GeeksforGeeks

A computer science portal for geeks

GeeksQuiz

- Home
- Algorithms
- DS
- GATE
- Interview Corner
- Q&A
- C
- C++
- Java
- Books
- Contribute
- Ask a Q
- About

Array
Bit Magic
C/C++
Articles
GFacts
Linked List
MCQ
Misc
Output
String
Tree
Graph

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 at its
   correct position in sorted array, and places all smaller (smaller than piv
   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)
    {
        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);

    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;
}
```

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.

Related Topics:

- Find Union and Intersection of two unsorted arrays
- Pythagorean Triplet in an array
- Maximum profit by buying and selling a share at most twice
- Design a data structure that supports insert, delete, search and getRandom in constant time
- Print missing elements that lie in range 0 – 99
- Iterative Merge Sort
- Group multiple occurrence of array elements ordered by first occurrence
- Given a sorted and rotated array, find if there is a pair with a given sum

Writing code in comment? Please use **ideone.com** and share the link here.

32 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

Hello • a month ago

sizeof(arr) gives how many bytes the array occupies
sizeof(*arr) gives how many bytes the first element of the array occupies
sizeof(*arr)/sizeof(arr) gives the length of the array

• Reply • Share ›

Justice • 2 months ago

Can someone please explain to me what the line of code "int n = sizeof(arr) / sizeof(*arr); does please? It's mainly to do with the * operator on the array - what does this do to the division?

• Reply • Share ›

akash • 2 months ago

Hi, regarding the recursive version of Quicksort I am using 1 less variable as input which being the lower bound variable for the array,
https://ideone.com/uYXm3X

Can someone, please tell me whether its a correct solution and does this modification would be of any use for optimizing the number of instructions slightly.

• Reply • Share ›

jsqihui • 3 months ago

http://codexpi.com/quicksort-p... clear solution I found

• Reply • Share ›

EigenHarsha • 4 months ago

Using one STL stack implementation :

Using one STL stack implementation :
http://ideone.com/r1hGI5

• Reply • Share ›

rihansh • 5 months ago

one can use queue and proceed in LOL (Level Order Traversal) manner ..

• Reply • Share ›

Abhishek K Das • 8 months ago

Here is an implementation of the iterative quickSort using STL stack:

```
void quickSort(int arr[], int lo, int hi) {
        stack<int> start;
        stack<int> end;
        start.push(lo);
        end.push(hi);
        int l, r, p;
        while (!start.empty()) {
                l = start.top();
                r = end.top();
                if (l < r) {
                        p = partition(arr, l, r);
                        start.pop();
                        end.pop();
                        start.push(p+1);
                        end.push(r);
                        start.push(l);
                        end.push(p-1);
                }
                else {
                        start.pop();
                        end.pop();
                }
        }
}
```

3 • Reply • Share ›

aj006 → Abhishek K Das • 3 months ago

Please provide the partition routine.

• Reply • Share ›

Cam • 8 months ago

This was really helpful. Thanks mate.

^ | v • Reply • Share ›

DS+Algo=Placement • 9 months ago

GeeksforGeeks and others
Please explain these:
1. To reduce the recursion depth, recur first for the smaller half of the array, and use a tail call to recurse into the other.

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

1 ^ | v • Reply • Share ›

Ashish Thakran • 9 months ago

Quicksort is slightly sensitive to input that happens to be in the right order, in which case it can skip some swaps. Mergesort doesn't have any such optimizations, which also makes Quicksort a bit faster compared to Mergesort.

Below link can be useful to find out the more difference and to know more about quicksort and mergesort

Why Quick sort is better than Merge sort

^ | v • Reply • Share ›

MK • 10 months ago

What would be the complexity of iterative version of quick sort? How to calculate it?

^ | v • Reply • Share ›

code_on • a year ago

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

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

Please elaborate.
Thanks

4 ^ | v • Reply • Share ›

Harman • a year ago

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

http://geeksprogrammings.blogs...

1 ^ | v • Reply • Share ›

Nizamuddin Saifi • a year ago

We can improve performance of partition function . by simple checking the a[i]!=a[++i] . it'll

We can improve performance of partition function , by simple checking the a[j]!=a[++i] , it'll reduce number of time 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]);="" }="" }="" swap(&
```

2 • Reply • Share ›

VIGY • 2 years ago

THANK YOU SOOOOO MUCH :-)

• Reply • Share ›

GeeksforGeeks • 2 years ago

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

• Reply • Share ›

Sumit Gera • 2 years ago

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

• Reply • Share ›

kapil • 2 years ago

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

1 • Reply • Share ›

??? • 2 years ago

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

2 • Reply • Share ›

Aashish Barnwal • 2 years ago

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

1 • Reply • Share ›

Kuldeep Tiwari • 2 years ago

With iterative implementation, we don&#039t need parameters low(l) and high(h) in method Quicksort. Method signature reduces to -
void quickSortIterative (int arr[]).

• Reply • Share ›

kuldeep • 2 years ago

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

• Reply • Share ›

Aashish ➦ kuldeep • 2 years ago

The boundaries high(h) and low(l) are not necessary to specify. However, the size of the array to be sorted is a must to pass as a parameter.

• Reply • Share ›

Anonymous • 2 years ago

Quick Sort with (Stable+ Efficient+ in-place Sorting+ NO Need of partition method) which works for all cases 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

6 • Reply • Share ›

Joey ➦ Anonymous • 4 months ago

This is not Iterative too!

• Reply • Share ›

sindhu ➦ Anonymous • 2 years 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 • 2 years 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 working fine.

• Reply • Share ›

Rahul ➦ Anonymous • 2 years ago

You are basically implementing the partition method inside the quickSort method itself. In no way are you improving quick sort. You are just writing the code in one function.

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

• Reply • Share ›

Anonymous ➦ Rahul • 2 years ago

@Rahul- Yes. I am implementing partition code inside one function only. I have NOT improved the efficiency. Efficiency can be improved by choosing the pivot values correctly which can done if we use a randomize function which can randomly distribute the elements of input array and then applying quicksort on new randomized distributed input array.

• Reply • Share ›

Siva Krishna • 2 years ago

very nice one

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

• Reply • Share ›

kaur • 2 years ago

awesome!! :) :)

• Reply • Share ›

Subscribe Add Disqus to your site Privacy

- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

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

Recent Comments

- lt_k

  i need help for coding this function in java...

  Java Programming Language · 1 hour ago

- Piyush

  What is the purpose of else if (recStack[*i])...

Detect Cycle in a Directed Graph · 1 hour ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 2 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 2 hours ago

-