|  |  |
| --- | --- |
| QuickSort O(n log n) | Bubblesort O(n2) |
| Sort(){  findPartitinIndex()  if(low<high)  sort(low,p-1)  sort(p+1,high)  }  findPartitinIndex(){  pivot=arr[high]  for(i==low to high){  if(arr[i]<pivot)  swap  pindex++  }  Swap(arr[pindex],arr[high])  Retun pindex; | Sort(){  For I from0 to n-1  For J= 0 to n-i-1  If (arr[j] > arr[J+1])  Swap arr[j] and arr[J+1]  repeatedly swapping the adjacent elements if they are in wrong order. |
| Selection Sort O(n2) | Merge sort O(log n) |
| Sort(){  For I from 0 to n-1  Int minindex=1  For j = i+1 to n  If(arr[j]< arr[minindex])  Minindex =j  Done  Swap arr[i] and arr[minindex]  done  selection sort algorithm sorts an array by repeatedly finding the minimum element and swapping with I element | MergeSort(arr[], l, r)  If r > l  1. Find the middle point to divide the array into two halves:  middle m = (l+r)/2  2. Call mergeSort for first half:  Call mergeSort(arr, l, m)  3. Call mergeSort for second half:  Call mergeSort(arr, m+1, r)  4. Merge the two halves sorted in step 2 and 3:  Call merge(arr, l, m, r)  Merge fn use two tmp arrays and copy from both at same time . index k = low to high, I,j are size of subarrays |

|  |  |
| --- | --- |
| Tree Traversal | BinarySearch Tree |
| Inorder (^) print BST in ascending  Left ,root,right  If(node ==null)  Return  Inorder(node.left)  Process node.data  Inorder(node.right)  Preorder <. create a copy of the tree  Root,left,right  Process data  Preorder(node.left)  Preorder(node.right)  Postorder > used to delete the tree left,right,root  Postorder(node.left)  Postorder(node.right)  Process(node.data) | Insert(node root, int key)  If(root ==null)  Root = Node(key)  Return root;  If(key<root.data)  Root.left=insert(root.left,key)  Else  Root.right= insert(root.right,key)  Search(Node root,int key)  If(root==null or root.data = key  Return root  If(root.data>key)  Return search(root.left,key)  Else  Return search(root.right,key) |
| Binary Search | SumofTwo from array |
| Binarysearch(arr,int low,int high,int x) if(high>-low)  Mid= low+(high-low)/2  If(arr[mid]==x)  Return mid  If(arr[mid]>x)  Return Binarysearch(arr,low,mid-1)  Else  Return Binarysearch(arr,mid+1,high)  Else -1 | For I from 0 to N  Val = map.get(num[i])  If(val!=null)  Return new int []{val,i}  Else  Map.put(target-nums[i],i) |