GeeksQuiz

Computer science mock tests for geeks

QuickSort

Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.
1) Always pick first element as pivot.
2) Always pick last element as pivot (implemented below)
3) Pick a random element as pivot.
4) Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Partition Algorithm
There can be many ways to do partition, following code adopts the method given in CLRS book. The logic is simple, we start from the leftmost element and keep track of index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

Implementation:
Following is C++ implementation of QuickSort.

```
/* A typical recursive 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 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];    // pivot
    int i = (l - 1);  // Index of smaller element

    for (int j = l; j <= h- 1; j++)
    {
        // If current element is smaller than or equal to pivot
        if (arr[j] <= x)
        {
            i++;    // increment index of smaller element
            swap(&arr[i], &arr[j]);  // Swap current element with index
        }
    }
    swap(&arr[i + 1], &arr[h]);
    return (i + 1);
}

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

/* 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");
}

// Driver program to test above functions
int main()
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    quickSort(arr, 0, n-1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Output:

```
Sorted array:
1 5 7 8 9 10
```

Analysis of QuickSort

Time taken by QuickSort in general can be written as following.

```
T(n) = T(k) + T(n-k-1) + θ(n)
```

The first two terms are for two recursive calls, the last term is for the partition process. k is the number of elements which are smaller than pivot.
The time taken by QuickSort depends upon the input array and partition strategy. Following are three cases.

Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.

```
T(n) = T(0) + T(n-1) + θ(n)
which is equivalent to
T(n) = T(n-1) + θ(n)
```

The solution of above recurrence is $\theta(n^2)$.

Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.

```
T(n) = 2T(n/2) + θ(n)
```

The solution of above recurrence is θ(nLogn). It can be solved using case 2 of Master Theorem.

Average Case:
To do average case analysis, we need to consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy.
We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set. Following is recurrence for this case.

```
T(n) = T(n/9) + T(9n/10) + θ(n)
```

Solution of above recurrence is also O(nLogn)

Although the worst case time complexity of QuickSort is $O(n^2)$ which is more than many other sorting algorithms like Merge Sort and Heap Sort, QuickSort is faster in practice, because its inner loop can be efficiently implemented on most architectures, and in most real-world data. QuickSort can be implemented in different ways by changing the choice of pivot, so that the worst case rarely occurs for a given type of data. However, merge sort is generally considered better when data is huge and stored in external storage.

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

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

Category: Searching and Sorting

Next i=1 and j=2 . So 7 and 1 will swap, the result now is 21768534.
The if conditions will not satisfy till j is at 6 so again i will increase to 2. 3 and 7 will swap positions so the result will be 21368574.
We come out of for loop as the condition fails now, we swap i+1(2+1=3) we finally swap 6(at position 4) with the last element i.e, 4. So finally our result is 21348576.

• 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

http://newtechnobuzzz.blogspot...

2 • Reply • Share ›

Aditya Goel ➦ Ashish Thakran • 5 months ago

I don't inderstand. How Quicksort is slightly sensitive to input that happens to be in the right order? We have not placed any such optimization in our code. A Program does what we make them do, not the other way around!

1 • Reply • Share ›

Guest • 18 days ago

If anyone looking for a simpler approach then here it is:

```
void quicksort(int array[], int firstIndex, int lastIndex)
{
     int pivotIndex, temp, index1, index2;

    if(firstIndex < lastIndex)
    {
        pivotIndex = lastIndex;
        index1 = firstIndex;
        index2 = lastIndex;

        while(index1 < index2)
        {
            while(array[index1] <= array[pivotIndex] && index1 < lastIndex)
            {
                index1++;
            }
```

```
while(array[index2]>array[pivotIndex])
{
    index2--;
}

if(index1<index2) {="" temp="array[index1];" array[index1]="array[index2];
```

• Reply • Share ›

debugger • 21 days ago

Can be optimised by avoiding unnecessary swaps when all the elements to the left are already smaller than pivot,i.e., when i ==j in partition()

• Reply • Share ›

Hitesh Lalwani • 4 months ago

I didn't get this particular section under average time case
"We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set. Following is recurrence for this case.
T(n) = T(n/9) + T(9n/10) + (n)
"

should this be n/10 in place of n/9 ?

• Reply • Share ›

eknoor ↗ Hitesh Lalwani • a month ago

You are right

• Reply • Share ›

Mahesh ↗ Hitesh Lalwani • 3 months ago

Yes, even i think the same - T(n) = T(n/10) + T (9n/10) + O(n)

• Reply • Share ›

shubhi • 9 months ago

@geeksforgeeks In the main()--when calling quickSort(arr, 0, n-1); ---it should be called as quickSort(arr, 1, n-1);----because we make sure that element at i=0 position is replaced by making i=l-1; So plz check once carefully

• Reply • Share ›

shubhi ↗ shubhi • 9 months ago

@GeeksforGeeks sorry it should be--- In the main()--when calling quickSort(arr, 0, n-1); ---it should be called as quickSort(arr, 1, n);

• Reply • Share ›

DS+Algo=Placement • 9 months ago

this is also in-place sorting??

• Reply • Share ›

Shubham Gupta DS+Algo=Placement • 8 months ago

yes, it an inplace sorting algorithm.
If you dont know the meaning of inplace then it means whether you are using extra space to sort the given array or not. If you are using extra space then it is not an inplace algo like mergesort. and if it doesnt use extra space then it is inplace algo.

• Reply • Share ›

sooraj • a year ago

```
//try this taking pivot as first
#include<stdio.h>
#include<conio.h>
int partition(int arr[],int s,int l){
int left,right,pivot,temp;
left=s;
right=l;
pivot=left;
while(left!=right||right!=pivot){
if(pivot==left){
if(arr[left]>arr[right]){
temp=arr[left];
arr[left]=arr[right];
arr[right]=temp;
pivot=right;
left++;
}
else if(arr[left]<arr[right]){ right--;="" }="" }="" else{="" if(arr[left]="">arr[right]){
```

see more

• Reply • Share ›

Gopal Shankar • a year ago

I see that if input is a sorted array, the above program almost swaps every element with itself. Please check. I prefer the partition function in answer section of http://stackoverflow.com/quest...

• Reply • Share ›

Kartik Gopal Shankar • a year ago

Yes, it seems to be doing. This is standard algorithm from CLRS book.

• Reply • Share ›