.

Time Complexity ⇒ Input Size :: The time complexity depends on the size of our input. Then we can compare our different algorithms to see which one is better.

Time Complexity != Time Taken

O(Worst Case) :: It indicates the maximum time required by an algorithm for all input values. It represents the worst case of an algorithm's time complexity.

Omega(Best Case) :: It indicates the minimum time required by an algorithm for all input values. It represents the best case of an algorithm's time complexity.

Theta(Average Case) ::It indicates the average bound of an algorithm. It represents the average case of an algorithm's time complexity.

Exp :: 01:: Print n natural numbers?

for(int i=0; i<n; i++)
{
cout<< i<<" ";
 	
return 0;
}
	Worst Case: 10 num print O(n)
	
	Best Case : 10 num print Ω(n)
	
	Avg Case: 10 num print θ(n)

Exp :: 02:: Find num in Array {1, 6, 85, 8, 8, 5, }

for(int i=0; i<n; i++)
{
if(arr[i]==Target)
{
cout<<"present";
break;
}
}
	Best Case:If we find the Target for the first time in a loop. Ω(1).
	
	Worst Case: If We find Target last time in a loop. O(n);
	
	Avg Case : If We can find the middle array in a loop. θ(n/2)= θ(n) . [1/2 ignore.]

How to choose Algorithm::

We choose any algorithm according to Time and Complexity which algorithm needs less.

#For Nested Loop:

for(int i=0; i<n; i++)
{
  for(int j=0; j<+100; j++)
  cout<<....;
}
	Print 100 time
	i=1 ⇒ j=100
	i=2 ⇒ j=100
	i=n ⇒ j=100
	(Sum all execute output)
	
	TC = O(100n) = O(n)

Sum of n num:

Algorithm 1	Algorithm 2

for(int i=0; i<= n; i++)
{
  Sum = sum + i;
} [execute n time for n output] (n==n)
	Sum = n(n+1)/2
	
	[execute one time for any output] (1==n)

O(n)	O

for(int i=1; i<=10; i++)
{
  cout<<"Hello CoderArmy";
}
	Print 10 time not depend input(n)
	
	TC= O(1)
	
	If change (i<=n):: now depend on input
	TC = O(n)

for(int i=1; i<=n; i++) { for(int j=1; j<=n; j++) cout<<" Print"; } }	i=1; j=n; (print) i=2; j=n; (print) i=3; j=n; (print) i=n; j=n; (print) n+n+n+....+n=n(1+1+1+...+1)=n*n=n^2 TC= O(n^2)

```
for(int i=1; i<=n; j++)
{  for(int j=1; j<=i; j++)
       cout<<"Print";
}
```

Print i=1; j=1
i=2; j=2
i=3; j=3
i=4; j=4
i=n; j=n
1+2+3+4+....+n=n(n+1)/2=(n^2 +n)/2
TC = O(n^2) 2 (constant)==> Ignore
Always take big power value

Prime num:

```
for(int =i=0=2; i<=n; i++0)
{  if(num%i==0)
   {   cout<<"Not Prime";
       Break;
   }  }
```

Worst Case: O(n) loop continue n time

Best Case: Ω(1) loop break in first loop

Avg Case: θ(n)

```
for(int i=1; i<=n; i++)
{  for(int j=01; j<=j^2; j++)
      cout<<"Concept Chamka";
}
```

Print output::
i=1; j=1 to 1;
i=2; j=1 to 4;
i=3; j=1 to 9; Big power value
i=n j= 1 to n^2
[1^2 + 2^2 + 3^2 + 4^2 +.....+n^2
=[n(n+1)(2n+1)/6]= 2n^3 + n^2 + n = n^3

```
For (int i=1; i<=n; i++)
{   for(j=1; j<=m; j++)
      cout<<"Chamka";
}
```

Print input
i=1; j= 1 to m;
i=2; j= 1 to m;
i=3; j= 1 to m;
i=4; j= 1 to m;
i=n; j= 1 to m;
[m+m+m+m+.....+m] = [m(1+1+1+....+1)]= m(n)=m*n
TC = O(n*m)

```
for(int i=1; i<=n; i++)
{ for ( int j=1; j<=j^2; j++)
 { for(int k=1; k<=n/2; k++)
    cout<<"Print";
 } }
```

Print input::
i=1; j= 1 to 1; k=n/2
i=2; j= 1 to 4; k=4* n/2=2n
i=3; j= 1 to 9; k=9* n/2=9n/2
i=4; j= 1 to 16; k=n/2= 16* n/2 =16n/2
i=n; j= 1 to n^2; k=n/2 = n^2 *n/2
[Sum all input]
n/2[1 + 2^2 + 3^2+...+n^2]= [n/2* n(n+1)(2n+1)/6]
TC = O(n^4)

```
for(int i=1; i<n; i=i*2)
{
    cout<<"Print";
}
```

Print input ::
i=1; "print"
i=2; "print" i=i*2
i=4; "print"
i=8; "print"
i=n; "print" :: i= 1 2 4 8 - - - n
2^0 2^1 2^2 2^3 2^k [k=k+1]
n = 2^k => log n = log2 ^k => log n = k*log 2
k=log n/ lg 2= $\log_2 n$ TC = O($\log_2 n$)

```
For (int i=n/2; ;i<=n; i++)
 { for(int j=1; j<=n/2; j++)
   { for(int k=1; k<=n; k++)
       cout<<"print";
   }
}                          if(k*2)
```

1st loop round → n/2
2nd loop round → n/2
3rd loop round → n { i, j & k not depend on each other}

TC = n/2 * n/2 * n = n^3/8 = O(n^3)
So, TC = n/2 * n/2 * $\log_2 n$ = n^2 log n =O(n^2 log n)

```
for(int i=n/2; i<=n; i++)
{  for(int j=1; j<=n; j=2*j)
  {  for(int k=1; k<=n; k=k*2)
      cout<<" Print";
  }
}
```

— n/2
— log2
— log2

n/2 * logn * log n = n/2(log n)^2 = O(n(log n)^2)

```
for(int j=1; i<=n; i++)
{ for(int j=1; j<=n; j=j+i;
   cout<<" print";
}
```

i=1; j = 1 to n; print n time
i=2; j = 1 to n; print n/2 [jump 2 number]
i=3; j = 1 to n; print n/3 [jump 3 number]
i=4; j = 1 to n; print n/4 [jump 4 number]
i=n; j = 1 to n; print n/n [jump n number]
Sum = (n + n/2 + n/3 + n/4+..... +n/n)
n[1 + ½ + ⅓ + ¼ + ... + 1/n] = lne n {Harmonic Series}
TC = (n loge n)

LEARN DSA WITH C++

WEEK :: 03 **DAY: 02** **DATE: 02-05-2023**

TIME & SPACE COMPLEXITY PART 2

Time Complexity: It states that the amount of time taken by an algorithm to run as a function of the length of the input.

Choose a better algorithm: 1. O(log10 (x)) 2. O(√n)

Put the value of n = Very high value(small value to high value).

log2N=k ⇒ 2 N = k ⇒ 2^k = N; || log2 8 = 2^x = 8⇒ x = 3;

n	**=**	**10**	**100**	**10000**
logn=		**1**	**2**	**4**
√n	**=**	**3**	**10**	**100**

It is better.

Better Time complexity algorithm : Table

O(1), O(log n), O(√n), O(n), O(n log n), O(n2), O(n^3), O(2^n), O(n!)

Space Complexity :: It states that the amount of space taken by an algorithm to run a function of the length of input.

1. **Auxiliary space.**
2. **Total space complexity.**

Auxiliary Space	Total space Complexity
It depends on the size of input but does not include the presence of input space which already takes space computer memory. Ignore :: Geven Input size space.	It depends on both spaces which are obtained from input.

 # Print 1 to n::

```
int n;
cin>>n;
for(int i=1; i<=10; i++)
        cout<<I<<" ";
```

Which one takes space::
here, n & i take space.
So, space complexity O(1).
Not depend on input.

 # Print Array input: {2, 4, 6, 7}; output: {20, 40, 60, 70};

```
#include<iostream>
using namespace std;

int main()
{
    int n;
```

```
    cout<<"Enter Input Size: ";
    cin>>n;
    cout<<"Enter the value of array:\n";
    int Input_arr[n];
    cin>>Input_arr[n];

    for(int i=0; i<n; i++)
    cin>>Input_arr[i];

    int arr[n];
    for(int i=0; i<n; i++)
    arr[i] = Input_arr[i]*10;

    for(int i=0; i<n; i++)
    cout<<arr[i]<<" ";

return 0;
};
```

Ignore the input loop.

Time complexity::
First loop = n
Second loop = n

=n + n = 2n = O(n)

Space Complexity ::

Total space Complexity :
Input loop = 1 + n
Array copy loop = n +1
Print loop = n + 1

= 1 + n + n +1 + n + 1 = O(n)

Auxiliary Space = O(n)

LEARN DSA WITH C++

WEEK :: 03 **DAY: 03** **DATE: 03-05-2023**

BINARY SEARCH

Binary Search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half.

Given array :: arr[ ] = { 3, 7, 10, 14, 17, 18, 25} **Find**: 7

	3	7	10	14	17	18	25
Index	0	1	2	3	4	5	6

First, We will go to the middle of the array. (0 + 6)/2 = 3

3	7	10	14	17	18	25

Check 7 & 14 is equal or not. Then check the left side.

3	7	10

Again go to the middle. And find 7

7

Then it's equal to 7.

Example02 ::

3	10	17	18	24	29	32	37	50
Index 0	1	2	3	4	5	6	7	8

Target : 3

We go to the middle of the array.

3	10	17	18	24	29	32	37	50

Check target & 24 is equal or not. (24>3) find left side

3	10	17	18

Again, find the middle point.

3	10

Check target & 10 is equal or not. (10>3) find left side

3

Finally we get a target.

Binary Search Algorithm:-

1. **Find the middle element (Index)**
2. **Check if the value are equal or not**
    a. **Both are equal [target = arr[mid]**
        cout<<"Find Target";
    b. **Target value > arr[mid]**
        Move to right side
    c. **Target value < arr[mid]**
        Move to left side

#Code::

```cpp
#include <iostream>
using namespace std;

int main()
{
    int nums[8] = {5, 6, 8, 9, 11, 15, 17, 21};

    int mid, target = 15, start = 0, end = 7;

    while (start <= end)
    {
        mid = (start + end) / 2;
        if (nums[mid] == target)
        {
            cout << mid;
            return 0;
        }
        else if (nums[mid] > target)
            end = mid - 1;
        else
            start = mid + 1;
    }
    cout << "-1";
    return 0;
};
```

Time Complexity :

It's going to be half by half. n/2 -> n/4 -> n/8 O(log2 n)

Space Complexity:

O(1) {Auxiliary} , O(n) {Total space complexity}

Exp:: 03:: arr[6] = { 1, 2, 2, 2, 4, 5, 8, 11 }; **Target** = 2 [left most and right most index]

```cpp
#include <iostream>
using namespace std;

int main()
{
    int v[8] = { 1, 2, 2, 2, 4, 5, 8, 11 }; // arr = v
    int x = 2;    // target

    long long mid, start = 0, end = 7, left_index, right_index;

  while (start <= end)
    {
        mid = (start + end) / 2;
        if (v[mid] == x)
        {
            left_index = mid;
            end = mid - 1;
        }
        else if (v[mid] > x)
            end = mid - 1;
        else
            start = mid + 1;
    }

    start = 0;
    end = 7;

    while (start <= end)
    {
        mid = (start + end) / 2;
        if (v[mid] == x)
        {
            right_index = mid;
            start = mid + 1;
        }
        else if (v[mid] > x)
            end = mid - 1;
        else
            start = mid + 1;
    }
```

```
    cout<<"left_index: "<<left_index<<" "<<"right_index:
"<<right_index;
    return 0;
};
```

Exp:: 03:: Search Insert Position << LeetCode >>

```
class Solution {
public:
    int searchInsert(vector<int>& arr, int target) {

        int mid, start = 0, end = arr.size() - 1, index;

        while(start<=end)
        {
            mid = (start + end)/2;
            if(arr[mid]==target)
            return mid;
            else if (arr[mid]<target)
            {
                start = mid+1;
                index = mid+1;
            }
            else
            {
                end = mid -1;
                index = mid;
            }
        }
        return index;
    }
};
```

Exp:: 04:: Peak Index in a Mountain Array << LeetCode >>

```
class Solution {
public:
    int peakIndexInMountainArray(vector<int>& arr) {
       int start =0, mid, end = arr.size()-1;

       while(start<=end)
       {
           mid = end + (start-end)/2;
           if(arr[mid]>arr[mid+1] && arr[mid]> arr[mid-1])
           return mid;
```

```
            else if(arr[mid]>arr[mid-1] && arr[mid]<arr[mid+1])
            start = mid+1;
            else
            end = mid-1;
        }
        return -1;
    }
};
```

Own Code ::

```
#include <iostream>
using namespace std;

int main()
{
    int arr[5] = {5, 6, 5, 4, 1};
    int start = 0, mid, end = 4;

    while (start <= end)
    {
        mid = end + (start - end) / 2;
        if (arr[mid] > arr[mid + 1] && arr[mid] > arr[mid - 1])
        {
            cout<< mid;
            return 0;
        }
        else if (arr[mid] > arr[mid - 1] && arr[mid] < arr[mid + 1])
            start = mid + 1;
        else
            end = mid - 1;
    }

    return 0;
};
```

LEARN DSA WITH C++

WEEK :: 03 **DAY: 04** **DATE: 04-05-2023**

BINARY SEARCH IN DETAIL

Explain :: Arr[7] = { 2, 3, 6, 7, 5, 1, 0 }; [Find index of pick value]

Arr[7] = { 2, 3, 6, 7, 5, 1, 0 }
index = 0 1 2 3 4 5 6

Pick element of Arr 7; And index = 3;
Satisfy condition = Arr[i-1] < Arr[i] > Arr[i+1] 6 < 7 > 5

Increasing order : Arr[i-1] < Arr[i] < arr [i+1] = find right side :- start = mid+1

Decreasing order : Arr[i+1] > Arr[i] > arr[i+1]= find left side : - end = mid -1

Code :: Last day Quation

Exp:: 01 :: Find missing element

arr	0	1	3	4	5
Index	0	1	2	3	4

1st, go middle of arr = 3 Check [index == element]
Miss match [index != element] ,then go left side but store index because if it is a small miss match element.

arr	0	1	2	5	8
Index	0	1	2	3	4

1st, go middle of arr = 3 Check [index == element]
Not Miss match [index == element] ,then go right side but store index because if it is a small miss match element.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int n;
    cin >> n;
    int arr[n];
```

```
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    int start = 0, end = n - 1, mid, index;

    while (start <= end)
    {

        mid = (start + end) / 2;

        if (mid < arr[mid])
            end = mid - 1;

        else
        {
            index = mid + 1;
            start = mid + 1;
        }
    }
    cout << index;

    return 0;
};
```

Exp:: 01 :: Find how many times rotate the arr. Arr[6] = { 4, 5, 6, 7, 2, 3 };

Explain :: [Pivot Method]

Coside pivot value = 4;

1st, go mid = 6; compare with pivot value

arr[mid] > pivot value :: store index and go right side :: start = mid +1

arr[mid] < pivot value :: store index and go left side :: start = mid - 1

<< Rotation >>

```
//User function template for C++
class Solution{
public:
      int findKRotation(int arr[], int n) {
          // code here
           if (arr[0] < arr[n - 1])
          return 0;

        int low = 0, high = n - 1;
        while (low < high)
        {
            int mid = (low + high)/2;
            if (arr[low] <= arr[mid] && arr[high] <= arr[mid])
            {
                low = mid + 1;
            }
            else
            {
```

```
                high = mid;
            }
        }
        return low;
    }

}
```

Exp:: 02 ::Search in a Rotated Array **<< GEEKSFORGEEKS >>**

Arr[6] = {5, 6, 8, 9, 1, 2}; **key =**

Index = 0 1 2 3 4 5

1st :: pivot arr[0] = 5;

2nd :: go mid = 8;

if(arr[mid] == key)

Return mid;

3rd:: compare arr[mid] != pivot (arr[0])

Case 1:: pivot < key end = mid -1;

Case 2:: arr[mid]] < key

Start = mid +1

4th :: arr[mid]<pivot

But pivot < key end=mid -1;

Case 1:: key<arr[mid] end = mid-1

Other case start = mid +1

Next Question

LEARN DSA WITH C++

WEEK :: 03 **DAY: 04** **DATE: 06-05-2023**

BINARY SEARCH IN DETAIL PART II

5	6	7	8	9	0	1	2	3	4

Target = 3

Index = 0 1 2 3 4 5 6 7 8 9

Find middle (0 + 9)/2
check key == arr[middle] return middle
arr[start <= arr[middle] arr sorted
5 <= target <= 9 Not alive, move right
again find middle 2,
check target alive between 1 & 2, Not - move right
again find middle check middle==target and print

1. if(arr[mid] == key)
   return mid;
2. else if (arr[start] < arr[mid])
   {
   if(key>= arr[start] && key<= arr[mid])
   End = mid - 1;
   else
   start = mid + 1;
3. else
   {
   If (key > arr[mid] && key<= arr[end]
   start = mid +1;
   }

```
class Solution{
    public:
    int search(int arr[], int l, int h, int key){
        // l: The starting index
        // h: The ending index, you have to search the key in
this range

        //complete the function here

        int mid, start =0, end=h, index=-1;

        while(start<=end)
        {
            mid = end+(start-end)/2;

            if(arr[mid]==key)
            return mid;
```

```
            else if(arr[mid]>arr[start])
            {
                if(arr[mid]>key && arr[start]<=key)
                end = mid -1;
                else
                start = mid +1;
            }
            else
            {
                if(arr[mid]<key && key<=arr[end])
                start = mid +1;
                else
                end = mid -1;
            }
        }
        return index;
    }
};
```

Allocate minimum number of pages << GEEKSFORGEEKS >>

```
class Solution
{
    public:
    //Function to find minimum number of pages.
    int findPages(int arr[], int n, int m)
    {
        //code here
        int start=arr[0], end=0, mid;
        if(m>n)
        return -1;

        for(int i=0; i<n; i++)
        {
            end = end+arr[i];
            if(start<arr[i])
            start = arr[i];
        }

        int ans;
        while(start<=end)
        {
            mid =(start+end)/2;
            int sum=0, count=1;
            for(int i=0; i<n; i++)
            {
                sum+=arr[i];
                if(sum>mid)
                {
                    count++;
                    sum = arr[i];
                }
            }
            if(count<=m)
```

```
            {
                end = mid-1;
                ans = mid;
            }
            else
            start = mid+1;
        }
        return ans;
    }
}
```

Aggressive Cows << GEEKSFORGEEKS >>

```
// User function Template for C++

class Solution {
public:

    int solve(int n, int k, vector<int> &stalls) {
        // Write your code here
        int start =1, end = stalls[0], mid, minimum = stalls[0];

        for(int i=1; i<n; i++)
        {
            minimum = min(minimum, stalls[i]);
            end = max(end, stalls[i]);
        };

        sort(stalls.begin(), stalls.end());
        end -= minimum;

        int ans;
        while(start<=end)
        {
            mid =(start+end)/2;
            int lastPos = stalls[0], count = 1;
            for(int i=0; i<n; i++)
            {
                if(stalls[i]-lastPos>=mid)
                {
                    count++;
                    lastPos = stalls[i];
                }
                if(count==k)
                break;
            }

            if(count==k)
            {
                ans = mid;
                start = mid +1;
            }
            else
            end = mid-1;
        }
        return ans;
    }
};
```