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);
    }
```


```
}
```

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);
```

Geometric Algorithms

Popular Posts

```
    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 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 rig
        // 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;
}
```

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

topic discussed above.

Related Tpoics:

- Remove minimum elements from either side such that 2*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

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

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...

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]
```

Reply • Share ›

VIGY • 7 months ago

THANK YOU SOOOOO MUCH :-)

Reply • Share ›

GeeksforGeeks • 11 months ago

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

Reply • Share ›

Sumit Gera • 11 months ago

Isn&#039t using stack[h - l +1] an invalid syntax?

Reply • Share ›

kapil • a year ago

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

1 Reply • Share ›

??? • a year ago

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

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).

Reply • Share ›

Kuldeep Tiwari • a year ago

With iterative implementation, we don&#039t need parameters low(l) and high

void quickSortIterative (int arr[]).

Reply • Share ›

kuldeep • a year ago

With iterative implementation, we don't need parameters low(l) and high(h) in r
reduces to -
void quickSortIterative (int arr[]).

Reply • Share ›

Aashish kuldeep • a year ago

The boundaries high(h) and low(l) are not necessary to specify. Howev
a must to pass as a parameter.

Reply • Share ›

Anonymous • a year ago

Quick Sort with (Stable+ Efficient+ in-place Sorting+ NO Need of partition met
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 ▲ | ▼ • 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.

▲ | ▼ • 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 worl

▲ | ▼ • Reply • Share ›

Rahul ➦ Anonymous • a year ago

You are basically implementing the partition method insi
way are you improving quick sort. You are just writing th

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

▲ | ▼ • Reply • Share ›

Anonymous ➦ Rahul • a year ago

@Rahul- Yes. I am implementing partition code inside c
improved the efficiency. Efficiency can be improved by
correctly which can done if we use a randomize functio
the elements of input array and then applying quicksort
input array.

▲ | ▼ • Reply • Share ›

very nice one

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

• Reply • Share ›

kaur • 2 years ago

awesome!! :) :)

• Reply • Share ›

Subscribe Add Disqus to your site