Linear Search

Study Point

- Suppose a linear array DATA contains n elements and suppose a specific ITEM of information is given. We want either to find the location LOC of ITEM in the array DATA, or to send some message, such as **LOC = 0**, to indicate that ITEM **does not appear** in DATA.
- The linear search algorithm solves this problem by comparing ITEM, one by one, with each element in DATA. That is, we compare ITEM with DATA[1], then DATA[2], and so on, until we find LOC such that ITEM = DATA[LOC].

Presented By:
Sheikh Muhammad Aamir

Algorithm:

(Linear Search) A linear array DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in the array DATA or sets LOC = 0.

Study Point

1. [Initialize] Set K := 1 and LOC := 0.
2. Repeat Steps 3 and 4 while LOC = 0 and K <= N.
3. If ITEM = DATA[K], then: Set LOC: =K.
4. Set K := K + 1. [Increments counter.]
   [End of Step 2 loop.]

1	2	3	4	5
10	25	15	22	17

K = 1, N=5 LOC = 0

ITEM = 15 (to be searched)

5. [Successful?]
   If LOC = 0, then:
      Write: ITEM is not in the array DATA.
   Else:
      Write: LOC is the location of ITEM.
   [End of If structure.]
6. Exit.

Presented By:
Sheikh Muhammad Aamir

Algorithm:

(Linear Search) A linear array DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of repeated ITEM in the array DATA or sets LOC = 0.

Study Point

1. [Initialize] Set K := 1 and LOC := 0.
2. Repeat Steps 3 and 4 while ~~LOC = 0 and~~ K <= N.
3. If ITEM = DATA[K], then:
   i. Set LOC: =K
   ii. Write: "Element found at location", LOC

1. Set K := K + 1. [Increments counter.]
   [End of Step 2 loop.]

1	2	3	4	5
10	25	15	22	15

K = 1, N=5 LOC = 0

ITEM = 15 (to be searched)

5. [Successful?]
   If LOC = 0, then:
   Write: ITEM is not in the array DATA.
   ~~Else:~~
   ~~Write: LOC is the location of ITEM.~~
   [End of If structure.]
6. Exit.

Presented By:
Sheikh Muhammad Aamir

Average Case

Study Point

"the expected value of *f(n)*"

- The analysis of the average case assumes a certain **probabilistic distribution for the input data**; one such **assumption** might be that all possible permutations of an input data set are **equally likely**.
- The average case also uses the following concept in probability theory.
- Suppose the numbers occur with respective probabilities $p_1, p_2, \ldots p_k$.
- Then the *expectation* or ***average value E*** is given by

$$\mathbf{E = n_1 p_1 + n_2 p_2 + \ldots + n_k P_k}$$

Presented By:
Sheikh Muhammad Aamir

Linear Search

Analysis of Linear Search Algorithm

Study Point

Worst Case

Clearly the worst case occurs when ITEM is the last element in the array DATA or is not there at all. In either situation, we have

$$C(n) = n$$

Accordingly, $C(n) = n$ is the worst-case complexity of the linear search algorithm.

Presented By:
Sheikh Muhammad Aamir

Analysis of Linear Search Algorithm...

Study Point

Average Case

Here we assume that ITEM does appear in DATA, and that it is equally likely to occur at any position in the array.

Accordingly, the number of comparisons can be any of the numbers **1, 2, 3, ..., n**, and each number occurs with probability $p = 1/n$. Then

$$C(n) = 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + \cdots + n \cdot \frac{1}{n}$$

$$= (1 + 2 + \cdots + n) \cdot \frac{1}{n}$$

$$= \frac{n(n+1)}{2} \cdot \frac{1}{n} = \frac{n+1}{2}$$

Presented By:
Sheikh Muhammad Aamir

$$= \frac{n(n+1)}{2} \cdot \frac{1}{n} = \frac{n+1}{2}$$

Presented By:
Sheikh Muhammad Aamir

Asymptotic Notations

Study Point

Asymptotic notations *are expressions that are used to show the complexity of an algorithm.*

In complexity, we **analyze** the algorithm's **efficiency** by measuring the running time and space

Presented By:

Types of analysis

Study Point

Best Case:

Performance of an algorithm is evaluated for the input, for which algorithm takes ***small amount of time or space***

Worst Case:

Performance of an algorithm is evaluated for the input, for which algorithm takes ***large amount of time or space***

Average Case:

Performance of an algorithm is evaluated for the input, for which algorithm takes time or space that lies in between above-mentioned cases

Asymptotic Notations

Study Point

Commonly asymptotic notations which are used to represent the complexity of an algorithm are as follows:

O	*(Big O)*	*to express worst case*
Ω	*(Big Omega)*	*to express best case*
Θ	*(Theta)*	*for both cases*

Presented By:
Sheikh Muhammad Aamir

39

Rate of growth; Big O notation

Study Point

- Suppose ***M*** is an algorithm, and suppose ***n*** is the size of the input data.
- The complexity ***f(n)*** of ***M*** increases as ***n*** increases
- It is usually the **rate of increase** *of* ***f(n)*** that we want to examine. This is done by comparing ***f(n)*** with some standard functions, such as

$\log_2 n$, n, $n \log_2 n$, n^2 n^3 2^n

Presented By:
Sheikh Muhammad Aamir

40

Typical Algorithm Complexities

Study Point

Complexity	Running Time
Constant	O (1)
Linear	O (N)
Logarithmic	O (Log(N))
Quadratic	O (n^2)
Cubic	O (n^3)
Exponential	O (2^n), O(N!), O (n^k) ...

Presented By:
Sheikh Muhammad Aamir

Rate of growth; Big O notation..

Study Point

- One way to compare the function ***f(n)*** with these standard functions is to use the functional *O* notation defined as follows:

$$f(n) = O(g(n))$$

Which is read as "f(n) is of order g(n)"

Example: for any polynomial of degree m

$P(n) = O(n^m)$

i.e. $8n^3 - 576n^2 + 832n - 248 = O(n^3)$

Presented By:
Sheikh Muhammad Aamir

Rate of growth; Big O notation...

Study Point

n \ g(n)	log n	n	n log n	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	10^3	10^4	10^6	10^9	10^{300}

Rate of Growth of Standard Functions

Presented By:
Sheikh Muhammad Aamir

Other Asymptotic Notations for

Study Point

Complexity in respect of Running-Time

Study Point

- Let the series is 1, 2, …, n and we have to find the sum of cubic series i.e. sum = 1 + 8 + 27 …

Example:

Algo / Program

```
1. sum = 0;
2. for( i=1 ; i<=n; i++ )
3.   sum = sum + i * i * i;
4. return sum;
```

Presented By:
Sheikh Muhammad Aamir

Complexity in respect of Running-Time...

Study Point

- Let the running-time for single action is 1

Now

- **sum = 0** // has **1** running time
- **i = 1** // has **1** running time
- **i<=n** // has **n+1** running time as loop terminated on false condition
- **i++** // has **n** running time
- **sum=sum+i*i*i** // has **4n** running time as this statement is under loop which is executed n times for true condition
- **return sum** //has 1 running time

Presented By:
Sheikh Muhammad Aamir

SUB-ALGORITHMS

Study Point

- A *sub-algorithm* is a complete and independently defined algorithmic module which is used (or *invoked* or *called)* by some main algorithm or by some other sub-algorithm.

- A sub-algorithm receives values, called *arguments,* from an originating (calling) algorithm; performs computations; and then sends back the result to the calling algorithm.

Presented By:
Sheikh Muhammad Aamir

SUB-ALGORITHMS...

Study Point

- The sub-algorithm is defined independently so that it **may be called by many different algorithms** or called at different times in the same algorithm. The relationship between an algorithm and a sub-algorithm is like the relationship between a main program and a subprogram in a programming language.

- The main difference between the format of a sub-algorithm and that of an algorithm is that the sub-algorithm will usually have a heading of the form NAME(PAR_1, PAR_2.....PAR_K)

Presented By:
Sheikh Muhammad Aamir

Study Point

- Here NAME refers to the name of the sub algorithm which is used when the subalgorithm is called, and PAR_1, PAR_2, ..., PAR_K refer to parameters which are used to transmit data between the sub algorithm and the calling algorithm.

- Another difference is that the sub algorithm will have a Return statement rather than an Exit statement; this emphasizes that control is transferred back to the calling program when the execution of the sub algorithm is completed.

Presented By:
Sheikh Muhammad Aamir

- The following function sub-algorithm MEAN finds the average

Study Point

- The following function sub-algorithm MEAN finds the average

AVG of three numbers A, B and C.

Study Point

Function: MEAN(A, B, C)
1. Set AVG := (A + B + C)/3.
2. Return(AVG).

- Note that MEAN is the name of the subalgorithm and A, B and C are the parameters. The Return statement includes, in parentheses, the variable AVG, whose value is returned to the calling program.

- The subalgorithm MEAN is invoked by an algorithm in the same way as a function subprogram is invoked by a calling program.

Presented By:
Sheikh Muhammad Aamir

59

VARIABLES, DATA TYPES

Write a procedure FIND(DATA, N, LOC1, LOC2) which finds the location LOC1 of the largest element and the location LOC2 of the second largest element in an array DATA with n > 1 elements.

Study Point

Logic:

The elements are examined one by one

Each new element DATA[K] is tested as follows; If

SECOND <= FIRST < DATA[K]

Then FIRST becomes the new SECOND and DATA[K] becomes the new FIRST element

On the other hand, if

SECOND < DATA[K] < FIRST

Then DATA [K] becomes the new SECOND element.

Initially, set FIRST = DATA[1] and SECOND = DATA[2] and check weather or not they are in the right order.

Presented By:
Sheikh Muhammad Aamir

61

Write a procedure FIND(DATA, N, LOC1, LOC2) which

Write a procedure FIND(DATA, N, LOC1, LOC2) which finds the location LOC1 of the largest element and the location LOC2 of the second largest element in an array DATA with n > 1 elements.

Study Point

FIND(DATA, N, LOC1, LOC2)

1. Set FIRST := DATA[1], SECOND := DATA[2], LOC1 := 1, LOC2 := 2.
2. [Are FIRST and SECOND initially correct?]
   If FIRST < SECOND, then:
   (a) Interchange FIRST and SECOND.
   (b) Set LOC1 := 2 and LOC2 := 1.

   [End of If structure.]
3. Repeat for K = 3 to N:
   If FIRST < DATA[K], then:
   (a) Set SECOND := FIRST and FIRST := DATA[K].
   (b) Set LOC2 := LOC1 and LOC1 := K.
   Else if SECOND < DATA[K], then:
   Set SECOND := DATA[K] and LOC2 := K.
   [End of If structure.]
   [End of loop.]
4. Return.

Presented By:
Sheikh Muhammad Aamir

1	2	3	4	5	6
15	75	55	85	35	45

Study Point

Initially suppose

FIRST = DATA[1], SECOND = DATA[2] i.e.

FIRST = 15, SECOND = 75

LOC1 = 1, LOC2 = 2

Initial order checking

If FIRST < SECOND then

Swap FIRST, SECOND and LOC1 = 2, LOC2 = 1

Here the case is true, so FIRST = 75, SECOND = 15

Loop iterations K= 3 to 6

If FIRST < DATA[K] then

Set SECOND = FIRST, and FIRST = DATA[K] and LOC2 = LOC1, LOC1 = K

Else if SECOND < DATA[K] then

Set SECOND = DATA[K] and LOC2 = K

Here for K= 3, FIRST = 75, SECOND = 55, LOC1 = 2, LOC2 = 3

For K = 4, FIRST = 85, SECOND = 75, LOC2 = 2, LOC1 = 4

For K = 5, FIRST = 85, SECOND = 75, LOC2 = 2, LOC1 = 4 (NO CHANGE)

For K = 6, FIRST = 85, SECOND = 75, LOC2 = 2, LOC1 = 4 (NO CHANGE)

Presented By:

Sheikh Muhammad Aamir