greatlearning
Power Ahead

Data Structures and Algorithms For Tech Interviews

greatlearning
Power Ahead

Agenda

- What is Data Structure?
- Array
- Linked List
- Stack
- Algorithms
- Searching
- Sorting

What is Data Structure?

- Made up of 2 words
  - **‘DATA’ + ‘STRUCTURE’**
- It is a way to arrange data in computers
- Example: You might want to store data in
  - Linear fashion - **Array/ Linked List**
  - One on the other - **Stack**
  - Hierarchical Fashion - **Tree**
  - Connect Nodes - **Graph**

greatlearning
Power Ahead

Array

What is an Array?

- Linear Data Structure
- Elements are stored in contiguous memory locations
- Can access elements randomly using index
- Stores homogeneous elements i.e, similar elements
- Syntax:
- Array declaration
    - datatype varname []=new datatype[size];
    - datatype[] varname=new datatype[size];

- Can also do declaration and initialization at once
    - Datatype varname [] = {ele1, ele2, ele3, ele4};

Advantages

- Random access
- Easy sorting and iteration
- Replacement of multiple variables

Disadvantages

- Size is fixed
- Difficult to insert and delete
- If capacity is more and occupancy less, most of the array gets wasted
- Needs contiguous memory to get allocated

Applications

- For storing information in linear fashion
- Suitable for applications that requires frequent searching

Demonstration

```
import java.util.*;

class JavaDemo {
        public static void main (String[] args) {
           int[] priceOfPen= new int[5];
           Scanner in=new Scanner(System.in);
           for(int i=0;i<priceOfPen.length;i++)
              priceOfPen[i]=in.nextInt();

           for(int i=0;i<priceOfPen.length;i++)
                 System.out.print(priceOfPen[i]+" ");
        }
}
```

greatlearning
Power Ahead

Linked List

What is Linked List?

- Linear Data Structure
- Elements can be stored as per memory availability
- Can access elements on linear fashion only
- Stores homogeneous elements i.e, similar elements
- Dynamic in size
- Easy insertion and deletion
- Starting element or Node is the key which is generally termed as head

Advantages

- Dynamic in size
- No wastage as capacity and size is always equal
- Easy insertion and deletion as 1 link manipulation is required
- Efficient memory allocation

Disadvantages

- If head node is lost, linked list is lost
- No random access possible

Applications

- Suitable where memory is limited
- Suitable for applications that requires frequent insertion and deletion

greatlearning
Power Ahead

Demonstration - Linked List Node

```
import java.util. *;

class LLNode{
int data;
LLNode next;

LLNode(int data){
        this.data=data;
        this.next=null;
        }
}
```

Demonstration - Insertion_Beg

```
LLNode insertInBeg(int key,LLNode head){
                        LLNode ttmp=new LLNode(key);
                        if(head==null)
                              head=ttmp;
                        else
                              {
                                    ttmp.next=head;
                                    head=ttmp;
                              }
                        return head;
}
```

Demonstration - Insertion_End

```
LLNode insertInEnd(int key, LLNode head){
  LLNode   ttmp = new   LLNode(key);
  LLNode   ttmp1 = head;

  if (ttmp1 == null)
     head = ttmp;

  else{
     while (ttmp1.next != null)
        ttmp1 = ttmp1.next;
     ttmp1.next = ttmp;
  }
   return head;
}
```

Insertion At Particular Position

```
LLNode insertAtPos(int key, int pos, LLNode head){
  LLNode ttmp = new LLNode(key);
  if (pos == 1){
     ttmp.next = head;
     head = ttmp;
     }else{
     LLNode ttmp1 = head;
     for (int i=1;ttmp1 != null & & i < pos;i++)
        ttmp1=ttmp1.next;
     ttmp.next=ttmp1.next;
     ttmp1.next=ttmp;
     }
  return head;
}
```

Demonstration - Deletion

```
LLNode delete(int pos, LLNode head){
  LLNode ttmp = head;
  if (pos == 1)
     head = ttmp.next;
  else{
     for (int i=1;ttmp != null & & i < pos-1;i++)
        ttmp=ttmp.next;
     ttmp.next=ttmp.next.next;
     }
  return head;
}
```

greatlearning
Power Ahead

Stack

What is Stack?

- Linear Data Structure
- Follows LIFO: Last In First Out
- Only the top elements is available to be accessed
- Insertion and deletion takes place from the top
- Eg: stack of plates, chairs etc
- 4 major operations:
  - push(ele) - used to insert element at top
  - pop() - removes the top element from stack
  - isEmpty() - returns true is stack is empty
  - peek() - to get the top element of stack
- All operation works in constant time i.e, O(1)

Advantages

- Maintains data in LIFO manner
- Last element is readily available for use
- All operations are of O(1) complexity

Disadvantages

- Manipulation is restricted to the top of stack
- Not much flexible

Applications

- Recursion
- Parsing
- Browser
- Editors

Demonstration- Using Array

```
class Stack{
  int[] a;
  int top;
  Stack(){
     a = new int[100];
     top = -1;
  }
  void push(int x){
     if (top == a.length - 1)
        System.out.println("overflow");
     else
        a[++top] = x;
  }
```

```
int pop(){
   if (top == -1){
      System.out.println("underflow");
      return -1;
   }
   else
      return (a[top - -]);
}

void display(){
for (int i=0;i <= top;i++)
   System.out.print(a[i] + " ");
System.out.println();
}
```

Demonstration- Using Array

greatlearning
Power Ahead

```
boolean isEmpty(){
  if (top == -1)
     return true;
  else
     return false;
  }

  int peek(){
     if (top == -1)
        return -1;
     return (a[top]);
  }

}
```

Demonstration- Using Linked List

```
class Stack{
    LNode push(int d,LNode head){
        LNode tmp1 = new LNode(d);
        if(head==null)
            head=tmp1;
        else{
            tmp1.next=head;
            head=tmp1;
        }
        return head;
    }
```

```
LNode pop(LNode head){
        if(head==null)
            System.out.println("underflow");
        else
            head=head.next;
        return head;
    }
boolean isEmpty(LNode head){
        if(head==null)
            return true;
        else
            return false; }
```

Demonstration- Using Linked List

```
void display(LNode head){
        System.out.println("\n list is : ");
        if(head==null){
                System.out.println("no LNodes");
                return;
        }
        LNode tmp=head;
        while(tmp!=null){
                System.out.print(tmp.data+" ");
                tmp=tmp.next;
        }
}
```

greatlearning
Power Ahead

Flow Chart

- Pictorial representation of steps to be performed
- Enable visualization of the problem
- Shows clear data flow with the help of arrows
- Can be used with non technical audience too
- Have a definite start and end point

greatlearning
Power Ahead

Pseudocode

- Pseudocode is step by step approach of a problem in simple english
- It can be easily understood by layman and follow no programming construct
- It basically maps the flowchart in simple statements
- Serves in documentation purpose which is vital in organizations

greatlearning
Power Ahead

Algorithm

- It’s is more close to actual programming
- It follows programming construct to some extent
- Algorithm is independent of any programming language
- It is used to analyze time space complexity
- It helps in further optimization of code

greatlearning
Power Ahead

Searching Algorithm

greatlearning
Power Ahead

Binary Search

- Binary Search is one of the searching techniques
- It can be used on sort arrays
- This searching technique follows the divide and conquer strategy and search space always reduces to half in every iteration
- This is a very efficient technique for searching but it needs some order on which partition of the array will occur

greatlearning
Power Ahead

Binary Search - Iterative Algorithm

binarySearch(arr, size)

```
loop until beg is not equal to end
 midIndex = (beg + end)/2
 if (item == arr[midIndex] )
    return midIndex
 else if (item > arr[midIndex] )
    beg = midIndex + 1
 else
    end = midIndex - 1
```

greatlearning
Power Ahead

Binary Search - Recursive Algorithm

```
binarySearch(arr, item, beg, end)
        if beg<=end
            midIndex = (beg + end) / 2
            if item == arr[midIndex]
                return midIndex
            else if item < arr[midIndex]
                return binarySearch(arr, item, midIndex + 1, end)
            else
                return binarySearch(arr, item, beg, midIndex - 1)
           return -1
```

greatlearning
Power Ahead

Binary Search - Time Complexity

- In each iteration, the search space is getting divided by 2. That means that in the current iteration you have to deal with half of the previous iteration array.
- And the above steps continue till beg<end
- Best case could be the case where the first mid-value get matched to the element to be searched
- Best Time Complexity : O(1)
- Average Time Complexity : O(logn)
- Worst Time Complexity : O(logn)

greatlearning
Power Ahead

Binary Search - Space Complexity

- No auxiliary space is required in Binary Search implementation
- Hence space complexity is : O(1)

greatlearning
Power Ahead

Binary Search - Demonstration

Item to be searched=20

input:

0	1	2	3	4
10	11	16	20	23

beg=0, end=4, mid=2

0	1	**2**	3	4
10	11	**16**	20	23

beg=3, end=4, mid=3

0	1	2	**3**	4
10	11	16	**20**	23

Element found at index 3, Hence 3 will get returned

greatlearning
Power Ahead

Sorting Algorithm

greatlearning
Power Ahead

Bubble Sort Algorithm

greatlearning
Power Ahead

Bubble Sort

- Bubble sort is one of the easiest and brute force sorting algorithm
- It is used to sort elements in either ascending or descending order
- Every element is compared with every other element in bubble sort
- It basically does swapping of elements if they are not in the right order depending on their value and the intended order
- Nested loop will be used to implement this algorithm

greatlearning
Power Ahead

Bubble Sort - Algorithm

```
Bubble Sort(arr, size)
    for i=0 to n-i-1
        for j=0 to n-i-2
            if arr[j]>arr[j+1]
                Swap arr[j] and arr[j+1]
```

Bubble Sort - Time Complexity

- Each and every element is compared with the other elements for array which takes n time and the above steps continues for n iterations
- In the best case that is sorted array, we can do some modification by using flag to check whether the lament is already sorted or not
- Best Time Complexity : O(n)
- Average Time Complexity : O(n^2)
- Worst Time Complexity : O(n^2)

greatlearning
Power Ahead

Bubble Sort - Space Complexity

- No auxiliary space is required in bubble sort implementation
- Hence space complexity is : O(1)

Bubble Sort - Demonstration

input:

0	1	2	3	4
23	10	16	11	20

After i=0

0	1	2	3	4
10	16	11	20	23

After i=1

0	1	2	3	4
10	11	16	20	23

Bubble Sort - Demonstration

After i=2

0	1	2	3	4
10	11	16	20	23

After i=3

0	1	2	3	4
10	11	16	20	23

After i=4

0	1	2	3	4
10	11	16	20	23

greatlearning
Power Ahead

Selection Sort Algorithm

greatlearning
Power Ahead

Selection Sort

- It is a simple sort algorithm that revolves around the comparison
- In each iteration, one element gets placed
- We choose the minimum element in the array and place is at the beginning of the array by swapping with the front element
- We can also do this by choosing maximum element and placing it at the rear end
- Selection sort basically selects an element in every iteration and place it at the appropriate position

greatlearning
Power Ahead

Selection Sort - Algorithm

SelectionSort(arr, n)

```
        iterate (n - 1) times
        set the first unsorted element index as the min
            for each of the unsorted elements
                if element < currentMin
                    set element's index as new min
            swap element at min with first unsorted position
end selectionSort
```

greatlearning
Power Ahead

Selection Sort - Time Complexity

- In the worst case, in every iteration, we have to traverse the entire array for finding min elements and this will continue for all n elements. Hence this will perform n^2 operations in total
- Best Time Complexity: O(n^2)
- Average Time Complexity: O(n^2)
- Worst Time Complexity: O(n^2)

greatlearning
Power Ahead

Selection Sort - Space Complexity

- No auxiliary space is required in Selection Sort implementation that is we are not using any arrays, linked list, stack, queue, etc to store our elements
- Hence space complexity is: O(1)

Selection Sort - Demonstration

Input:

0	1	2	3	4
23	10	16	11	20

First step - marking of sorted part

0	1	2	3	4
10	23	16	11	20

After i=1

0	1	2	3	4
10	11	16	23	20

Selection Sort - Demonstration

After i=2

0	1	2	3	4
10	11	16	23	20

After i=3

0	1	2	3	4
10	11	16	20	23

After i=4, no iteration is required as the last element is already sorted

0	1	2	3	4
10	11	16	20	23

greatlearning
Power Ahead

Insertion Sort Algorithm

greatlearning
Power Ahead

Insertion Sort

- It is one of the easiest and brute force sorting algorithm
- Insertion sort is used to sort elements in either ascending or descending order
- In insertion sort, we maintain a sorted part and unsorted part
- It works just like playing cards i.e picking one card and sorting it with the cards that we have in our hand already which in turn are sorted
- With every iteration, one item from unsorted is moved to the sorted part.
- First element is picked and considered as sorted
- Then we start picking from 2nd elements onwards and start comparison with elements in sorted part
- We shift the elements from sorted by one element until an appropriate location is not found for the picked element
- This continues till all the elements get exhausted

Insertion Sort - Algorithm

Insertion Sort(arr, size)

consider 0th element as sorted part

for each element from i=2 to n-1

tmp = arr[i]

for j=i-1 to 0

If a[j]>tmp

Then right shift it by one position

put tmp at current j

greatlearning
Power Ahead

Insertion Sort - Time Complexity

- In the worst case, it will take n to pick all elements and then at max n shifts to set it to the right position.
- In best case that is sorted array we will just pick the elements but no shifting will take place leading it to n time complexity that is every element is traversed at least once
- Best Time Complexity : O(n)
- Average Time Complexity : O(n^2)
- Worst Time Complexity : O(n^2)

greatlearning
Power Ahead

Insertion Sort - Space Complexity

- No auxiliary space is required in Insertion sort implementation that is we are not using any arrays, linked list, stack, queue, etc to store our elements
- Hence space complexity is : O(1)

Insertion Sort - Demonstration

input:

0	1	2	3	4
23	10	16	11	20

First step - marking of sorted part

0	1	2	3	4
23	10	16	11	20

After i=1

0	1	2	3	4
10	23	16	11	20

After i=2

0	1	2	3	4
10	16	23	11	20

Insertion Sort - Demonstration

After i=3

0	1	2	3	4
10	11	16	23	20

After i=4

0	1	2	3	4
10	11	16	20	23

Selection Vs Bubble Vs Insertion

greatlearning
Power Ahead

Selection	Bubble	Insertion
Select smallest in every iteration do single swap	Adjacent swap of every element with the other element where ording is incorrect	Take and put the element one by one and put it in the right place in the sorted part.
Best case time complexity is O(n2)	Best case time complexity is O(n)	Best case time complexity is O(n)
Works better than Insertion as no of swaps are significantly low	Worst efficiency as too many swaps are required in comparison to selection and insertion	Works better than bubble as no of swaps are significantly low
It is in-place	It is in-place	It is in-place
Not stable	Stable	Stable

greatlearning
Power Ahead

Thank You