DataStructures:

Data structure is a way of storing and organizing data. Data structure provide a way to process and store data efficiently.

**For example:**

Imagine you have pile of books on the table and you are going to read these books one by one from the top. Can you think of any data structure which can simulate with this?  
Yes, youp can simply use stack to store the books, so it can be accessed in Last in first out fashion.

## ****Arrays****

* 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 a linear fashion
* Suitable for applications that require frequent searching

## ****Demonstration of Array****

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]+" ");

}

}

Input:

23 13 56 78 10

Output:

23 13 56 78 10

Also Read: [*How to choose the right programming language for Data Science?*](https://www.mygreatlearning.com/blog/choose-the-right-programming-language/)

## ****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, the linked list is lost
* No random access possible

#### **Applications**

* Suitable where memory is limited
* Suitable for applications that require frequent insertion and deletion

## ****Demonstration of Linked List****

import java.util.\*;

class LLNode{

int data;

LLNode next;

LLNode(int data)

{

this.data=data;

this.next=null;

}

}

class Demo{

LLNode head;

LLNode insertInBeg(int key,LLNode head)

{

LLNode ttmp=new LLNode(key);

if(head==null)

head=ttmp;

else

{

ttmp.next=head;

head=ttmp;

}

return head;

}

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;

}

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;

}

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;

}

int length(LLNode head)

{

LLNode ttmp=head;

int c=0;

if(ttmp==null)

return 0;

else

{

while(ttmp!=null)

{ ttmp=ttmp.next;

c++;

}

}

return c;

}

LLNode reverse(LLNode head)

{

LLNode prevLNode=null,curLNode=head,nextLNode=null;

while(curLNode!=null)

{

nextLNode=curLNode.next;

curLNode.next=prevLNode;

prevLNode=curLNode;

curLNode=nextLNode;

}

head=prevLNode;

return head;

}

void display(LLNode head)

{

LLNode ttmp=head;

while(ttmp!=null)

{System.out.print(ttmp.data+" ");

ttmp=ttmp.next;

}

}

public static void main(String[] args)

{

LinkedListDemo l=new LinkedListDemo();

l.head=null;

Scanner in=new Scanner(System.in);

do

{

System.out.println("\n\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*");

System.out.println("\n1.Insert In End");

System.out.println("\n2.Insert In Beg");

System.out.println("\n3.Insert At A Particular Pos");

System.out.println("\n4.Delete At a Pos");

System.out.println("\n5.Length");

System.out.println("\n6.Reverse");

System.out.println("\n7.Display");

System.out.println("\n8.EXIT");

System.out.println("\nenter ur choice : ");

int n=in.nextInt();

switch(n)

{case 1: System.out.println("\nenter the value ");

l.head=l.insertInEnd(in.nextInt(),l.head);

break;

case 2: System.out.println("\nenter the value");

l.head=l.insertInBeg(in.nextInt(),l.head);

break;

case 3: System.out.println("\nenter the value");

l.head=l.insertAtPos(in.nextInt(),in.nextInt(),l.head);

break;

case 4:

l.head=l.delete(in.nextInt(),l.head);

break;

case 5:

System.out.println(l.length(l.head));

break;

case 6:

l.head=l.reverse(l.head);

break;

case 7:

l.display(l.head);

break;

case 8: System.exit(0);

break;

default: System.out.println("\n Wrong Choice!");

break;

}

System.out.println("\n do u want to cont... ");

}while(in.nextInt()==1);

}

}

Output:

\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*

1.Insert In End

2.Insert In Beg

3.Insert At A Particular Pos

4.Delete At a Pos

5.Length

6.Reverse

7.Display

8.EXIT

enter ur choice :

1

enter the value

23

do u want to cont...

1

\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*

1.Insert In End

2.Insert In Beg

3.Insert At A Particular Pos

4.Delete At a Pos

5.Length

6.Reverse

7.Display

8.EXIT

enter ur choice :

1

enter the value

56

do u want to cont...

1

\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*

1.Insert In End

2.Insert In Beg

3.Insert At A Particular Pos

4.Delete At a Pos

5.Length

6.Reverse

7.Display

8.EXIT

enter ur choice :

2

enter the value

10

do u want to cont...

1

\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*

1.Insert In End

2.Insert In Beg

3.Insert At A Particular Pos

4.Delete At a Pos

5.Length

6.Reverse

7.Display

8.EXIT

enter ur choice :

7

10 23 56

do u want to cont...

1

\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*

1.Insert In End

2.Insert In Beg

3.Insert At A Particular Pos

4.Delete At a Pos

5.Length

6.Reverse

7.Display

8.EXIT

enter ur choice :

3

enter the value

67

2

do u want to cont...

1

\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*

1.Insert In End

2.Insert In Beg

3.Insert At A Particular Pos

4.Delete At a Pos

5.Length

6.Reverse

7.Display

8.EXIT

enter ur choice :

7

10 23 67 56

do u want to cont...

1

\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*

1.Insert In End

2.Insert In Beg

3.Insert At A Particular Pos

4.Delete At a Pos

5.Length

6.Reverse

7.Display

8.EXIT

enter ur choice :

4

2

do u want to cont...

1

\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*

1.Insert In End

2.Insert In Beg

3.Insert At A Particular Pos

4.Delete At a Pos

5.Length

6.Reverse

7.Display

8.EXIT

enter ur choice :

7

10 67 56

do u want to cont...

1

\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*

1.Insert In End

2.Insert In Beg

3.Insert At A Particular Pos

4.Delete At a Pos

5.Length

6.Reverse

7.Display

8.EXIT

enter ur choice :

6

do u want to cont...

1

\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*

1.Insert In End

2.Insert In Beg

3.Insert At A Particular Pos

4.Delete At a Pos

5.Length

6.Reverse

7.Display

8.EXIT

enter ur choice :

7

56 67 10

do u want to cont...

## ****Stack****

* Linear Data Structures using Java
* Follows LIFO: Last In First Out
* Only the top elements are available to be accessed
* Insertion and deletion takes place from the top
* Eg: a 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 the stack
* All operation works in constant time i.e, O(1)

#### **Advantages**

* Maintains data in a LIFO manner
* The last element is readily available for use
* All operations are of O(1) complexity

#### **Disadvantages**

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

#### **Applications**

* Recursion
* Parsing
* Browser
* Editors

Also Read: [Data Structures using C](https://www.mygreatlearning.com/blog/data-structures-using-c/)

## ****Demonstration of Stack – using****[Array](https://www.mygreatlearning.com/blog/what-is-an-array-learn-more-in-one-read/)

import java.util.\*;

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();

}

boolean isEmpty()

{

if(top==-1)

return true;

else

return false;

}

int peek()

{

if(top==-1)

return -1;

return (a[top]);

}

}

public class Demo

{

public static void main(String args[])

{

Stack s=new Stack();

Scanner in= new Scanner(System.in);

do

{System.out.println("\n\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*");

System.out.println("\n1.PUSH");

System.out.println("\n2.POP");

System.out.println("\n3.PEEK");

System.out.println("\n4 IS EMPTY");

System.out.println("\n5.EXIT");

System.out.println("\n enter ur choice : ");

switch(in.nextInt())

{

case 1:

System.out.println("\nenter the value ");

s.push(in.nextInt());

break;

case 2:

System.out.println("\n popped element : "+ s.pop());

break;

case 3:

System.out.println("\n top element : "+ s.peek());

break;

case 4: System.out.println("\n is empty : "+ s.isEmpty());

break;

case 5: System.exit(0);

break;

default: System.out.println("\n Wrong Choice!");

break;

}

System.out.println("\n do u want to cont... ");

}while(in.nextInt()==1);

}

}

Output:

\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*

1.PUSH

2.POP

3.PEEK

4 IS EMPTY

5.EXIT

enter ur choice :

1

enter the value

12

do u want to cont...

1

\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*

1.PUSH

2.POP

3.PEEK

4 IS EMPTY

5.EXIT

enter ur choice :

1

enter the value

56

do u want to cont...

1

\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*

1.PUSH

2.POP

3.PEEK

4 IS EMPTY

5.EXIT

enter ur choice :

2

popped element : 56

do u want to cont...

1

\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*

1.PUSH

2.POP

3.PEEK

4 IS EMPTY

5.EXIT

enter ur choice :

4

is empty : false

do u want to cont...

1

\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*

1.PUSH

2.POP

3.PEEK

4 IS EMPTY

5.EXIT

enter ur choice :

2

popped element : 12

do u want to cont...

## ****Demonstration of Stack – using LinkedList****

import java.util.\*;

class LNode

{

int data;

LNode next;

LNode(int d)

{

data=d;

}

}

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;

}

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;

}

}

boolean isEmpty(LNode head)

{

if(head==null)

return true;

else

return false;

}

int peek(LNode head)

{

if(head==null)

return -1;

return head.data;

}

}

public class Demo{

public static void main(String[] args)

{

Stack s=new Stack();

LNode head=null;

Scanner in=new Scanner(System.in);

do

{System.out.println("\n\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*");

System.out.println("\n1.PUSH");

System.out.println("\n2.POP");

System.out.println("\n3.PEEK");

System.out.println("\n4 IS EMPTY");

System.out.println("\n5 DISPLAY");

System.out.println("\n6.EXIT");

System.out.println("\n enter ur choice : ");

switch(in.nextInt())

{

case 1:

System.out.println("\nenter the value ");

head=s.push(in.nextInt(),head);

break;

case 2:

head=s.pop(head);

break;

case 3:

System.out.println("\n top element : "+ s.peek(head));

break;

case 4:

System.out.println("\n is empty : "+ s.isEmpty(head));

break;

case 5: s.display(head);

break;

case 6: System.exit(0);

break;

default: System.out.println("\n Wrong Choice!");

break;

}

System.out.println("\n do u want to cont... ");

}while(in.nextInt()==1);

}

}

Output

\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*

1.PUSH

2.POP

3.PEEK

4 IS EMPTY

5 DISPLAY

6.EXIT

enter ur choice :

1

enter the value

12

do u want to cont...

1

\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*

1.PUSH

2.POP

3.PEEK

4 IS EMPTY

5 DISPLAY

6.EXIT

enter ur choice :

1

enter the value

56

do u want to cont...

1

\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*

1.PUSH

2.POP

3.PEEK

4 IS EMPTY

5 DISPLAY

6.EXIT

enter ur choice :

5

list is :

56 12

do u want to cont...

1

\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*

1.PUSH

2.POP

3.PEEK

4 IS EMPTY

5 DISPLAY

6.EXIT

enter ur choice :

3

top element : 56

do u want to cont...

1

\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*

1.PUSH

2.POP

3.PEEK

4 IS EMPTY

5 DISPLAY

6.EXIT

enter ur choice :

4

is empty : false

do u want to cont...

1

## ****Queue****

* Linear Data Structure
* Follows FIFO: First In First Out
* Insertion can take place from the rear end.
* Deletion can take place from the front end.
* Eg: queue at ticket counters, bus station
* 4 major operations:
  + enqueue(ele) – used to insert element at top
  + dequeue() – removes the top element from queue
  + peekfirst() – to get the first element of the queue
  + peeklast() – to get the last element of the queue
* All operation works in constant time i.e, O(1)

#### **Advantages**

* Maintains data in FIFO manner
* Insertion from beginning and deletion from end takes O(1) time

#### **Applications**

* Scheduling
* Maintaining playlist
* Interrupt handling

## ****Demonstration of Queue- using Array****

import java.util.\*;

class Queue{

int front;

int rear;

int[] arr;

Queue()

{

front=rear=-1;

arr=new int[10];

}

void enqueue(int a)

{

if(rear==arr.length-1)

System.out.println("overflow");

else

arr[++rear]=a;

if(front==-1)

front++;

}

int dequeue()

{

int x=-1;

if(front==-1)

System.out.println("underflow");

else

x=arr[front++];

if(rear==0)

rear--;

return x;

}

void display()

{

for(int i=front;i<=rear;i++)

System.out.print(arr[i]+" ");

System.out.println();

}

}

public class QueueDemo{

public static void main(String[] args)

{

Queue ob=new Queue();

ob.enqueue(1);

ob.enqueue(2);

ob.enqueue(3);

ob.enqueue(4);

ob.enqueue(5);

ob.display();

ob.dequeue();

ob.display();

}

}

Output:

1 2 3 4 5

2 3 4 5

## ****Demonstration of Queue- using LinkedList****

class LNode{

int data;

LNode next;

LNode(int d)

{

data=d;

}

}

class Queue{

LNode enqueue(LNode head,int a)

{

LNode tmp=new LNode(a);

if(head==null)

head=tmp;

else

{

LNode tmp1=head;

while(tmp1.next!=null)

tmp1=tmp1.next;

tmp1.next=tmp;

}

return head;

}

LNode dequeue(LNode head)

{

if(head==null)

System.out.println("underflow");

else

head=head.next;

return head;

}

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;

}

}

}

public class QueueDemoLL{

public static void main(String[] args)

{

Queue ob=new Queue();

LNode head=null;

head=ob.enqueue(head,1);

head=ob.enqueue(head,2);

head=ob.enqueue(head,3);

head=ob.enqueue(head,4);

head=ob.enqueue(head,5);

ob.display(head);

head=ob.dequeue(head);

ob.display(head);

}

}

Output

list is :

1 2 3 4 5

list is :

2 3 4 5

## [Binary Tree](https://www.mygreatlearning.com/academy/learn-for-free/courses/binary-trees?gl_blog_id=17069)

* Hierarchical  Data Structure
* Topmost element is known as the root of the tree
* Every Node can have at most 2 children in the binary tree
* Can access elements randomly using index
* Eg: File system hierarchy
* Common traversal methods:
  + preorder(root) : print-left-right
  + postorder(root) : left-right-print
  + inorder(root) : left-print-right

#### **Advantages**

* Can represent data with some relationship
* Insertion and search are much efficient

#### **Disadvantages**

* Sorting is difficult
* Not much flexible

#### **Applications**

* File system hierarchy
* Multiple variations of the binary tree have a wide variety of applications

## ****Demonstration of Binary Tree****

class TLNode

{

int data;

TLNode left,right;

TLNode(int d)

{

data=d;

}

}

public class BinaryTree

{

static void preorder(TLNode r)

{

if(r==null)

return;

System.out.print(r.data+" ");

preorder(r.left);

preorder(r.right);

}

static void inorder(TLNode r)

{

if(r==null)

return;

inorder(r.left);

System.out.print(r.data+" ");

inorder(r.right);

}

static void postorder(TLNode r)

{

if(r==null)

return;

postorder(r.left);

postorder(r.right);

System.out.print(r.data+" ");

}

public static void main(String[] args)

{

TLNode root=new TLNode(1);

root.left=new TLNode(2);

root.right=new TLNode(3);

root.left.left=new TLNode(4);

root.left.right=new TLNode(5);

root.right.left=new TLNode(6);

root.right.right=new TLNode(7);

preorder(root);

System.out.println();

inorder(root);

System.out.println();

postorder(root);

System.out.println();

}

}

Output

1 2 4 5 3 6 7

4 2 5 1 6 3 7

4 5 2 6 7 3 1

Also Read: [Understanding Trees in Data Structures](https://www.mygreatlearning.com/blog/understanding-trees-in-data-structures/)

## ****Binary Search Tree****

* Binary tree with the additional restriction
* Restriction:
  + The left child must always be less than the root node
  + The right child must always be greater than the root node
* Insertion, Deletion, Search is much more efficient than a binary tree

#### **Advantages**

* Maintains order in elements
* Can easily find the min and max Nodes in the tree
* Inorder traversal gives sorted elements

#### **Disadvantages**

* Random access not possible
* Ordering adds complexity

#### **Applications**

* Suitable for sorted hierarchical data

## ****Demonstration of Binary Search Tree****

class TLNode{

int data;

TLNode left,right;

TLNode(int d)

{

data=d;

}

}

public class BST{

TLNode root;

TLNode insert(int d,TLNode root)

{

if(root==null)

root=new TLNode(d);

else if(d<=root.data)

root.left=insert(d,root.left);

else

root.right=insert(d,root.right);

return root;

}

TLNode search(int d,TLNode root)

{

if(root.data==d)

return root;

else if(d<root.data)

return search(d,root.left);

else

return search(d,root.right);

}

void inorder(TLNode r)

{

if(r==null)

return;

inorder(r.left);

System.out.println(r.data);

inorder(r.right);

}

TLNode delete(TLNode root, int data)

{

if (root == null) return root;

if (data < root.data)

root.left = delete(root.left, data);

else if (data > root.data)

root.right = delete(root.right, data);

else

{

if (root.left == null)

return root.right;

else if (root.right == null)

return root.left;

root.data = minValue(root.right);

root.right = delete(root.right, root.data);

}

return root;

}

int minValue(TLNode root)

{

int minv = root.data;

while (root.left != null)

{

minv = root.left.data;

root = root.left;

}

return minv;

}

public static void main(String[] args)

{

BST ob=new BST();

ob.root=ob.insert(50,ob.root);

ob.root=ob.insert(30,ob.root);

ob.root=ob.insert(20,ob.root);

ob.root=ob.insert(20,ob.root);

ob.root=ob.insert(70,ob.root);

ob.root=ob.insert(60,ob.root);

ob.root=ob.insert(80,ob.root);

ob.root=ob.delete(ob.root,50);

System.out.println("\*\*\*\*\*\*" +ob.root.data);

ob.inorder(ob.root);

TLNode find=ob.search(30,ob.root);

if(find==null)

System.out.println("not found");

else

System.out.println("found : "+find.data);

}

}

Output:

\*\*\*\*\*\*60

20

20

30

60

70

80

found : 30

## ****Heap****

* Binary Heap can be visualized array as a complete binary tree
* Arr[0] element will be treated as root
* length(A) – size of array
* heapSize(A) – size of heap
* Generally used when we are dealing with minimum and maximum elements
* For ith node

|  |  |
| --- | --- |
| (i-1)/2 | Parent |
| (2\*i)+1 | Left child |
| (2\*i)+2 | Right Child |

#### **Advantages**

* Can be of 2 types: min heap and max heap
* Min heap keeps smallest and element and top and max keeps largest
* O(1) for dealing with min or max elements

#### **Disadvantages**

* Random access not possible
* Only min or max element is available for accessibility

#### **Applications**

* Suitable for applications dealing with priority
* Scheduling algorithm
* caching

## ****Demonstration of Max Heap****

import java.util.\*;

class Heap{

int heapSize;

void build\_max\_heap(int[] a)

{

heapSize=a.length;

for(int i=(heapSize/2);i>=0;i--)

max\_heapify(a,i);

}

void max\_heapify(int[] a,int i)

{

int l=2\*i+1;

int r=2\*i+2;

int largest=i;

if(l<heapSize &&a[l]>a[largest])

largest=l;

if(r<heapSize &&a[r]>a[largest])

largest=r;

if(largest!=i)

{

int t=a[i];

a[i]=a[largest];

a[largest]=t;

max\_heapify(a,largest);

}

}

//to delete the max element

int extract\_max(int[] a)

{

if(heapSize<0)

System.out.println("underflow");

int max=a[0];

a[0]=a[heapSize-1];

heapSize--;

max\_heapify(a,0);

return max;

}

void increase\_key(int[] a,int i,int key)

{

if(key<a[i])

System.out.println("error");

a[i]=key;

while(i>=0 && a[(i-1)/2]<a[i])

{

int t=a[(i-1)/2];

a[(i-1)/2]=a[i];

a[i]=t;

i=(i-1)/2;

}

}

void print\_heap(int a[])

{

for(int i=0;i<heapSize;i++)

System.out.println(a[i]+" ");

}

}

public class HeapDemo{

public static void main(String[] args)

{

Scanner in=new Scanner(System.in);

int n=in.nextInt();

int a[]=new int[n];

System.out.println("enter the elements of array");

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

a[i]=in.nextInt();

Heap ob=new Heap();

ob.build\_max\_heap(a);

ob.print\_heap(a);

System.out.println("maximum element is : "+ob.extract\_max(a));

ob.print\_heap(a);

System.out.println("maximum element is : "+ob.extract\_max(a));

ob.increase\_key(a,6,800);

ob.print\_heap(a);

}

}

Output

7

enter the elements of array

50 100 10 1 3 20 5

100

50

20

1

3

10

5

maximum element is : 100

50

5

20

1

3

10

maximum element is : 50

800

5

20

1

3

## ****Hashing****

* Uses special Hash function
* A hash function maps element to an address for storage
* This provides constant-time access
* Collision is handled by collision resolution techniques
* Collision resolution technique
  + Chaining
  + Open Addressing

#### **Advantages**

* The hash function helps in fetching element in constant time
* An efficient way to store elements

#### **Disadvantages**

* Collision resolution increases complexity

#### **Applications**

* Suitable for the application needs constant time fetching

## ****Demonstration of HashSet – to find string has unique characters****

import java.util.\*;

class HashSetDemo1{

static boolean isUnique(String s)

{

HashSet<Character> set =new HashSet<Character>();

for(int i=0;i<s.length();i++)

{

char c=s.charAt(i);

if(c==' ')

continue;

if(set.add(c)==false)

return false;

}

return true;

}

public static void main(String[] args)

{

String s="helo wqty ";

boolean ans=isUnique(s);

if(ans)

System.out.println("string has unique characters");

else

System.out.println("string does not have unique characters");

}

}

Output:

string has unique characters

## ****Demonstration of HashMap – count the characters in string****

import java.util.\*;

class HashMapDemo

{

static void check(String s)

{

HashMap<Character,Integer> map=new HashMap<Character,Integer>();

for(int i=0;i<s.length();i++)

{char c=s.charAt(i);

if(!map.containsKey(c))

map.put(c,1);

else

map.put(c,map.get(c)+1);

}

Iterator<Character> itr = map.keySet().iterator();

while (itr.hasNext()) {

Object x=itr.next();

System.out.println("count of "+x+" : "+map.get(x));

}

}

public static void main(String[] args)

{

String s="hello";

check(s);

}

}

Output

count of e : 1

count of h : 1

count of l : 2

count of o : 1

**Demonstration of HashTable – to find string has unique characters**

import java.util.\*;

class hashTabledemo {

public static void main(String[] arg)

{

// creating a hash table

Hashtable<Integer, String> h =

new Hashtable<Integer, String>();

Hashtable<Integer, String> h1 =

new Hashtable<Integer, String>();

h.put(3, "Geeks");

h.put(2, "forGeeks");

h.put(1, "isBest");

// create a clone or shallow copy of hash table h

h1 = (Hashtable<Integer, String>)h.clone();

// checking clone h1

System.out.println("values in clone: " + h1);

// clear hash table h

h.clear();

// checking hash table h

System.out.println("after clearing: " + h);

System.out.println("values in clone: " + h1);

}

}

Output

values in clone: {3=Geeks, 2=forGeeks, 1=isBest}

after clearing: {}

values in clone: {3=Geeks, 2=forGeeks, 1=isBest}

## ****Graph****

* Basically it is a group of edges and vertices
* Graph representation
  + G(V, E); where V(G) represents a set of vertices and E(G) represents a set of edges
* The graph can be directed or undirected
* The graph can be connected or disjoint

#### **Advantages**

* finding connectivity
* Shortest path
* min cost to reach from 1 pt to other
* Min spanning tree

#### **Disadvantages**

* Storing graph(Adjacency list and Adjacency matrix) can lead to complexities

#### **Applications**

* Suitable for a circuit network
* Suitable for applications like Facebook, LinkedIn, etc
* Medical science

## ****Demonstration of Graph****

import java.util.\*;

class Graph

{

int v;

LinkedList<Integer> adj[];

Graph(int v)

{

this.v=v;

adj=new LinkedList[v];

for(int i=0;i<v;i++)

adj[i]=new LinkedList<Integer>();

}

void addEdge(int u,int v)

{

adj[u].add(v);

}

void BFS(int s)

{

boolean[] visited=new boolean[v];

LinkedList<Integer> q=new LinkedList<Integer>();

q.add(s);

visited[s]=true;

while(!q.isEmpty())

{

int x=q.poll();

System.out.print(x+" ");

Iterator<Integer> itr=adj[x].listIterator();

while(itr.hasNext())

{

int p=itr.next();

if(visited[p]==false)

{

visited[p]=true;

q.add(p);

}

}

}

}

void DFSUtil(int s,boolean[] visited)

{

visited[s]=true;

System.out.println(s);

Iterator<Integer> itr=adj[s].listIterator();

while(itr.hasNext())

{

int x=itr.next();

if(visited[x]==false)

{

//visited[x]=true;

DFSUtil(x,visited);

}

}

}

void DFS(int s){

boolean visited[]=new boolean[v];

DFSUtil(s,visited);

}

public static void main(String[] args)

{

Graph g=new Graph(4);

g.addEdge(0,1);

g.addEdge(0,2);

g.addEdge(1,2);

g.addEdge(2,0);

g.addEdge(2,3);

g.addEdge(3,3);

g.BFS(2);

g.DFS(2);

}

}

Output:

2 0 3 1 2

0

1

3

# Java Algorithms

The Java collections framework provides various algorithms that can be used to manipulate elements stored in data structures.

Algorithms in Java are static methods that can be used to perform various operations on collections.

Since algorithms can be used on various collections, these are also known as **generic algorithms**.

Let's see the implementation of different methods available in the collections framework.

**1. Sorting Using sort()**

The sort() method provided by the collections framework is used to sort elements. For example,

import java.util.ArrayList;

import java.util.Collections;

class Main {

public static void main(String[] args) {

// Creating an array list

ArrayList<Integer> numbers = new ArrayList<>();

// Add elements

numbers.add(4);

numbers.add(2);

numbers.add(3);

System.out.println("Unsorted ArrayList: " + numbers);

// Using the sort() method

Collections.sort(numbers);

System.out.println("Sorted ArrayList: " + numbers);

}

}

**Output**

Unsorted ArrayList: [4, 2, 3]

Sorted ArrayList: [2, 3, 4]

Here the sorting occurs in natural order (ascending order). However, we can customize the sorting order of the sort() method using *the Comparator interface*.

**2. Shuffling Using shuffle()**

The shuffle() method of the Java collections framework is used to destroy any kind of order present in the data structure. It does just the opposite of the sorting. For example,

import java.util.ArrayList;

import java.util.Collections;

class Main {

public static void main(String[] args) {

// Creating an array list

ArrayList<Integer> numbers = new ArrayList<>();

// Add elements

numbers.add(1);

numbers.add(2);

numbers.add(3);

System.out.println("Sorted ArrayList: " + numbers);

// Using the shuffle() method

Collections.shuffle(numbers);

System.out.println("ArrayList using shuffle: " + numbers);

}

}

**Output**

Sorted ArrayList: [1, 2, 3]

ArrayList using shuffle: [2, 1, 3]

When we run the program, the shuffle() method will return a random output.

The shuffling algorithm is mainly used in games where we want random output.

**3. Routine Data Manipulation**

In Java, the collections framework provides different methods that can be used to manipulate data.

* reverse() - reverses the order of elements
* fill() - replace every element in a collection with the specified value
* copy() - creates a copy of elements from the specified source to destination
* swap() - swaps the position of two elements in a collection
* addAll() - adds all the elements of a collection to other collection

For example,

import java.util.Collections;

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// Creating an ArrayList

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(1);

numbers.add(2);

System.out.println("ArrayList1: " + numbers);

// Using reverse()

Collections.reverse(numbers);

System.out.println("Reversed ArrayList1: " + numbers);

// Using swap()

Collections.swap(numbers, 0, 1);

System.out.println("ArrayList1 using swap(): " + numbers);

ArrayList<Integer> newNumbers = new ArrayList<>();

// Using addAll

newNumbers.addAll(numbers);

System.out.println("ArrayList2 using addAll(): " + newNumbers);

// Using fill()

Collections.fill(numbers, 0);

System.out.println("ArrayList1 using fill(): " + numbers);

// Using copy()

Collections.copy(newNumbers, numbers);

System.out.println("ArrayList2 using copy(): " + newNumbers);

}

}

**Output**

ArrayList1: [1, 2]

Reversed ArrayList1: [2, 1]

ArrayList1 Using swap(): [1, 2]

ArrayList2 using addALl(): [1, 2]

ArrayList1 using fill(): [0, 0]

ArrayList2 using copy(): [0, 0]

**Note**: While performing the copy() method both the lists should be of the same size.

**4. Searching Using binarySearch()**

The binarySearch() method of the Java collections framework searches for the specified element. It returns the position of the element in the specified collections. For example,

import java.util.Collections;

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// Creating an ArrayList

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(1);

numbers.add(2);

numbers.add(3);

// Using binarySearch()

int pos = Collections.binarySearch(numbers, 3);

System.out.println("The position of 3 is " + pos);

}

}

**Output**

The position of 3 is 2.

**Note**: The collection should be sorted before performing the binarySearch() method.

.

**5. Composition**

* frequency() - returns the count of the number of times an element is present in the collection
* disjoint() - checks if two collections contain some common element

For example,

import java.util.Collections;

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// Creating an ArrayList

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(1);

numbers.add(2);

numbers.add(3);

numbers.add(2);

System.out.println("ArrayList1: " + numbers);

int count = Collections.frequency(numbers, 2);

System.out.println("Count of 2: " + count);

ArrayList<Integer> newNumbers = new ArrayList<>();

newNumbers.add(5);

newNumbers.add(6);

System.out.println("ArrayList2: " + newNumbers);

boolean value = Collections.disjoint(numbers, newNumbers);

System.out.println("Two lists are disjoint: " + value);

}

}

**Output**

ArrayList1: [1, 2, 3, 2]

Count of 2: 2

ArrayList2: [5, 6]

Two lists are disjoint: true

**6. Finding Extreme Values**

The min() and max() methods of the Java collections framework are used to find the minimum and the maximum elements, respectively. For example,

import java.util.Collections;

import java.util.ArrayList;

class Main {

public static void main(String[] args) {

// Creating an ArrayList

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(1);

numbers.add(2);

numbers.add(3);

// Using min()

int min = Collections.min(numbers);

System.out.println("Minimum Element: " + min);

// Using max()

int max = Collections.max(numbers);

System.out.println("Maximum Element: " + max);

}

}

**Output**

Minimum Element: 1

Maximum Element: 3