GeeksQuiz

Computer science mock tests for geeks

Binary Insertion Sort

We can use binary search to reduce the number of comparisons in normal insertion sort. Binary Insertion Sort find use binary search to find the proper location to insert the selected item at each iteration.
In normal insertion, sort it takes O(i) (at ith iteration) in worst case. we can reduce it to O(logi) by using binary search.

```
// C program for implementation of binary insertion sort
#include <stdio.h>

// A binary search based function to find the position
// where item should be inserted in a[low..high]
int binarySearch(int a[], int item, int low, int high)
{
    if (high <= low)
        return (item > a[low])?  (low + 1): low;

    int mid = (low + high)/2;

    if(item == a[mid])
        return mid+1;

    if(item > a[mid])
        return binarySearch(a, item, mid+1, high);
    return binarySearch(a, item, low, mid-1);
}

// Function to sort an array a[] of size 'n'
void insertionSort(int a[], int n)
{
    int i, loc, j, k, selected;

    for (i = 1; i < n; ++i)
    {
        j = i - 1;
        selected = a[i];

        // find location where selected sould be inseretd
        loc = binarySearch(a, selected, 0, j);

        // Move all elements after location to create space
        while (j >= loc)
        {
            a[j+1] = a[j];
            j--;
        }
```

```
        a[j+1] = selected;
    }
}

// Driver program to test above function
int main()
{
    int a[] = {37, 23, 0, 17, 12, 72, 31,
               46, 100, 88, 54};
    int n = sizeof(a)/sizeof(a[0]), i;

    insertionSort(a, n);

    printf("Sorted array: \n");
    for (i = 0; i < n; i++)
        printf("%d ",a[i]);

    return 0;
}
```

Output:

```
Sorted array:
0 12 17 23 31 37 46 54 72 88 100
```

Time Complexity: The algorithm as a whole still has a running worst case running time of O(n2) because of the series of swaps required for each insertion.

This article is contributed by **Amit Auddy**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Category: Algorithms Searching and Sorting

comparisons. The program is unnecessary creating an overhead of O(logn). Standard Insertion sort will be faster.

1 ▲ | ▼ • Reply • Share ›

creeping_death ➦ Aditya Goel • a month ago

I think this particular implementation is actually worse than standard insertion sort, because of the memory overhead needed for recursive binary search

1 ▲ | ▼ • Reply • Share ›

RK • 3 months ago

The tightest upper bound for time complexity is still O(n^2). How is this any better than the usual insertion sort? I don't think that it reduces the constant factor or lower order terms of the time complexity either.

▲ | ▼ • Reply • Share ›

Morphine • 4 months ago

using binary search to reduce the number of comparisons, isnt the complexity is O(n log n)

▲ | ▼ • Reply • Share ›

Aditya Goel ➦ Morphine • 3 months ago

Binary search just gives us the location where selectedshould be inseretd. We still have to move all elements after location to create space. That can take O(n) in worst case.

▲ | ▼ • Reply • Share ›

gooseberry • 9 months ago

I think there is a problem with the while loop used in function insertionSort.
The while loop continues endlessly. there should be a lower check for j >=0.

Please correct me if I am wrong.

▲ | ▼ • Reply • Share ›

GeeksforGeeks Mod ➦ gooseberry • 9 months ago

Please take a closer look. loc is a value returned by binary search and binary search returns an index.

If you still feel it's incorrect, please provide an example array for which it fails.

▲ | ▼ • Reply • Share ›

Zsw-seu ➦ GeeksforGeeks • 2 months ago

I think there is a problem with binarySearch. Example is:

1,2,3,4,5,7

and I want to insert 7,then

the binarySearch will not be Stable.But the normal Insertion Sort will be Stable.
So I think the function binarySearch should be:

```
int binarySearch(int a[], int item, int low, int high)
{
if (high <= low)
return (item< a[low])? low : (low+1);

int mid = (low + high)/2;

if(item == a[mid])
return mid+1;

if(item > a[mid])
return binarySearch(a, item, mid+1, high);
return binarySearch(a, item, low, mid-1);
}
```