# Copyright

The content of this book comes from [geeksforgeeks.org](http://geeksforgeeks.org) and it's licensed under [Creative Commons Attribution-NonCommercial-NoDerivs 2.5 India](http://creativecommons.org/licenses/by-nc-nd/2.5/in/deed.en_US)

Made by [Jing](http://www.jing-zhou.me/). 2015.

Updated on August 13, 2015

Head over to [this github repo](https://github.com/gnijuohz/geeksforgeeks-as-books) to report issues or contribute.

# Program for array rotation

Write a function rotate(ar[], d, n) that rotates arr[] of size n by d elements.  
   
 Array

Rotation of the above array by 2 will make array

ArrayRotation1

**METHOD 1 (Use temp array)**

Input arr[] = [1, 2, 3, 4, 5, 6, 7], d = 2, n =7  
1) Store d elements in a temp array  
 temp[] = [1, 2]  
2) Shift rest of the arr[]  
 arr[] = [3, 4, 5, 6, 7, 6, 7]  
3) Store back the d elements  
 arr[] = [3, 4, 5, 6, 7, 1, 2]

**Time complexity** O(n)  
 **Auxiliary Space:** O(d)

**METHOD 2 (Rotate one by one)**

leftRotate(arr[], d, n)  
start  
 For i = 0 to i   
To rotate by one, store arr[0] in a temporary variable temp, move arr[1] to arr[0], arr[2] to arr[1] …and finally temp to arr[n-1]  
Let us take the same example arr[] = [1, 2, 3, 4, 5, 6, 7], d = 2  
  
Rotate arr[] by one 2 times  
  
We get [2, 3, 4, 5, 6, 7, 1] after first rotation and [ 3, 4, 5, 6, 7, 1, 2] after second rotation.  
   
/\*Function to left Rotate arr[] of size n by 1\*/  
void leftRotatebyOne(int arr[], int n);  
  
/\*Function to left rotate arr[] of size n by d\*/  
void leftRotate(int arr[], int d, int n)  
{  
 int i;  
 for (i = 0; i < d; i++)  
 leftRotatebyOne(arr, n);  
}  
  
void leftRotatebyOne(int arr[], int n)  
{  
 int i, temp;  
 temp = arr[0];  
 for (i = 0; i < n-1; i++)  
 arr[i] = arr[i+1];  
 arr[i] = temp;  
}  
  
/\* utility function to print an array \*/  
void printArray(int arr[], int size)  
{  
 int i;  
 for(i = 0; i < size; i++)  
 printf("%d ", arr[i]);  
}  
  
/\* Driver program to test above functions \*/  
int main()  
{  
 int arr[] = {1, 2, 3, 4, 5, 6, 7};  
 leftRotate(arr, 2, 7);  
 printArray(arr, 7);  
 getchar();  
 return 0;  
}

**Time complexity:** O(n\*d)  
 **Auxiliary Space:** O(1)

**METHOD 3 (A Juggling Algorithm)**  
 This is an extension of method 2. Instead of moving one by one, divide the array in different sets  
 where number of sets is equal to GCD of n and d and move the elements within sets.  
 If GCD is 1 as is for the above example array (n = 7 and d =2), then elements will be moved within one set only, we just start with temp = arr[0] and keep moving arr[I+d] to arr[I] and finally store temp at the right place.

Here is an example for n =12 and d = 3. GCD is 3 and

Let arr[] be {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}  
  
a) Elements are first moved in first set – (See below diagram for this movement)  
  
  
  
 arr[] after this step --> {4 2 3 7 5 6 10 8 9 1 11 12}  
  
b) Then in second set.  
 arr[] after this step --> {4 5 3 7 8 6 10 11 9 1 2 12}  
  
c) Finally in third set.  
 arr[] after this step --> {4 5 6 7 8 9 10 11 12 1 2 3}

/\* function to print an array \*/  
void printArray(int arr[], int size);  
  
/\*Fuction to get gcd of a and b\*/  
int gcd(int a,int b);  
  
/\*Function to left rotate arr[] of siz n by d\*/  
void leftRotate(int arr[], int d, int n)  
{  
 int i, j, k, temp;  
 for (i = 0; i < gcd(d, n); i++)  
 {  
 /\* move i-th values of blocks \*/  
 temp = arr[i];  
 j = i;  
 while(1)  
 {  
 k = j + d;  
 if (k >= n)  
 k = k - n;  
 if (k == i)  
 break;  
 arr[j] = arr[k];  
 j = k;  
 }  
 arr[j] = temp;  
 }  
}  
  
/\*UTILITY FUNCTIONS\*/  
/\* function to print an array \*/  
void printArray(int arr[], int size)  
{  
 int i;  
 for(i = 0; i < size; i++)  
 printf("%d ", arr[i]);  
}  
  
/\*Fuction to get gcd of a and b\*/  
int gcd(int a,int b)  
{  
 if(b==0)  
 return a;  
 else  
 return gcd(b, a%b);  
}  
  
/\* Driver program to test above functions \*/  
int main()  
{  
 int arr[] = {1, 2, 3, 4, 5, 6, 7};  
 leftRotate(arr, 2, 7);  
 printArray(arr, 7);  
 getchar();  
 return 0;  
}

**Time complexity:** O(n)  
 **Auxiliary Space:** O(1)

Please see following posts for other methods of array rotation:  
 [Block swap algorithm for array rotation](http://geeksforgeeks.org/?p=2878)  
 [Reversal algorithm for array rotation](http://geeksforgeeks.org/?p=2838)

**References:**  
 <http://www.cs.bell-labs.com/cm/cs/pearls/s02b.pdf>

Please write comments if you find any bug in above programs/algorithms.

### Source

<http://www.geeksforgeeks.org/array-rotation/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [array](http://www.geeksforgeeks.org/tag/array/)

Post navigation

[← Program to count leaf nodes in a binary tree](http://www.geeksforgeeks.org/write-a-c-program-to-get-count-of-leaf-nodes-in-a-binary-tree/) [Reversal algorithm for array rotation →](http://www.geeksforgeeks.org/program-for-array-rotation-continued-reversal-algorithm/)

# Block swap algorithm for array rotation

Write a function rotate(ar[], d, n) that rotates arr[] of size n by d elements.  
   
 Array

Rotation of the above array by 2 will make array

ArrayRotation1

**Algorithm:**

Initialize A = arr[0..d-1] and B = arr[d..n-1]  
1) Do following until size of A is equal to size of B  
  
 a) If A is shorter, divide B into Bl and Br such that Br is of same   
 length as A. Swap A and Br to change ABlBr into BrBlA. Now A  
 is at its final place, so recur on pieces of B.   
  
 b) If A is longer, divide A into Al and Ar such that Al is of same   
 length as B Swap Al and B to change AlArB into BArAl. Now B  
 is at its final place, so recur on pieces of A.  
  
2) Finally when A and B are of equal size, block swap them.

**Recursive Implementation:**

#include<stdio.h>  
  
/\*Prototype for utility functions \*/  
void printArray(int arr[], int size);  
void swap(int arr[], int fi, int si, int d);  
  
void leftRotate(int arr[], int d, int n)  
{   
 /\* Return If number of elements to be rotated is   
 zero or equal to array size \*/   
 if(d == 0 || d == n)  
 return;  
   
 /\*If number of elements to be rotated is exactly   
 half of array size \*/   
 if(n-d == d)  
 {  
 swap(arr, 0, n-d, d);   
 return;  
 }   
   
 /\* If A is shorter\*/   
 if(d < n-d)  
 {   
 swap(arr, 0, n-d, d);  
 leftRotate(arr, d, n-d);   
 }   
 else /\* If B is shorter\*/   
 {  
 swap(arr, 0, d, n-d);  
 leftRotate(arr+n-d, 2\*d-n, d); /\*This is tricky\*/  
 }  
}  
  
/\*UTILITY FUNCTIONS\*/  
/\* function to print an array \*/  
void printArray(int arr[], int size)  
{  
 int i;  
 for(i = 0; i < size; i++)  
 printf("%d ", arr[i]);  
 printf("%\n ");  
}   
  
/\*This function swaps d elements starting at index fi  
 with d elements starting at index si \*/  
void swap(int arr[], int fi, int si, int d)  
{  
 int i, temp;  
 for(i = 0; i<d; i++)   
 {  
 temp = arr[fi + i];  
 arr[fi + i] = arr[si + i];  
 arr[si + i] = temp;  
 }   
}   
  
/\* Driver program to test above functions \*/  
int main()  
{  
 int arr[] = {1, 2, 3, 4, 5, 6, 7};  
 leftRotate(arr, 2, 7);  
 printArray(arr, 7);  
 getchar();  
 return 0;  
}

**Iterative Implementation:**  
 Here is iterative implementation of the same algorithm. Same utility function swap() is used here.

void leftRotate(int arr[], int d, int n)  
{  
 int i, j;  
 if(d == 0 || d == n)  
 return;  
 i = d;  
 j = n - d;  
 while (i != j)  
 {  
 if(i < j) /\*A is shorter\*/  
 {  
 swap(arr, d-i, d+j-i, i);  
 j -= i;  
 }  
 else /\*B is shorter\*/  
 {  
 swap(arr, d-i, d, j);  
 i -= j;  
 }  
 // printArray(arr, 7);  
 }  
 /\*Finally, block swap A and B\*/  
 swap(arr, d-i, d, i);  
}

**Time Complexity:** O(n)

Please see following posts for other methods of array rotation:  
 <http://geeksforgeeks.org/?p=2398>  
 <http://geeksforgeeks.org/?p=2838>

**References:**  
 <http://www.cs.bell-labs.com/cm/cs/pearls/s02b.pdf>

Please write comments if you find any bug in the above programs/algorithms or want to share any additional information about the block swap algorithm.

### Source

<http://www.geeksforgeeks.org/block-swap-algorithm-for-array-rotation/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [array](http://www.geeksforgeeks.org/tag/array/)

Post navigation

[← Reversal algorithm for array rotation](http://www.geeksforgeeks.org/program-for-array-rotation-continued-reversal-algorithm/) [Data Structures and Algorithms | Set 3 →](http://www.geeksforgeeks.org/data-structures-and-algorithms-set-3-2/)

# k largest(or smallest) elements in an array | added Min Heap method

**Question:** Write an efficient program for printing k largest elements in an array. Elements in array can be in any order.

For example, if given array is [1, 23, 12, 9, 30, 2, 50] and you are asked for the largest 3 elements i.e., k = 3 then your program should print 50, 30 and 23.

**Method 1 (Use Bubble k times)**  
 Thanks to Shailendra for suggesting this approach.  
 1) Modify [Bubble Sort](http://en.wikipedia.org/wiki/Bubble_sort) to run the outer loop at most k times.  
 2) Print the last k elements of the array obtained in step 1.

Time Complexity: O(nk)

Like Bubble sort, other sorting algorithms like [Selection Sort](http://en.wikipedia.org/wiki/Selection_sort) can also be modified to get the k largest elements.

**Method 2 (Use temporary array)**  
 K largest elements from arr[0..n-1]

1) Store the first k elements in a temporary array temp[0..k-1].  
 2) Find the smallest element in temp[], let the smallest element be *min*.  
 3) For each element *x* in arr[k] to arr[n-1]  
 If *x* is greater than the min then remove *min* from temp[] and insert *x*.  
 4) Print final k elements of *temp[]*

Time Complexity: O((n-k)\*k). If we want the output sorted then O((n-k)\*k + klogk)

Thanks to nesamani1822 for suggesting this method.

**Method 3(Use Sorting)**  
 1) Sort the elements in descending order in O(nLogn)  
 2) Print the first k numbers of the sorted array O(k).

Time complexity: O(nlogn)

**Method 4 (Use Max Heap)**  
 1) Build a Max Heap tree in O(n)  
 2) Use [Extract Max](http://www.cs.utsa.edu/~dj/cs3343/lecture7.html) k times to get k maximum elements from the Max Heap O(klogn)

Time complexity: O(n + klogn)

**Method 5(Use Oder Statistics)**  
 1) Use order statistic algorithm to find the kth largest element. Please [see the topic selection in worst-case linear time](http://www.cse.ust.hk/~dekai/271/notes/L05/L05.pdf) O(n)  
 2) Use [QuickSort](http://en.wikipedia.org/wiki/Quicksort) Partition algorithm to partition around the kth largest number O(n).  
 3) Sort the k-1 elements (elements greater than the kth largest element) O(kLogk). This step is needed only if sorted output is required.

Time complexity: O(n) if we don’t need the sorted output, otherwise O(n+kLogk)

Thanks to [Shilpi](http://geeksforgeeks.org/forum/topic/print-k-largest-numbers)for suggesting the first two approaches.

**Method 6 (Use Min Heap)**  
 This method is mainly an optimization of method 1. Instead of using temp[] array, use Min Heap.

Thanks to [geek4u](http://geeksforgeeks.org/forum/topic/kth-largest-element)for suggesting this method.

1) Build a Min Heap MH of the first k elements (arr[0] to arr[k-1]) of the given array. O(k)

2) For each element, after the kth element (arr[k] to arr[n-1]), compare it with root of MH.  
 ……a) If the element is greater than the root then make it root and call [heapify](http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/heapSort.htm)for MH  
 ……b) Else ignore it.  
 // The step 2 is O((n-k)\*logk)

3) Finally, MH has k largest elements and root of the MH is the kth largest element.

Time Complexity: O(k + (n-k)Logk) without sorted output. If sorted output is needed then O(k + (n-k)Logk + kLogk)

All of the above methods can also be used to find the kth largest (or smallest) element.

Please write comments if you find any of the above explanations/algorithms incorrect, or find better ways to solve the same problem.

**References:**  
 <http://en.wikipedia.org/wiki/Selection_algorithm>

Asked by [geek4u](http://geeksforgeeks.org/forum/topic/print-k-largest-numbers)

### Source

<http://www.geeksforgeeks.org/k-largestor-smallest-elements-in-an-array/>

# Leaders in an array

Write a program to print all the LEADERS in the array. An element is leader if it is greater than all the elements to its right side. And the rightmost element is always a leader. For example int the array {16, 17, 4, 3, 5, 2}, leaders are 17, 5 and 2.

Let the input array be arr[] and size of the array be *size*.

**Method 1 (Simple)**  
 Use two loops. The outer loop runs from 0 to size – 1 and one by one picks all elements from left to right. The inner loop compares the picked element to all the elements to its right side. If the picked element is greater than all the elements to its right side, then the picked element is the leader.

/\*Function to print leaders in an array \*/  
void printLeaders(int arr[], int size)  
{  
 int i, j;  
  
 for (i = 0; i < size; i++)  
 {  
 for (j = i+1; j < size; j++)  
 {  
 if(arr[i] <= arr[j])  
 break;  
 }   
 if(j == size) // the loop didn't break  
 {  
 printf("%d ", arr[i]);  
 }  
 }  
}  
  
/\*Driver program to test above function\*/  
int main()  
{  
 int arr[] = {16, 17, 4, 3, 5, 2};  
 printLeaders(arr, 6);  
 getchar();  
}  
// Output: 17 5 2

**Time Complexity:** O(n\*n)

**Method 2 (Scan from right)**  
 Scan all the elements from right to left in array and keep track of maximum till now. When maximum changes it’s value, print it.

/\*Function to print leaders in an array \*/  
void printLeaders(int arr[], int size)  
{  
 int max\_from\_right = arr[size-1];  
 int i;  
  
 /\* Rightmost element is always leader \*/  
 printf("%d ", max\_from\_right);  
   
 for(i = size-2; i >= 0; i--)  
 {  
 if(max\_from\_right < arr[i])  
 {  
 printf("%d ", arr[i]);  
 max\_from\_right = arr[i];  
 }  
 }   
}  
  
/\*Driver program to test above function\*/  
int main()  
{  
 int arr[] = {16, 17, 4, 3, 5, 2};  
 printLeaders(arr, 6);  
 getchar();   
}   
// Output: 2 5 17

**Time Complexity:** O(n)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

### Source

<http://www.geeksforgeeks.org/leaders-in-an-array/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [array](http://www.geeksforgeeks.org/tag/array/)

# Maximum sum such that no two elements are adjacent

**Question:** Given an array of positive numbers, find the maximum sum of a subsequence with the constraint that no 2 numbers in the sequence should be adjacent in the array. So 3 2 7 10 should return 13 (sum of 3 and 10) or 3 2 5 10 7 should return 15 (sum of 3, 5 and 7).Answer the question in most efficient way.

**Algorithm:**  
 Loop for all elements in arr[] and maintain two sums incl and excl where incl = Max sum including the previous element and excl = Max sum excluding the previous element.

Max sum excluding the current element will be max(incl, excl) and max sum including the current element will be excl + current element (Note that only excl is considered because elements cannot be adjacent).

At the end of the loop return max of incl and excl.

**Example:**

arr[] = {5, 5, 10, 40, 50, 35}  
  
 inc = 5   
 exc = 0  
  
 For i = 1 (current element is 5)  
 incl = (excl + arr[i]) = 5  
 excl = max(5, 0) = 5  
  
 For i = 2 (current element is 10)  
 incl = (excl + arr[i]) = 15  
 excl = max(5, 5) = 5  
  
 For i = 3 (current element is 40)  
 incl = (excl + arr[i]) = 45  
 excl = max(5, 15) = 15  
  
 For i = 4 (current element is 50)  
 incl = (excl + arr[i]) = 65  
 excl = max(45, 15) = 45  
  
 For i = 5 (current element is 35)  
 incl = (excl + arr[i]) = 80  
 excl = max(5, 15) = 65  
  
And 35 is the last element. So, answer is max(incl, excl) = 80

Thanks to [Debanjan](http://groups.google.co.in/group/algogeeks/browse_thread/thread/eb90efd8f8d4a040/6700a1c909841637?lnk=gst&q=Given+an+array+all+of+whose+elements+are+positive+numbers%2C+find+the+maximum+sum+of+a+subsequence+with+the+constraint+that+no+2+numbers+in+the+sequence+should+be+adjacent+in+the+array#6700a1c909841637) for providing code.

**Implementation:**

#include<stdio.h>  
  
/\*Function to return max sum such that no two elements  
 are adjacent \*/  
int FindMaxSum(int arr[], int n)  
{  
 int incl = arr[0];  
 int excl = 0;  
 int excl\_new;  
 int i;  
  
 for (i = 1; i < n; i++)  
 {  
 /\* current max excluding i \*/  
 excl\_new = (incl > excl)? incl: excl;  
  
 /\* current max including i \*/  
 incl = excl + arr[i];  
 excl = excl\_new;  
 }  
  
 /\* return max of incl and excl \*/  
 return ((incl > excl)? incl : excl);  
}  
  
/\* Driver program to test above function \*/  
int main()  
{  
 int arr[] = {5, 5, 10, 100, 10, 5};  
 printf("%d \n", FindMaxSum(arr, 6));  
 getchar();  
 return 0;  
}

**Time Complexity:** O(n)

Now try the same problem for array with negative numbers also.

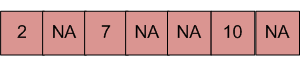
Please write comments if you find any bug in the above program/algorithm or other ways to solve the same problem.

### Source

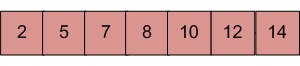
<http://www.geeksforgeeks.org/maximum-sum-such-that-no-two-elements-are-adjacent/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [array](http://www.geeksforgeeks.org/tag/array/)

# Merge an array of size n into another array of size m+n

Asked by Binod  
 **Question:**  
 There are two sorted arrays. First one is of size m+n containing only m elements. Another one is of size n and contains n elements. Merge these two arrays into the first array of size m+n such that the output is sorted.  
   
 Input: array with m+n elements (mPlusN[]).  
 NA => Value is not filled/available in array mPlusN[]. There should be n such array blocks.

Input: array with n elements (N[]).  
 MergeN

Output: N[] merged into mPlusN[] (Modified mPlusN[])  
 

**Algorithm:**

Let first array be mPlusN[] and other array be N[]  
1) Move m elements of mPlusN[] to end.  
2) Start from nth element of mPlusN[] and 0th element of N[] and merge them   
 into mPlusN[].

**Implementation:**

#include <stdio.h>  
  
/\* Assuming -1 is filled for the places where element  
 is not available \*/  
#define NA -1  
  
/\* Function to move m elements at the end of array mPlusN[] \*/  
void moveToEnd(int mPlusN[], int size)  
{  
 int i = 0, j = size - 1;  
 for (i = size-1; i >= 0; i--)  
 if (mPlusN[i] != NA)  
 {  
 mPlusN[j] = mPlusN[i];  
 j--;  
 }  
}  
  
/\* Merges array N[] of size n into array mPlusN[]  
 of size m+n\*/  
int merge(int mPlusN[], int N[], int m, int n)  
{  
 int i = n; /\* Current index of i/p part of mPlusN[]\*/  
 int j = 0; /\* Current index of N[]\*/  
 int k = 0; /\* Current index of of output mPlusN[]\*/  
 while (k < (m+n))  
 {  
 /\* Take an element from mPlusN[] if  
 a) value of the picked element is smaller and we have  
 not reached end of it  
 b) We have reached end of N[] \*/  
 if ((i < (m+n) && mPlusN[i] <= N[j]) || (j == n))  
 {  
 mPlusN[k] = mPlusN[i];  
 k++;  
 i++;  
 }  
 else // Otherwise take emenet from N[]  
 {  
 mPlusN[k] = N[j];  
 k++;  
 j++;  
 }  
 }  
}  
  
/\* Utility that prints out an array on a line \*/  
void printArray(int arr[], int size)  
{  
 int i;  
 for (i=0; i < size; i++)  
 printf("%d ", arr[i]);  
  
 printf("\n");  
}  
  
/\* Driver function to test above functions \*/  
int main()  
{  
 /\* Initialize arrays \*/  
 int mPlusN[] = {2, 8, NA, NA, NA, 13, NA, 15, 20};  
 int N[] = {5, 7, 9, 25};  
 int n = sizeof(N)/sizeof(N[0]);  
 int m = sizeof(mPlusN)/sizeof(mPlusN[0]) - n;  
  
 /\*Move the m elements at the end of mPlusN\*/  
 moveToEnd(mPlusN, m+n);  
  
 /\*Merge N[] into mPlusN[] \*/  
 merge(mPlusN, N, m, n);  
  
 /\* Print the resultant mPlusN \*/  
 printArray(mPlusN, m+n);  
  
 return 0;  
}

Output:

2 5 7 8 9 13 15 20 25

**Time Complexity:** O(m+n)

Please write comment if you find any bug in the above program or a better way to solve the same problem.

### Source

<http://www.geeksforgeeks.org/merge-one-array-of-size-n-into-another-one-of-size-mn/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [array](http://www.geeksforgeeks.org/tag/array/)

# Reversal algorithm for array rotation

Write a function rotate(arr[], d, n) that rotates arr[] of size n by d elements.

Array

Rotation of the above array by 2 will make array

ArrayRotation1

**Method 4(The Reversal Algorithm)**  
 Please read [this](http://geeksforgeeks.org/?p=2398)for first three methods of array rotation.

**Algorithm:**

rotate(arr[], d, n)  
 reverse(arr[], 1, d) ;  
 reverse(arr[], d + 1, n);  
 reverse(arr[], l, n);

Let AB are the two parts of the input array where A = arr[0..d-1] and B = arr[d..n-1]. The idea of the algorithm is:  
 Reverse A to get ArB. /\* Ar is reverse of A \*/  
 Reverse B to get ArBr. /\* Br is reverse of B \*/  
 Reverse all to get (ArBr) r = BA.

For arr[] = [1, 2, 3, 4, 5, 6, 7], d =2 and n = 7  
 A = [1, 2] and B = [3, 4, 5, 6, 7]  
 Reverse A, we get ArB = [2, 1, 3, 4, 5, 6, 7]  
 Reverse B, we get ArBr = [2, 1, 7, 6, 5, 4, 3]  
 Reverse all, we get (ArBr)r = [3, 4, 5, 6, 7, 1, 2]

**Implementation:**

/\*Utility function to print an array \*/  
void printArray(int arr[], int size);  
  
/\* Utility function to reverse arr[] from start to end \*/  
void rvereseArray(int arr[], int start, int end);  
  
/\* Function to left rotate arr[] of size n by d \*/  
void leftRotate(int arr[], int d, int n)  
{  
 rvereseArray(arr, 0, d-1);  
 rvereseArray(arr, d, n-1);  
 rvereseArray(arr, 0, n-1);  
}  
  
/\*UTILITY FUNCTIONS\*/  
/\* function to print an array \*/  
void printArray(int arr[], int size)  
{  
 int i;  
 for(i = 0; i < size; i++)  
 printf("%d ", arr[i]);  
 printf("%\n ");  
}  
  
/\*Function to reverse arr[] from index start to end\*/  
void rvereseArray(int arr[], int start, int end)  
{  
 int i;  
 int temp;  
 while(start < end)  
 {  
 temp = arr[start];  
 arr[start] = arr[end];  
 arr[end] = temp;  
 start++;  
 end--;  
 }  
}  
  
/\* Driver program to test above functions \*/  
int main()  
{  
 int arr[] = {1, 2, 3, 4, 5, 6, 7};  
 leftRotate(arr, 2, 7);  
 printArray(arr, 7);  
 getchar();  
 return 0;  
}

**Time Complexity:** O(n)

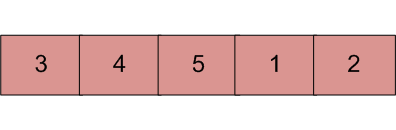
**References:**  
 <http://www.cs.bell-labs.com/cm/cs/pearls/s02b.pdf>

### Source

<http://www.geeksforgeeks.org/program-for-array-rotation-continued-reversal-algorithm/>

# Search an element in a sorted and pivoted array

**Question:**  
 An element in a sorted array can be found in O(log n) time via binary search. But suppose I rotate the sorted array at some pivot unknown to you beforehand. So for instance, 1 2 3 4 5 might become 3 4 5 1 2. Devise a way to find an element in the rotated array in O(log n) time.



**Solution:**  
 Thanks to Ajay Mishra for initial solution.

**Algorithm:**  
 Find the pivot point, divide the array in two sub-arrays and call binary search.  
 The main idea for finding pivot is – for a sorted (in increasing order) and pivoted array, pivot element is the only only element for which next element to it is smaller than it.  
 Using above criteria and binary search methodology we can get pivot element in O(logn) time

Input arr[] = {3, 4, 5, 1, 2}  
Element to Search = 1  
 1) Find out pivot point and divide the array in two   
 sub-arrays. (pivot = 2) /\*Index of 5\*/  
 2) Now call binary search for one of the two sub-arrays.  
 (a) If element is greater than 0th element then   
 search in left array  
 (b) Else Search in right array   
 (1 will go in else as 1 If element is found in selected sub-array then return index   
 Else return -1.

**Implementation:**

/\* Program to search an element in a sorted and pivoted array\*/  
#include <stdio.h>  
  
int findPivot(int[], int, int);  
int binarySearch(int[], int, int, int);  
  
/\* Searches an element no in a pivoted sorted array arrp[]  
 of size arr\_size \*/  
int pivotedBinarySearch(int arr[], int arr\_size, int no)  
{  
 int pivot = findPivot(arr, 0, arr\_size-1);  
  
 // If we didn't find a pivot, then array is not rotated at all  
 if (pivot == -1)  
 return binarySearch(arr, 0, arr\_size-1, no);  
  
 // If we found a pivot, then first compare with pivot and then  
 // search in two subarrays around pivot  
 if (arr[pivot] == no)  
 return pivot;  
 if (arr[0] <= no)  
 return binarySearch(arr, 0, pivot-1, no);  
 else  
 return binarySearch(arr, pivot+1, arr\_size-1, no);  
}  
  
/\* Function to get pivot. For array 3, 4, 5, 6, 1, 2 it will  
 return 3. If array is not rotated at all, then it returns -1 \*/  
int findPivot(int arr[], int low, int high)  
{  
 // base cases  
 if (high < low) return -1;  
 if (high == low) return low;  
  
 int mid = (low + high)/2; /\*low + (high - low)/2;\*/  
 if (mid < high && arr[mid] > arr[mid + 1])  
 return mid;  
 if (mid > low && arr[mid] < arr[mid - 1])  
 return (mid-1);  
 if (arr[low] >= arr[mid])  
 return findPivot(arr, low, mid-1);  
 else  
 return findPivot(arr, mid + 1, high);  
}  
  
/\* Standard Binary Search function\*/  
int binarySearch(int arr[], int low, int high, int no)  
{  
 if (high < low)  
 return -1;  
 int mid = (low + high)/2; /\*low + (high - low)/2;\*/  
 if (no == arr[mid])  
 return mid;  
 if (no > arr[mid])  
 return binarySearch(arr, (mid + 1), high, no);  
 else  
 return binarySearch(arr, low, (mid -1), no);  
}  
  
  
/\* Driver program to check above functions \*/  
int main()  
{  
 // Let us search 3 in below array  
 int arr1[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};  
 int arr\_size = sizeof(arr1)/sizeof(arr1[0]);  
 int no = 3;  
 printf("Index of the element is %d\n", pivotedBinarySearch(arr1, arr\_size, no));  
  
 // Let us search 3 in below array  
 int arr2[] = {3, 4, 5, 1, 2};  
 arr\_size = sizeof(arr2)/sizeof(arr2[0]);  
 printf("Index of the element is %d\n", pivotedBinarySearch(arr2, arr\_size, no));  
  
 // Let us search for 4 in above array  
 no = 4;  
 printf("Index of the element is %d\n", pivotedBinarySearch(arr2, arr\_size, no));  
  
 // Let us search 0 in below array  
 int arr3[] = {1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1};  
 no = 0;  
 arr\_size = sizeof(arr3)/sizeof(arr3[0]);  
 printf("Index of the element is %d\n", pivotedBinarySearch(arr3, arr\_size, no));  
  
 // Let us search 3 in below array  
 int arr4[] = {2, 3, 0, 2, 2, 2, 2, 2, 2, 2};  
 no = 3;  
 arr\_size = sizeof(arr4)/sizeof(arr4[0]);  
 printf("Index of the element is %d\n", pivotedBinarySearch(arr4, arr\_size, no));  
  
 // Let us search 2 in above array  
 no = 2;  
 printf("Index of the element is %d\n", pivotedBinarySearch(arr4, arr\_size, no));  
  
 // Let us search 3 in below array  
 int arr5[] = {1, 2, 3, 4};  
 no = 3;  
 arr\_size = sizeof(arr5)/sizeof(arr5[0]);  
 printf("Index of the element is %d\n", pivotedBinarySearch(arr5, arr\_size, no));  
  
 return 0;  
}

Output:

Index of the element is 8  
Index of the element is 0  
Index of the element is 1  
Index of the element is 3  
Index of the element is 1  
Index of the element is 0  
Index of the element is 2

Please note that the solution may not work for cases where the input array has duplicates.

**Time Complexity** O(logn)

Please write comments if you find any bug in above codes/algorithms, or find other ways to solve the same problem.

### Source

<http://www.geeksforgeeks.org/search-an-element-in-a-sorted-and-pivoted-array/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [array](http://www.geeksforgeeks.org/tag/array/)

Post navigation

[← Function to check if a singly linked list is palindrome](http://www.geeksforgeeks.org/function-to-check-if-a-singly-linked-list-is-palindrome/) [The Great Tree-List Recursion Problem. →](http://www.geeksforgeeks.org/the-great-tree-list-recursion-problem/)

# Sort elements by frequency | Set 1

Asked By Binod  
   
 **Question:**  
 Print the elements of an array in the decreasing frequency if 2 numbers have same frequency then print the one which came 1st  
 E.g. 2 5 2 8 5 6 8 8 output: 8 8 8 2 2 5 5 6.

**METHOD 1 (Use Sorting)**

1) Use a sorting algorithm to sort the elements O(nlogn)   
 2) Scan the sorted array and construct a 2D array of element and count O(n).  
 3) Sort the 2D array according to count O(nlogn).

**Example:**

Input 2 5 2 8 5 6 8 8  
  
 After sorting we get  
 2 2 5 5 6 8 8 8  
  
 Now construct the 2D array as  
 2, 2  
 5, 2  
 6, 1  
 8, 3  
  
 Sort by count  
 8, 3  
 2, 2  
 5, 2  
 6, 1

There is one issue with above approach (thanks to ankit for pointing this out). If we modify the input to 5 2 2 8 5 6 8 8, then we should get 8 8 8 5 5 2 2 6 and not 8 8 8 2 2 5 5 6 as will be the case.  
 To handle this, we should use indexes in step 3, if two counts are same then we should first process(or print) the element with lower index. In step 1, we should store the indexes instead of elements.

Input 5 2 2 8 5 6 8 8  
  
 After sorting we get  
 Element 2 2 5 5 6 8 8 8  
 Index 1 2 0 4 5 3 6 7  
  
 Now construct the 2D array as  
 Index, Count  
 1, 2  
 0, 2  
 5, 1  
 3, 3  
  
 Sort by count (consider indexes in case of tie)  
 3, 3  
 0, 2  
 1, 2  
 5, 1  
   
 Print the elements using indexes in the above 2D array.

**METHOD 2(Use BST and Sorting)**  
 1. Insert elements in BST one by one and if an element is already present then increment the count of the node. Node of the Binary Search Tree (used in this approach) will be as follows.

struct tree  
{  
 int element;  
 int first\_index /\*To handle ties in counts\*/  
 int count;  
}BST;

2.Store the first indexes and corresponding counts of BST in a 2D array.  
 3 Sort the 2D array according to counts (and use indexes in case of tie).

**Time Complexity:** O(nlogn) if a[Self Balancing Binary Search Tree](http://en.wikipedia.org/wiki/Self-balancing_binary_search_tree) is used.

**METHOD 3(Use Hashing and Sorting)**  
 Using a hashing mechanism, we can store the elements (also first index) and their counts in a hash. Finally, sort the hash elements according to their counts.

These are just our thoughts about solving the problem and may not be the optimal way of solving. We are open for better solutions.

**Related Links**  
 http://www.trunix.org/programlama/c/kandr2/krx604.html  
 <http://drhanson.s3.amazonaws.com/storage/documents/common.pdf>  
 http://www.cc.gatech.edu/classes/semantics/misc/pp2.pdf

### Source

<http://www.geeksforgeeks.org/sort-elements-by-frequency/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [array](http://www.geeksforgeeks.org/tag/array/)

# Find the smallest and second smallest element in an array

**Question:** Write an efficient C program to find smallest and second smallest element in an array.

**Difficulty Level:** Rookie

**Algorithm:**

1) Initialize both first and second smallest as INT\_MAX  
 first = second = INT\_MAX  
2) Loop through all the elements.  
 a) If the current element is smaller than first, then update first   
 and second.   
 b) Else if the current element is smaller than second then update   
 second

**Implementation:**

#include <stdio.h>  
#include <limits.h> /\* For INT\_MAX \*/  
  
/\* Function to print first smallest and second smallest elements \*/  
void print2Smallest(int arr[], int arr\_size)  
{  
 int i, first, second;  
  
 /\* There should be atleast two elements \*/  
 if (arr\_size < 2)  
 {  
 printf(" Invalid Input ");  
 return;  
 }  
  
 first = second = INT\_MAX;  
 for (i = 0; i < arr\_size ; i ++)  
 {  
 /\* If current element is smaller than first then update both  
 first and second \*/  
 if (arr[i] < first)  
 {  
 second = first;  
 first = arr[i];  
 }  
  
 /\* If arr[i] is in between first and second then update second \*/  
 else if (arr[i] < second && arr[i] != first)  
 second = arr[i];  
 }  
 if (second == INT\_MAX)  
 printf("There is no second smallest element\n");  
 else  
 printf("The smallest element is %d and second Smallest element is %d\n",  
 first, second);  
}  
  
  
/\* Driver program to test above function \*/  
int main()  
{  
 int arr[] = {12, 13, 1, 10, 34, 1};  
 int n = sizeof(arr)/sizeof(arr[0]);  
 print2Smallest(arr, n);  
 return 0;  
}

Output:

The smallest element is 1 and second Smallest element is 10

The same approach can be used to find the largest and second largest elements in an array.

**Time Complexity:** O(n)

Please write comments if you find any bug in the above program/algorithm or other ways to solve the same problem.

### Source

<http://www.geeksforgeeks.org/to-find-smallest-and-second-smallest-element-in-an-array/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [array](http://www.geeksforgeeks.org/tag/array/)

# Two elements whose sum is closest to zero

**Question:** An Array of integers is given, both +ve and -ve. You need to find the two elements such that their sum is closest to zero.

For the below array, program should print -80 and 85.

[Array_1](http://geeksforgeeks.org/wp-content/uploads/2010/01/Array_1.gif)

**METHOD 1 (Simple)**  
 For each element, find the sum of it with every other element in the array and compare sums. Finally, return the minimum sum.

**Implementation**

# include <stdio.h>  
# include <stdlib.h> /\* for abs() \*/  
# include <math.h>  
void minAbsSumPair(int arr[], int arr\_size)  
{  
 int inv\_count = 0;  
 int l, r, min\_sum, sum, min\_l, min\_r;  
  
 /\* Array should have at least two elements\*/  
 if(arr\_size < 2)  
 {  
 printf("Invalid Input");  
 return;  
 }  
  
 /\* Initialization of values \*/  
 min\_l = 0;  
 min\_r = 1;  
 min\_sum = arr[0] + arr[1];  
  
 for(l = 0; l < arr\_size - 1; l++)  
 {  
 for(r = l+1; r < arr\_size; r++)  
 {  
 sum = arr[l] + arr[r];  
 if(abs(min\_sum) > abs(sum))  
 {  
 min\_sum = sum;  
 min\_l = l;  
 min\_r = r;  
 }  
 }  
 }  
  
 printf(" The two elements whose sum is minimum are %d and %d",  
 arr[min\_l], arr[min\_r]);  
}  
  
/\* Driver program to test above function \*/  
int main()  
{  
 int arr[] = {1, 60, -10, 70, -80, 85};  
 minAbsSumPair(arr, 6);  
 getchar();  
 return 0;  
}

**Time complexity:** O(n^2)

**METHOD 2 (Use Sorting)**  
 Thanks to baskin for suggesting this approach. We recommend to read [this post](http://geeksforgeeks.org/?p=484) for background of this approach.

**Algorithm**  
 1) Sort all the elements of the input array.  
 2) Use two index variables l and r to traverse from left and right ends respectively. Initialize l as 0 and r as n-1.  
 3) sum = a[l] + a[r]  
 4) If sum is -ve, then l++  
 5) If sum is +ve, then r–  
 6) Keep track of abs min sum.  
 7) Repeat steps 3, 4, 5 and 6 while l Implementation

# include <stdio.h>  
# include <math.h>  
# include <limits.h>  
  
void quickSort(int \*, int, int);  
  
/\* Function to print pair of elements having minimum sum \*/  
void minAbsSumPair(int arr[], int n)  
{  
 // Variables to keep track of current sum and minimum sum  
 int sum, min\_sum = INT\_MAX;  
  
 // left and right index variables  
 int l = 0, r = n-1;  
  
 // variable to keep track of the left and right pair for min\_sum  
 int min\_l = l, min\_r = n-1;  
  
 /\* Array should have at least two elements\*/  
 if(n < 2)  
 {  
 printf("Invalid Input");  
 return;  
 }  
  
 /\* Sort the elements \*/  
 quickSort(arr, l, r);  
  
 while(l < r)  
 {  
 sum = arr[l] + arr[r];  
  
 /\*If abs(sum) is less then update the result items\*/  
 if(abs(sum) < abs(min\_sum))  
 {  
 min\_sum = sum;  
 min\_l = l;  
 min\_r = r;  
 }  
 if(sum < 0)  
 l++;  
 else  
 r--;  
 }  
  
 printf(" The two elements whose sum is minimum are %d and %d",  
 arr[min\_l], arr[min\_r]);  
}  
  
/\* Driver program to test above function \*/  
int main()  
{  
 int arr[] = {1, 60, -10, 70, -80, 85};  
 int n = sizeof(arr)/sizeof(arr[0]);  
 minAbsSumPair(arr, n);  
 getchar();  
 return 0;  
}  
  
/\* FOLLOWING FUNCTIONS ARE ONLY FOR SORTING  
 PURPOSE \*/  
void exchange(int \*a, int \*b)  
{  
 int temp;  
 temp = \*a;  
 \*a = \*b;  
 \*b = temp;  
}  
  
int partition(int arr[], int si, int ei)  
{  
 int x = arr[ei];  
 int i = (si - 1);  
 int j;  
  
 for (j = si; j <= ei - 1; j++)  
 {  
 if(arr[j] <= x)  
 {  
 i++;  
 exchange(&arr[i], &arr[j]);  
 }  
 }  
  
 exchange (&arr[i + 1], &arr[ei]);  
 return (i + 1);  
}  
  
/\* Implementation of Quick Sort  
arr[] --> Array to be sorted  
si --> Starting index  
ei --> Ending index  
\*/  
void quickSort(int arr[], int si, int ei)  
{  
 int pi; /\* Partitioning index \*/  
 if(si < ei)  
 {  
 pi = partition(arr, si, ei);  
 quickSort(arr, si, pi - 1);  
 quickSort(arr, pi + 1, ei);  
 }  
}

**Time Complexity:** complexity to sort + complexity of finding the optimum pair = O(nlogn) + O(n) = O(nlogn)

Asked by Vineet

Please write comments if you find any bug in the above program/algorithm or other ways to solve the same problem.

### Source

<http://www.geeksforgeeks.org/two-elements-whose-sum-is-closest-to-zero/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [array](http://www.geeksforgeeks.org/tag/array/)

Post navigation

[← Why C treats array parameters as pointers?](http://www.geeksforgeeks.org/why-c-treats-array-parameters-as-pointers/) [C Language | Set 4 →](http://www.geeksforgeeks.org/c-language-set-4/)

# Write a program to reverse an array

**Iterative way:**  
 1) Initialize start and end indexes.   
 start = 0, end = n-1  
 2) In a loop, swap arr[start] with arr[end] and change start and end as follows.  
 start = start +1; end = end – 1

/\* Function to reverse arr[] from start to end\*/  
void rvereseArray(int arr[], int start, int end)  
{  
 int temp;  
 while(start < end)  
 {  
 temp = arr[start];   
 arr[start] = arr[end];  
 arr[end] = temp;  
 start++;  
 end--;  
 }   
}   
  
/\* Utility that prints out an array on a line \*/  
void printArray(int arr[], int size)  
{  
 int i;  
 for (i=0; i < size; i++)  
 printf("%d ", arr[i]);  
  
 printf("\n");  
}   
  
/\* Driver function to test above functions \*/  
int main()   
{  
 int arr[] = {1, 2, 3, 4, 5, 6};  
 printArray(arr, 6);  
 rvereseArray(arr, 0, 5);  
 printf("Reversed array is \n");  
 printArray(arr, 6);  
 getchar();  
 return 0;  
}

Time Complexity: O(n)

**Recursive Way:**  
 1) Initialize start and end indexes  
 start = 0, end = n-1  
 2) Swap arr[start] with arr[end]  
 3) Recursively call reverse for rest of the array.

/\* Function to reverse arr[] from start to end\*/  
void rvereseArray(int arr[], int start, int end)  
{  
 int temp;  
 if(start >= end)  
 return;  
 temp = arr[start];   
 arr[start] = arr[end];  
 arr[end] = temp;  
 rvereseArray(arr, start+1, end-1);   
}   
  
/\* Utility that prints out an array on a line \*/  
void printArray(int arr[], int size)  
{  
 int i;  
 for (i=0; i < size; i++)  
 printf("%d ", arr[i]);  
  
 printf("\n");  
}   
  
/\* Driver function to test above functions \*/  
int main()   
{  
 int arr[] = {1, 2, 3, 4, 5};  
 printArray(arr, 5);  
 rvereseArray(arr, 0, 4);  
 printf("Reversed array is \n");  
 printArray(arr, 5);  
 getchar();  
 return 0;  
}

Time Complexity: O(n)

Please write comments if you find any bug in the above programs or other ways to solve the same problem.

### Source

<http://www.geeksforgeeks.org/write-a-program-to-reverse-an-array/>

Category: [Arrays](http://www.geeksforgeeks.org/category/c-arrays/) Tags: [array](http://www.geeksforgeeks.org/tag/array/)