GeeksforGeeks
A computer science portal for geeks

Custom Search

Geeks Classes

Write an Article

Iterative Quick Sort

Following is a typical recursive implementation of Quick Sort that uses last element as pivot.

C++

```
// CPP code for recursive function of Quicksort
#include<bits/stdc++.h>

using namespace std;

// Function to swap numbers
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

/* This function takes last element as pivot,
   places the pivot element at its correct
   position in sorted  array, and places
   all smaller (smaller than pivot) to left
   of pivot and all greater elements to
   right of pivot */
int partition (int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);

    for (int j = l; j <= h- 1; j++)
    {
        if (arr[j] <= x)
        {
            i++;
            swap (&arr[i], &arr[j]);
        }
    }
    swap (&arr[i + 1], &arr[h]);
    return (i + 1);
}

/* A[] --> Array to be sorted,
l --> Starting index,
h --> Ending index */
void quickSort(int A[], int l, int h)
{
    if (l < h)
    {
        /* Partitioning index */
        int p = partition(A, l, h);
        quickSort(A, l, p - 1);
        quickSort(A, p + 1, h);
    }

}

// Driver code
int main(){

    int n = 5;
    int arr[n] = {4, 2, 6, 9, 2};

    quickSort(arr, 0, n-1);

        for(int i = 0; i < n; i++){
            cout << arr[i] << " ";
    }

    return 0;
}
```

Run on IDE

Java

```
// Java program for implementation of QuickSort
import java.util.*;

class QuickSort
{
    /* This function takes last element as pivot,
    places the pivot element at its correct
    position in sorted array, and places all
    smaller (smaller than pivot) to left of
    pivot and all greater elements to right
    of pivot */
    static int partition(int arr[], int low, int high)
    {
        int pivot = arr[high];
        int i = (low-1); // index of smaller element
        for (int j=low; j<=high-1; j++)
        {
            // If current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;

                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }

    /* The main function that implements QuickSort()
```

```
     arr[] --> Array to be sorted,
     low --> Starting index,
     high --> Ending index */
     static void qSort(int arr[], int low, int high)
     {
         if (low < high)
         {
             /* pi is partitioning index, arr[pi] is
             now at right place */
             int pi = partition(arr, low, high);

             // Recursively sort elements before
             // partition and after partition
             qSort(arr, low, pi-1);
             qSort(arr, pi+1, high);
         }
     }

     // Driver code
     public static void main(String args[])
     {

         int n = 5;
         int arr[] = {4, 2, 6, 9, 2};

         qSort(arr, 0, n-1);

         for(int i =0;i<n;i++){
             System.out.print(arr[i]+" ");
         }

     }
 }
```

Run on IDE

Python3

```
# A typical recursive Python
# implementation of QuickSort

# Function takes last element as pivot,
# places the pivot element at its correct
# position in sorted array, and places all
# smaller (smaller than pivot) to left of
# pivot and all greater elements to right
# of pivot
def partition(arr, low, high):
    i = (low - 1)         # index of smaller element
    pivot = arr[high]     # pivot

    for j in range(low, high):

        # If current element is smaller
        # than or equal to pivot
        if arr[j] <= pivot:

            # increment index of
            # smaller element
            i += 1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return (i + 1)

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low --> Starting index,
# high --> Ending index

# Function to do Quick sort
def quickSort(arr,low,high):
    if low < high:

        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr, low, high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)

# Driver Code
if __name__ == '__main__' :

    arr = [4, 2, 6, 9, 2]
    n = len(arr)

    # Calling quickSort function
    quickSort(arr, 0, n - 1)

    for i in range(n):
        print(arr[i], end = " ")
```

Run on IDE

C#

```
// C# program for implementation of
// QuickSort
using System;

class GFG {

    /* This function takes last element
    as pivot, places the pivot element
    at its correct position in sorted
    array, and places all smaller
    (smaller than pivot) to left of
    pivot and all greater elements to
    right of pivot */
    static int partition(int []arr,
                         int low, int high)
    {
        int temp;
        int pivot = arr[high];

        // index of smaller element
        int i = (low-1);
        for (int j = low; j <= high-1; j++)
        {

            // If current element is
            // smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;

                // swap arr[i] and arr[j]
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high]
        // (or pivot)
        temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }

    /* The main function that implements
    QuickSort() arr[] --> Array to be
    sorted,
    low --> Starting index,
    high --> Ending index */
    static void qSort(int []arr, int low,
                                int high)
    {
        if (low < high)
        {
            /* pi is partitioning index,
            arr[pi] is now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements
            // before partition and after
            // partition
            qSort(arr, low, pi-1);
            qSort(arr, pi+1, high);
        }
    }

    // Driver code
    public static void Main()
    {

        int n = 5;
        int []arr = {4, 2, 6, 9, 2};

        qSort(arr, 0, n-1);

        for(int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");
    }
}

// This code is contributed by nitin mittal.
```

Run on IDE

Output:

```
2 2 4 6 9
```

The above implementation can be optimized in many ways

1) The above implementation uses last index as pivot. This causes worst-case behavior on alrea rted arrays, which is a commonly occurring case. The problem can be solved by choosing either a random

index for the pivot, or choosing the middle index of the partition or choosing the median of the first, middle and last element of the partition for the pivot. (See this for details)

2) To reduce the recursion depth, recur first for the smaller half of the array, and use a tail call to recurse into the other.

3) Insertion sort works better for small subarrays. Insertion sort can be used for invocations on such small arrays (i.e. where the length is less than a threshold t determined experimentally). For example, this library implementation of qsort uses insertion sort below size 7.

Despite above optimizations, the function remains recursive and uses function call stack to store intermediate values of l and h. The function call stack stores other bookkeeping information together with parameters. Also, function calls involve overheads like storing activation record of the caller function and then resuming execution.

The above function can be easily converted to iterative version with the help of an auxiliary stack. Following is an iterative implementation of the above recursive code.

C/C++

```
// An iterative implementation of quick sort
#include <stdio.h>

// A utility function to swap two elements
void swap ( int* a, int* b )
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function is same in both iterative and recursive*/
int partition (int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);

    for (int j = l; j <= h- 1; j++)
    {
        if (arr[j] <= x)
        {
            i++;
            swap (&arr[i], &arr[j]);
        }
    }
    swap (&arr[i + 1], &arr[h]);
    return (i + 1);
}

/* A[] --> Array to be sorted,
   l  --> Starting index,
   h  --> Ending index */
void quickSortIterative (int arr[], int l, int h)
{
    // Create an auxiliary stack
    int stack[ h - l + 1 ];
    // initialize top of stack
    int top = -1;

    // push initial values of l and h to stack
    stack[ ++top ] = l;
    stack[ ++top ] = h;

    // Keep popping from stack while is not empty
    while ( top >= 0 )
    {
        // Pop h and l
        h = stack[ top-- ];
        l = stack[ top-- ];

        // Set pivot element at its correct position
        // in sorted array
        int p = partition( arr, l, h );

        // If there are elements on left side of pivot,
        // then push left side to stack
        if ( p-1 > l )
        {
            stack[ ++top ] = l;
            stack[ ++top ] = p - 1;
        }

        // If there are elements on right side of pivot,
        // then push right side to stack
        if ( p+1 < h )
        {
            stack[ ++top ] = p + 1;
            stack[ ++top ] = h;
        }
    }
}

// A utility function to print contents of arr
void printArr( int arr[], int n )
{
    int i;
    for ( i = 0; i < n; ++i )
        printf( "%d ", arr[i] );
}

// Driver program to test above functions
int main()
{
    int arr[] = {4, 3, 5, 2, 1, 3, 2, 3};
    int n = sizeof( arr ) / sizeof( *arr );
    quickSortIterative( arr, 0, n - 1 );
    printArr( arr, n );
    return 0;
}
```

Run on IDE

Java

```
// Java program for implementation of QuickSort
import java.util.*;

class QuickSort
{
    /* This function takes last element as pivot,
    places the pivot element at its correct
    position in sorted array, and places all
```

```
       smaller (smaller than pivot) to left of
       pivot and all greater elements to right
       of pivot */
    static int partition(int arr[], int low, int high)
    {
        int pivot = arr[high];

        // index of smaller element
        int i = (low-1);
        for (int j = low; j <= high-1; j++)
        {
            // If current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;

                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }

/* A[] --> Array to be sorted,
   l  --> Starting index,
   h  --> Ending index */
static void quickSortIterative (int arr[], int l, int h)
{
    // Create an auxiliary stack
    int[] stack = new int[h-l+1];

    // initialize top of stack
    int top = -1;

    // push initial values of l and h to stack
    stack[++top] = l;
    stack[++top] = h;

    // Keep popping from stack while is not empty
    while (top >= 0)
    {
        // Pop h and l
        h = stack[top--];
        l = stack[top--];

        // Set pivot element at its correct position
        // in sorted array
        int p = partition(arr, l, h);

        // If there are elements on left side of pivot,
        // then push left side to stack
        if (p-1 > l)
        {
            stack[++top] = l;
            stack[++top] = p - 1;
        }

        // If there are elements on right side of pivot,
        // then push right side to stack
        if (p+1 < h)
        {
                stack[++top] = p + 1;
                stack[++top] = h;
            }
        }
}
    // Driver code
    public static void main(String args[])
    {
        int arr[] = {4, 3, 5, 2, 1, 3, 2, 3};
        int n = 8;

        // Function calling
        quickSortIterative(arr, 0, n-1);

        for(int i = 0; i < n; i++){
            System.out.print(arr[i] + " ");
        }
    }
}
```

Run on IDE

Python

```
# Python program for implementation of Quicksort

# This function is same in both iterative and recursive
def partition(arr,l,h):
    i = ( l - 1 )
    x = arr[h]

    for j in range(l , h):
        if   arr[j] <= x:

            # increment index of smaller element
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[h] = arr[h],arr[i+1]
    return (i+1)

# Function to do Quick sort
# arr[] --> Array to be sorted,
# l  --> Starting index,
# h  --> Ending index
def quickSortIterative(arr,l,h):

    # Create an auxiliary stack
    size = h - l + 1
    stack = [0] * (size)

    # initialize top of stack
    top = -1

    # push initial values of l and h to stack
    top = top + 1
    stack[top] = l
    top = top + 1
    stack[top] = h

    # Keep popping from stack while is not empty
    while top >= 0:

        # Pop h and l
        h = stack[top]
        top = top - 1
        l = stack[top]
        top = top - 1

        # Set pivot element at its correct position in
        # sorted array
        p = partition( arr, l, h )

        # If there are elements on left side of pivot,
        # then push left side to stack
        if p-1 > l:
            top = top + 1
            stack[top] = l
            top = top + 1
            stack[top] = p - 1

        # If there are elements on right side of pivot,
        # then push right side to stack
        if p+1 < h:
            top = top + 1
            stack[top] = p + 1
            top = top + 1
            stack[top] = h

# Driver code to test above
arr = [4, 3, 5, 2, 1, 3, 2, 3]
n = len(arr)
quickSortIterative(arr, 0, n-1)
print ("Sorted array is:")
for i in range(n):
    print ("%d" %arr[i]),

# This code is contributed by Mohit Kumra
```

Run on IDE

C#

```
// C# program for implementation of QuickSort
using System;

class GFG {

    /* This function takes last element as pivot,
    places the pivot element at its correct
    position in sorted array, and places all
    smaller (smaller than pivot) to left of
    pivot and all greater elements to right
    of pivot */
    static int partition(int []arr, int low,
                                    int high)
    {
        int temp;
        int pivot = arr[high];

        // index of smaller element
        int i = (low-1);
        for (int j = low; j <= high-1; j++)
        {
            // If current element is smaller
            // than or equal to pivot
            if (arr[j] <= pivot)
            {
                i++;

                // swap arr[i] and arr[j]
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high]
        // (or pivot)

        temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }

    /* A[] --> Array to be sorted,
    l --> Starting index,
    h --> Ending index */
    static void quickSortIterative (int []arr,
                                int l, int h)
    {
        // Create an auxiliary stack
        int[] stack = new int[h-l+1];

        // initialize top of stack
        int top = -1;

        // push initial values of l and h to
        // stack
        stack[++top] = l;
        stack[++top] = h;

        // Keep popping from stack while
        // is not empty
        while (top >= 0)
        {
            // Pop h and l
            h = stack[top--];
            l = stack[top--];

            // Set pivot element at its
            // correct position in
            // sorted array
            int p = partition(arr, l, h);

            // If there are elements on
            // left side of pivot, then
            // push left side to stack
            if (p-1 > l)
            {
                stack[++top] = l;
                stack[++top] = p - 1;
            }

            // If there are elements on
            // right side of pivot, then
            // push right side to stack
            if (p+1 < h)
            {
                stack[++top] = p + 1;
                stack[++top] = h;
            }
        }
    }

    // Driver code
    public static void Main()
    {
        int []arr = {4, 3, 5, 2, 1, 3, 2, 3};
        int n = 8;

        // Function calling
        quickSortIterative(arr, 0, n-1);

        for(int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");
    }
}

// This code is contributed by anuj_67.
```

Run on IDE

Output:

```
1 2 2 3 3 3 4 5
```

The above mentioned optimizations for recursive quick sort can also be applied to iterative version.

1) Partition process is same in both recursive and iterative. The same techniques to choose optimal pivot can also be applied to iterative version.

2) To reduce the stack size, first push the indexes of smaller half.

3) Use insertion sort when the size reduces below a experimentally calculated threshold.

References:

http://en.wikipedia.org/wiki/Quicksort

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Practice Tags : Sorting

Article Tags : Sorting Quick Sort

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

QuickSort on Singly Linked List
QuickSort
Heap Sort
QuickSort Tail Call Optimization (Reducing worst case space to Log n)
When does the worst case of Quicksort occur?

Multiset Equivalence Problem
Print all triplets with given sum
Maximise the number of toys that can be purchased with amount K
Minimum swaps so that binary search can be applied
Closest product pair in an array

(Login to Rate)

3.4 Average Difficulty : **3.4/5.0**
Based on **68** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!

A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY
About Us
Careers
Privacy Policy
Contact Us

LEARN
Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE
Company-wise
Topic-wise
Contests
Subjective Questions

CONTRIBUTE
Write an Article
Write Interview Experience
Internships
Videos

@geeksforgeeks, Some rights reserved