

Iterative Quick Sort

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

```
/* A typical recursive implementation of quick sort */

/* This function takes last element as pivot, places the pivot element
   correct position in sorted array, and places all smaller (smaller t
   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 in
void quickSort(int A[], int l, int h)
{
    if (l < h)
    {
        int p = partition(A, l, h); /* Partitioning index */
        quickSort(A, l, p - 1);
        quickSort(A, p + 1, h);
    }
}
```

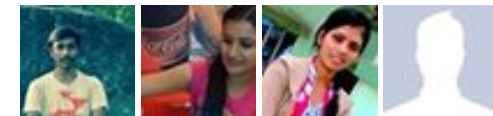
Google™ Custom Search



GeeksforGeeks



53,520 people like [GeeksforGeeks](#).



Interview Experiences

Advanced Data Structures

Dynamic Programming

Greedy Algorithms

Backtracking

Pattern Searching

Divide & Conquer

Mathematical Algorithms

Recursion

}

The above implementation can be optimized in many ways

- 1) The above implementation uses last index as pivot. This causes worst-case behavior on already sorted 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.

```
// 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);
```



Popular Posts

[All permutations of a given string](#)

[Memory Layout of C Programs](#)

[Understanding "extern" keyword in C](#)

[Median of two sorted arrays](#)

[Tree traversal without recursion and without stack!](#)

[Structure Member Alignment, Padding and Data Packing](#)

[Intersection point of two Linked Lists](#)

[Lowest Common Ancestor in a BST.](#)

[Check if a binary tree is BST or not](#)

[Sorted Linked List to Balanced BST](#)

```

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 )
        {

```

Deploy Early. Deploy Often.

DevOps from
Rackspace:

Automation

[FIND OUT HOW ►](#)



```

        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;
}

```

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

Recent Comments

Aman Hi, Why arent we checking for conditions...

[Write a C program to Delete a Tree.](#) · 9 minutes ago

[kzs please provide solution for the problem...](#)

[Backtracking | Set 2 \(Rat in a Maze\)](#) · 12 minutes ago

Sanjay Agarwal bool

[tree::Root_to_leaf_path_given_sum\(tree...](#)

[Root to leaf path sum equal to a given number](#) · 37 minutes ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

[Count trailing zeroes in factorial of a number](#) · 39 minutes ago

newCoder3006 If the array contains negative numbers also. We...

[Find subarray with given sum](#) · 1 hour ago

newCoder3006 Code without using while loop. We can do it...

[Find subarray with given sum](#) · 1 hour ago

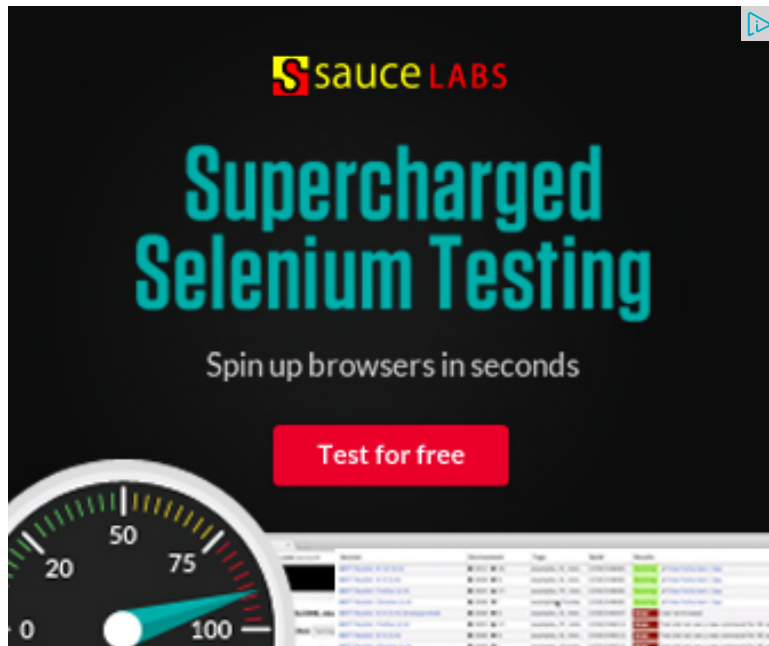
AdChoices 

[▶ Java Array](#)

[▶ C++ Sort List](#)

[▶ Java Sort](#)

topic discussed above.



AdChoices

► [QuickSort](#)

► [N Sort](#)

► [Sort and Stack](#)

AdChoices

► [Bubble Sort C](#)

► [Insertion Sort](#)

► [Selection Sort](#)

Related Tpoics:

- [Remove minimum elements from either side such that \$2 \times \text{min}\$ becomes more than max](#)
- [Divide and Conquer | Set 6 \(Search in a Row-wise and Column-wise Sorted 2D Array\)](#)
- [Bucket Sort](#)
- [Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1](#)
- [Find the number of zeroes](#)
- [Find if there is a subarray with 0 sum](#)
- [Divide and Conquer | Set 5 \(Strassen's Matrix Multiplication\)](#)
- [Count all possible groups of size 2 or 3 that have sum as multiple of 3](#)



20



Tweet

0



0

Writing code in comment? Please use [ideone.com](#) and share the link here.

19 Comments

GeeksforGeeks

Sort by Newest ▼



Join the discussion...



code_on • 9 days ago

1. To reduce the recursion depth, recur first for the smaller half of the array, ar other.

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

Please elaborate.

Thanks

^ | v • Reply • Share ›



Harman • 18 days ago

Quick Sort is a Sorting Algorithm based on Divide And Conquer Technique
I have also a good reference with Description, Program Example , Snapshot a simple explanation

<http://geeksprogrammings.blogs...>

^ | v • Reply • Share ›



Nizamuddin Saifi • 6 months ago

We can improve performance of partition function , by simple checking the a[j] swap function will call. eg,

```
int partition(int a[],int l,int h)
{
    int x=a[h];
    int i=l-1,j;

    for(j=l;j<=h-1;++j)
    {
```

`if(a[j]<x &&"" a[j]!="a[++i]") {="" swap(&a[j],&a[i]`

^ | v · Reply · Share ›



VIGY · 7 months ago

THANK YOU SOOOOO MUCH :-)

^ | v · Reply · Share ›



GeeksforGeeks · 11 months ago

It is a valid C99 syntax. Please see <http://en.wikipedia.org/wiki/C...>

^ | v · Reply · Share ›



Sumit Gera · 11 months ago

Isn't using stack[h - l + 1] an invalid syntax?

^ | v · Reply · Share ›



kapil · a year ago

How will recursion depth be reduced if we recur smaller part of the array first?

1 ^ | v · Reply · Share ›



??? · a year ago

Not a kind explanation, but also terrific code I think! Thanks a lot!

^ | v · Reply · Share ›



Aashish Barnwal · a year ago

The boundaries high(h) and low(l) are not necessary to specify. However, the to pass as a parameter. So the method signature can be reduced to:
void quickSortIterative (int arr[], int size).

^ | v · Reply · Share ›



Kuldeep Tiwari · a year ago

With iterative implementation, we don't need parameters low(l) and high

void quickSortIterative (int arr[]).

^ | v • Reply • Share ›



kuldeep • a year ago

With iterative implementation, we don't need parameters low(l) and high(h) in recursive implementation, it reduces to -

void quickSortIterative (int arr[]).

^ | v • Reply • Share ›



Aashish → kuldeep • a year ago

The boundaries high(h) and low(l) are not necessary to specify. However, low(l) must be passed as a parameter.

^ | v • Reply • Share ›



Anonymous • a year ago

Quick Sort with (Stable+ Efficient+ in-place Sorting+ NO Need of partition method) including repeating elements is given below (java):

```
public static void quickSort(int A[], int l, int h)
{
    int pivot=(l+h)/2;
    int pivotValue=A[pivot];

    int i=l, j=h;
    while(i<=j)
    {
        while(A[i]<pivotValue)
            i++;

        while(A[j]>pivotValue)
            j--;
    }
}
```


`if(i<=j)`

[see more](#)

3 ^ | v • Reply • Share ›



sindhu → Anonymous • a year ago

Above code will run into infinite loop for input:
40 20 10 80 60 50 7 30 100 - take 60 as pivot element.

^ | v • Reply • Share ›



Anonymous → sindhu • a year ago

I have tried my code with above sample input---
40 20 10 80 60 50 7 30 100 - take 60 as pivot element. It is work

^ | v • Reply • Share ›



Rahul → Anonymous • a year ago

You are basically implementing the partition method inside quicksort. In this way you are improving quick sort. You are just writing the

```
/* Paste your code here (You may delete these lines) */
```

^ | v • Reply • Share ›



Anonymous → Rahul • a year ago

@Rahul- Yes. I am implementing partition code inside quicksort. I have improved the efficiency. Efficiency can be improved by choosing a better pivot element correctly which can be done if we use a randomize function to choose the elements of input array and then applying quicksort on the input array.

^ | v • Reply • Share ›



very nice one

/* Paste your code here (You may **delete** these lines **if not** writing c

^ | v • Reply • Share ›



kaur • 2 years ago

awesome!! :) :)

^ | v • Reply • Share ›



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress & MooTools**, customized by geeksforgeeks team