Ademola Abisayo Paul

@sayopaul

Timbuktu

SECTION A

1. List any two sorting algorithm you know and discuss exhaustively.

a. The bubble sort algorithm is a sorting algorithm that makes multiple passes through a list. It starts from the first element of the list and compares the element to the one on the right of it ( i.e next element ) . If it is greater than it , the elements switch/exchange positions . i.e take the index of the first element to be *i* . For each iteration through the list in a single pass , the algorithm checks *i+1* to compare . If *i+1* is greater than *i* , there is a switch, else there is no switch and *i* is incremented . The aim is to move the larger numbers to the end of the list and so , with each pass this is achieved . Given a list with *n* number of items , the total number of passes required to sort it would be *n – 1 .* Since the larger numbers are moved to the end with each pass , it means that for each pass we have *n – 1* items left to sort  *.*  
  
For example , say we have a list of numbers [7,6,2,3,5] to sort using bubble sort , we would do this ;  
  
Let the index of the element per iteration be *i;*

FIRST PASS7,6,2,3,5

When *i*=1 , x[1] = 7 , *i*+1= x[2] = 6 .

Since *i*>*i*+1 ( i.e 7 > 6 ) , we switch them and i is incremented .

6,7,2,3,5 is the result after the first step . Now , we continue ;

When *i*=2,x[2]=7 , *i*+1 = x[3]=2

Since *i*>*i*+1 ( i.e 7 > 2 ) , we switch them and *i* is incremented .

6,2,7,3,5

When *i*=3,x[3]=7 , *i*+1 = x[4]=3

Since *i*>*i*+1 ( i.e 7 > 3 ) , we switch them and *i* is incremented .

6,2,3,7,5

When *i*=4,x[4]=7 , *i*+1 = x[5]=5

Since *i*>*i*+1 ( i.e 7 > 5 ) , we switch them and *i* is incremented .

6,2,3,5,7 .

With this , we have come to the end of the first pass and we can see that the largest number in the list is not at the end of the list .

SECOND PASS - since the largest number is already at the end , we do not need to check for it again when sorting hence we have 4 items to sort ( I.e n-1 items where n=5 ) .

6,2,3,5,7

When *i*=1,x[1]=6 , *i*+1 = x[2]=2

Since *i*>*i*+1 ( i.e 6 > 2 ) , we switch them and *i* is incremented .

2,6,3,5,7

When *i*=2,x[2]=6 , *i*+1 = x[3]=3

Since *i*>*i*+1 ( i.e 6 > 3 ) , we switch them and *i* is incremented .

2,3,6,5,7

When *i*=3,x[3]=6 , *i*+1 = x[4]=5

Since *i*>*i*+1 ( i.e 6 > 5 ) , we switch them and *i* is incremented .

2,3,5,6,7

This is the end of the second pass .

THIRD PASS - since the two largest numbers are already at the end , we do not need to check for them again when sorting hence we have 3 items to sort ( I.e n-2 items where n=5 ) .

2,3,5,6,7

When *i*=1,x[1]=2 , *i*+1 = x[2]=3

Since *i*<*i*+1 ( i.e 2 < 3 ) , there would be no switch and *i* is incremented .

2,3,5,6,7

When *i*=2,x[2]=3 , *i*+1 = x[3]=5

Since *i*<*i*+1 ( i.e 3 < 5 ) , there would be no switch and *i* is incremented .

2,3,5,6,7

This is the end of the third pass .

FOURTH PASS - since the three largest numbers are already at the end , we do not need to check for them again when sorting hence we have 2 items to sort ( I.e n-3 items where n=5 ) .

2,3,5,6,7

When *i*=1,x[1]=2 , *i*+1 = x[2]=3

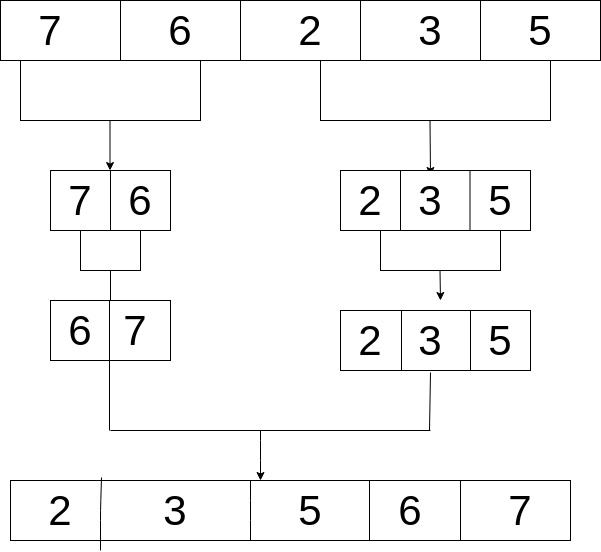
Since *i*<*i*+1 ( i.e 2 < 3 ) , there would be no switch and *i* is incremented .

2,3,5,6,7

This is the end of the fourth pass and we can see that the list is already sorted out .

b. MERGE SORT

The merge sort algorithm is a recursive sorting algorithm that operates in a divide and conquer pattern . “Divide” in the sense that it divides the list into halves recursively . “Conquer” in the sense that the halves are now sorted and merged back into a single list . One of the advantages of the merge sort is that it has the same time complexity for its best , average and worse case scenarios as it would still be the same process everytime . Sorting the list above with merge sort , we would have the following ;



2. Write an algorithm for the sorting algorithms you listed in (1.) above .

Algorithm for bubble sort :

STEP 1 : START

STEP 2: Input a list , list .

STEP 3: Set count = list.count , step = 0 , j = 0

STEP 4: if step < count

STEP 6: for all elements of list

STEP 5: If list[j] > list[j+1]

STEP 6. Set temp = list[j], list[j] = list[j+1], list[j+1]=temp

STEP 7: Set step = step + 1

STEP 8: Go to Step 4  
STEP 9 : Print list

STEP 10 : STOP

Algorithm for Merge Sort :

STEP 1 : START  
STEP 2: Input a list , list

STEP 3 : set count = list.count

STEP 4 : if count = 1 then go to step 7

STEP 5: else divide the list recursively into two halves until it can no more be divided

STEP 6: merge the smaller lists into new list in sorted order

STEP 7 ; Print list

STEP 8 : STOP

3. Write an implementation for the binary search algorithm.

Live demo can be found here <http://sandbox.onlinephpfunctions.com/code/7f6a89da3b80577f270f4a40d4970baeed5bc5ac>

<?php

function binarySearch($list,$itemToBeFound){

$firstIndex = 0;

$lastIndex = (count($list) - 1);

$found = false;

while($firstIndex <= $lastIndex && $found == false){

$midpoint = floor(($firstIndex + $lastIndex) / 2);

if ($itemToBeFound == $list[$midpoint]){

$found = true;

}else{

if($itemToBeFound < $list[$midpoint]){

$lastIndex = $midpoint - 1;

}else if($itemToBeFound > $list[$midpoint]){

$firstIndex = $midpoint + 1;

}

}

}

return $found;

}

//tests the function above

$founded = binarySearch([0, 1, 2, 8, 13, 17, 19, 32, 42],19);

var\_dump($founded);

?>

**SECTION B**

Check out the following links and write algorithms for the problems there

1. <https://app.codility.com/programmers/lessons/1-iterations/binary_gap/>

function solution($N){

$numOfZero = 0;

$zeroArray=[0];

$binary="";

while($N > 1){

$remainder = $N % 2;

$N /= 2;

$binary = $remainder.$binary;

}

$binaryArray = str\_split($binary);

for( $i=0; $i < count($binaryArray); $i++ ){

if($binaryArray[$i] == 1){

for($j= ($i+1); $j< (count($binaryArray) - 1); $j++){

if($binaryArray[$j] == 0 && $binaryArray[$j+1] == 0){

$numOfZero++;

}elseif($binaryArray[$j] == 0 && $binaryArray[$j+1] == 1){

$numOfZero++;

$zeroArray[]=$numOfZero;

$numOfZero=0;

break;

}

}

}

}

return (max($zeroArray) > 0 ) ? max($zeroArray) : 0;

}

2. <https://app.codility.com/programmers/lessons/2-arrays/cyclic_rotation/>

function solution($A,$K){

$newArray = [];

if( $K == 0 ){

return $A;

}

for($i=0; $i<count($A); $i++){

if(count($A) ==1){

return $A;

}

$lastInteger = (count($A) - 1);

$lastElement = $A[$lastInteger];

if(empty($newArray)){

$newArray[]=$lastElement;

unset($A[$lastInteger]);

}

$newArray[]=$A[$i];

}

return solution($newArray,(int)(($K)-1));

}

3. <https://app.codility.com/programmers/lessons/2-arrays/odd_occurrences_in_array/>

function solution ($A){

$indexes = [];

for($i=0; $i<=count($A); $i++){

for($j=$i+1; $j<=count($A)-1; $j++ ){

if ($A[$i] == $A[$j]){

$indexes[]= $i;

$indexes[] = $j;

}

}

}

foreach($indexes as $index){

unset($A[$index]);

}

return reset($A);

}