|  |  |
| --- | --- |
|  |  |
| Rotate array | 1. Args int nums[] and k rotations 2. Create temp array-a 3. Iterate 4. temp[i+k%original.legnth] = nums[i]   **int** [] temp = **null**;      **for**(**int** i =0; i<arr.length;i++){    temp[i+k+arr.length] =arr[i];      } |
| Re-arrange Odd Even | 1. Temp variables j=-1, temp 2. Iterate 3. If arr is even, %2==0 4. Increment j++ 5. Swap 6. Temp = arr[i] 7. Arr[i]=arr[j]     **int** j =-1; //this is -1 so that we will always have the evens arranged to teh right.  **int** temp;    **for**(**int** i=0; i<arr.length;i++){        **if**(arr[i] % 2==0){        //move the counter by one to the right  j++;    //do the swap      temp= arr[i];  arr[i] = arr[j];  arr[j] = temp;        } |
| Equilibrium | 1. Int sum, int left sum 2. Iterate and sum+=arr[i] 3. Nested iterate and sum =-arr[i] 4. If leftsum ==sum return 1 5. Leftsum += arr[i] |
| Reverse Array | 1. Iterate length/2 2. Temp = arr[i] 3. Arr[i] = arr[length -1 -i] 4. arr[length -1 -i] = temp |
| Swap | * Temp = arr[i] * Arr[i] = arr[b] * Arr[b] = temp |
| HashMap put | 1. Create hash 2. Entry entry = table[hash] 3. If entry != null 4. If entry.key same value k 5. Entry value =v; 6. Else (collision) 7. While entry.next !=null   Entry bucket = new Entry  Entry.next =bucket   1. Else 2. Entry newBucket = new Entry(k,v)   Table[hash] =newBucket |
| Greatest Profit between two elemetns | **public** **int** maxProfit(**int**[] prices) {  **if**(prices==**null**||prices.length<=1)  **return** 0;    **int** min=prices[0]; *// min so far*  **int** max=0;    **for**(**int** i=1; i<prices.length; i++){  max = Math.max(result, prices[i]-min);  min = Math.min(min, prices[i]);  }    **return** max; } |
| Prime Number | //checks whether an int is prime or not.  boolean isPrime(int n) {  //check if n is a multiple of 2  if (n%2==0) return false;  //if not, then just check the odds  for(int i=3;i\*i<=n;i+=2) {  if(n%i==0)  return false;  }  return true;  } |
| Stack | public MyStack(int s) {  maxSize = s;  stackArray = new long[maxSize];  top = -1;  }  public void push(long j) {  stackArray[++top] = j;  }  public long pop() {  return stackArray[top--];  }  public long peek() {  return stackArray[top];  }  public boolean isEmpty() {  return (top == -1);  }  public boolean isFull() {  return (top == maxSize - 1);  }  public static void main(String[] args) {  MyStack theStack = new MyStack(10);  theStack.push(10);  theStack.push(20);  theStack.push(30);  theStack.push(40);  theStack.push(50);    while (!theStack.isEmpty()) {  long value = theStack.pop();  System.out.print(value);  System.out.print(" ");  }  System.out.println("");  }  } |
| Find distinct | **int**[] unique = Arrays.*stream*(arr).distinct().toArray(); |
| Max difference beteween two array elements | |  |  | | --- | --- | | int maxDifference(int arr[], int n) | | | { |  |  | | --- | | int min\_element=arr[0]; | | int diff=arr[1]-arr[0]; |  |  |  | | --- | --- | | for(i=1;i<n;i++) | | | { |  |  | | --- | | if(arr[i]-min\_element>diff) | | diff=arr[i]-min\_element; | |  |  | | --- | | if(arr[i]<min\_element) | | min\_element=arr[i]; | |  |  | | --- | | } | | return diff; | |  |  | | --- | | } | |
| Find duplicates | String[] strArray = {"Java", "JSP", "Servlets", "Java", "Struts", "JSP", "JDBC"};            HashSet<String> set = new HashSet<String>();            for (String arrayElement : strArray)          {              if(!set.add(arrayElement))              {                  System.out.println("Duplicate Element is : "+arrayElement);              } |
| Array Highest and Lowest | **int**[] large=**new** **int**[] {47498, 14526, 74562, 42681, 75283, 45796};        **int** [] data=Arrays.*stream*(large).sorted().toArray();      System.***out***.println(data[0]); //lowest  System.***out***.println(data[data.length-1]); //highest |
| Access an array element | System.***out***.println(data[0]); //lowest |
| Find all elements with sum of k | int count = 0;// Initialize result            // Consider all possible pairs and check their sums          for (int i = 0; i < arr.length; i++)              for (int j = i + 1; j < arr.length; j++)                  if ((arr[i] + arr[j]) == sum)                      count++; |
| **intersection of two arrays** | HashSet<String> set1 = new HashSet<String>(Arrays.asList(inputArray1));            HashSet<String> set2 = new HashSet<String>(Arrays.asList(inputArray2));            set1.retainAll(set2); |
| **intersection of multiple arrays** | HashSet<Integer> unionSet = new HashSet<Integer>();  Then add to the set |
| **Frequence** | Counter elementCountMap.put(i, elementCountMap.get(i)+1); |
| **Filter object** | public List<Article> getAllJavaArticles() {  return articles.stream()  .filter(article -> article.getTags().contains("Java"))  .collect(Collectors.toList());  } |

public List<Article> getAllJavaArticles() {

return articles.stream()

.filter(article -> article.getTags().contains("Java"))

.collect(Collectors.toList());

}